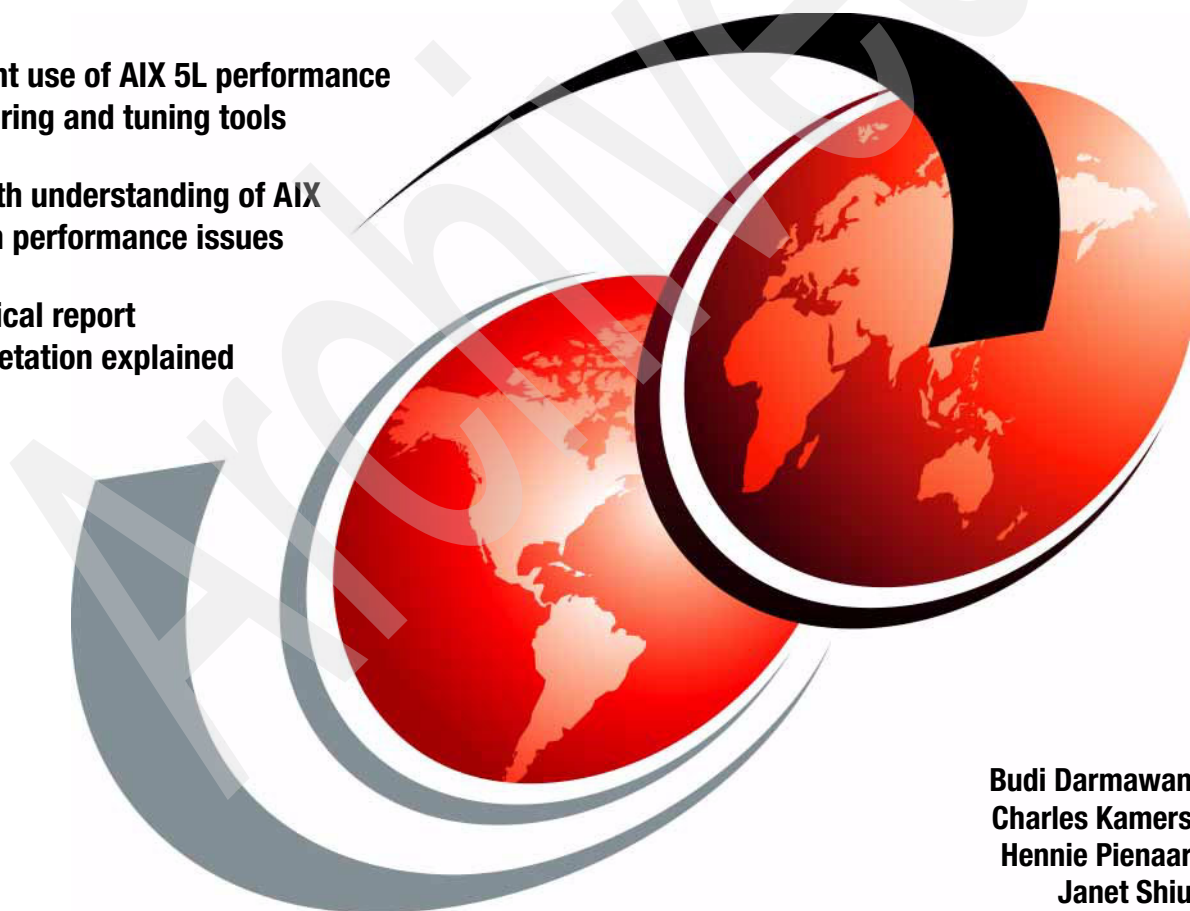


AIX 5L Performance Tools Handbook

Efficient use of AIX 5L performance
monitoring and tuning tools

In-depth understanding of AIX
system performance issues

Statistical report
interpretation explained



Budi Darmawan
Charles Kamers
Hennie Pienaar
Janet Shiu



International Technical Support Organization

AIX 5L Performance Tools Handbook

August 2003

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xxi.

Second Edition (August 2003)

This edition applies to Version 5, Release 2 of AIX 5L.

© Copyright International Business Machines Corporation 2001, 2003. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

Contents

Figures	xvii
Tables	xix
Notices	xxi
Trademarks	xxii
Preface	xxiii
The team that wrote this redbook	xxiv
Become a published author	xxv
Comments welcome	xxv
Summary of changes	xxvii
August 2003, Second Edition	xxvii
Part 1. AIX 5L performance tools	1
Chapter 1. Introduction to AIX performance monitoring and tuning	3
1.1 Performance expectation	4
1.2 CPU performance	5
1.2.1 Initial advice	5
1.2.2 Processes and threads	6
1.2.3 Scheduling	7
1.2.4 SMP performance	9
1.3 Memory performance	12
1.3.1 Initial advice	12
1.3.2 Memory segments	13
1.3.3 Paging mechanism	14
1.3.4 Memory load control mechanism	15
1.3.5 Paging space allocation policies	15
1.3.6 Memory leaks	17
1.3.7 Shared memory	17
1.4 Disk I/O performance	18
1.4.1 Initial advice	18
1.4.2 Disk subsystem design approach	19
1.4.3 Bandwidth-related performance considerations	19
1.4.4 Disk design	20
1.4.5 Logical Volume Manager concepts	24
1.5 Network performance	31

1.5.1	Initial advice	31
1.5.2	TCP/IP protocols	33
1.5.3	Network tunables	34
1.6	Kernel tunables	43
1.6.1	Tunables commands	43
1.6.2	Tunable files	45
1.7	The /proc file system	46
	Chapter 2. Getting started	53
2.1	Tools and filesets	54
2.2	Tools by resource matrix	57
2.3	Performance tuning approach	60
2.3.1	CPU bound system	60
2.3.2	Memory bound system	62
2.3.3	Disk I/O bound system	63
2.3.4	Network I/O bound system	65
	Part 2. Multi-resource monitoring and tuning tools	67
	Chapter 3. The fdpr command	71
3.1	fdpr	72
3.1.1	Information about measurement and sampling	75
3.2	Examples for fdpr	76
	Chapter 4. The iostat command	81
4.1	iostat	82
4.1.1	Information about measurement and sampling	83
4.2	Examples for iostat	83
4.2.1	System throughput report	84
4.2.2	tty and CPU utilization report	88
4.2.3	Disk utilization report	89
4.2.4	Disk utilization report for MPIO	90
4.2.5	Adapter throughput report	91
	Chapter 5. The netpmn command	93
5.1	netpmn	94
5.1.1	Information about measurement and sampling	95
5.2	Examples for netpmn	96
5.2.1	Process statistics	99
5.2.2	FLIH and SLIH CPU statistics	101
5.2.3	TCP socket call statistics	102
5.2.4	Detailed statistics	103
	Chapter 6. Performance Diagnostic Tool (PDT)	105

6.1	PDT	106
6.1.1	Information about measurement and sampling	106
6.2	Examples for PDT	106
6.2.1	Editing the configuration files	108
6.2.2	Using reports generated by PDT	111
6.2.3	Creating a PDT report manually	114
Chapter 7. The perfpmr command		115
7.1	perfpmr	116
7.1.1	Information about measurement and sampling	116
7.1.2	Building and submitting a test case	120
7.2	Examples for perfpmr	124
Chapter 8. The ps command		127
8.1	ps	128
8.1.1	Information about measurement and sampling	130
8.2	Examples for ps	131
8.2.1	Displaying the top 10 CPU-consuming processes	131
8.2.2	Displaying the top 10 memory-consuming processes	132
8.2.3	Displaying the processes in order of being penalized	133
8.2.4	Displaying the processes in order of priority	134
8.2.5	Displaying the processes in order of nice value	134
8.2.6	Displaying the processes in order of real memory use	135
8.2.7	Displaying the processes in order of I/O	136
8.2.8	Displaying WLM classes	137
8.2.9	Viewing threads	137
Chapter 9. The sar command		139
9.1	sar	140
9.1.1	Information about measurement and sampling	142
9.2	Examples for sar	142
9.2.1	Monitoring one CPU at a time	142
9.2.2	Collecting statistics by using cron	146
9.2.3	Displaying access time system routines	149
9.2.4	Monitoring buffer activity for transfers, access, and caching	150
9.2.5	Monitoring system calls	151
9.2.6	Monitoring activity for each block device	153
9.2.7	Monitoring kernel process activity	154
9.2.8	Monitoring the message and semaphore activities	155
9.2.9	Monitoring the kernel scheduling queue statistics	156
9.2.10	Monitoring the paging statistics	157
9.2.11	Monitoring the processor utilization	158
9.2.12	Monitoring tty device activity	160
9.2.13	Monitoring kernel tables	160

9.2.14	Monitoring system context switching activity	162
Chapter 10.	The schedo and schedtune commands	165
10.1	schedo	166
10.1.1	Recommendations and precautions	167
10.2	Examples for schedo	168
10.2.1	Displaying current settings	168
10.2.2	Tuning CPU parameters	169
10.2.3	Tuning memory parameters	171
10.3	schedtune	177
Chapter 11.	The topas command	179
11.1	topas	180
11.1.1	Information about measurement and sampling	181
11.2	Examples for topas	181
11.2.1	Common uses of the topas command	181
11.2.2	Using subcommands	185
11.2.3	Monitoring CPU usage	187
11.2.4	Monitoring disk problem	189
Chapter 12.	The truss command	191
12.1	truss	192
12.1.1	Information about measurement and sampling	197
12.2	Examples for truss	197
12.2.1	Using truss	197
12.2.2	Using the summary output	198
12.2.3	Monitoring running processes	200
12.2.4	Analyzing file descriptor I/O	202
12.2.5	Checking program parameters	205
12.2.6	Checking program environment variables	205
12.2.7	Tracking child processes	206
12.2.8	Checking user library call	209
Chapter 13.	The vmstat command	211
13.1	vmstat	212
13.1.1	Information about measurement and sampling	213
13.2	Examples for vmstat	213
13.2.1	Virtual memory activity	213
13.2.2	Forks report	219
13.2.3	Interrupts report	220
13.2.4	VMM statistics report	221
13.2.5	Sum structure report	224
13.2.6	I/O report	226

Chapter 14. The vmo, ioo, and vmtune commands	229
14.1 vmo	230
14.1.1 Information about measurement and sampling	231
14.1.2 Recommendations and precautions for vmo	235
14.2 Examples for vmo	235
14.3 ioo	239
14.3.1 Information about measurement and sampling	241
14.3.2 Recommendations and precautions	246
14.4 Examples for ioo	246
14.4.1 Displaying I/O setting	247
14.4.2 Changing tunable values	249
14.4.3 Logical volume striping	249
14.4.4 Increasing write activity throughput	251
14.5 vmtune	251
Chapter 15. Kernel tunables commands	255
15.1 tuncheck	256
15.1.1 Examples for tuncheck	256
15.2 tunrestore	258
15.2.1 Examples for tunrestore	259
15.3 tunsave	260
15.3.1 Examples for tunsave	261
15.4 tundefault	264
15.4.1 Examples for tundefault	264
15.5 tunchange	265
15.5.1 Examples for tunchange	266
Chapter 16. Process-related commands	267
16.1 procwdx	268
16.2 procfiles	268
16.3 procflags	270
16.4 proccred	270
16.5 procmap	271
16.6 procldd	272
16.7 procsig	273
16.8 procstack	274
16.9 procstop	275
16.10 procrun	275
16.11 procwait	276
16.12 proctree	276
Part 3. CPU-related performance tools	279
Chapter 17. The alstat and emstat commands	281

17.1	Alignment and emulation exception	282
17.2	alstat	283
17.2.1	Information about measurement and sampling	283
17.2.2	Examples for alstat	283
17.2.3	Detecting and resolving alignment problems	285
17.3	emstat	285
17.3.1	Information about measurement and sampling	286
17.3.2	Examples for emstat	286
17.3.3	Detecting and resolving emulation problems	288
Chapter 18. The bindintcpu and bindprocessor commands		289
18.1	bindintcpu	290
18.1.1	Examples for bindintcpu	290
18.2	bindprocessor	292
18.2.1	Information about measurement and sampling	292
18.2.2	Examples for bindprocessor	293
Chapter 19. The gprof, pprof, prof, and tprof commands		297
19.1	CPU profiling tools	298
19.1.1	Comparison of tprof versus prof and gprof	299
19.2	gprof	300
19.2.1	Information about measurement and sampling	301
19.2.2	Profiling with the fork and exec subroutines	301
19.2.3	Examples for gprof	302
19.3	pprof	309
19.3.1	Information about measurement and sampling	309
19.3.2	Examples for pprof	311
19.4	prof	320
19.4.1	Information about measurement and sampling	321
19.4.2	Examples for prof	322
19.5	tprof	324
19.5.1	Information about measurement and sampling	326
19.5.2	Examples for tprof	329
Chapter 20. The nice and renice commands		349
20.1	nice	350
20.1.1	Information about measurement and sampling	350
20.2	Examples for nice	351
20.2.1	Reducing the priority of a process	352
20.2.2	Improving the priority of a process	352
20.3	renice	352
20.3.1	Information about measurement and sampling	353
20.4	Examples for renice	353

Chapter 21. The time and timex commands	355
21.1 time	356
21.1.1 Information about measurement and sampling	356
21.1.2 Examples for time	356
21.2 timex	357
21.2.1 Information about measurement and sampling	357
21.2.2 Examples for timex	358
Part 4. Memory-related performance tools	363
Chapter 22. The ipcs command	365
22.1 ipcs	366
22.1.1 Information about measurement and sampling	366
22.1.2 Examples for ipcs	366
Chapter 23. The rmss command	379
23.1 rmss	380
23.1.1 Information about measurement and sampling	381
23.1.2 Recommendations and precautions	382
23.1.3 Examples for rmss	382
Chapter 24. The svmon command	387
24.1 svmon	388
24.1.1 Information about measurement and sampling	391
24.1.2 Examples for svmon	393
Part 5. Disk I/O-related performance tools	455
Chapter 25. The filemon command	457
25.1 filemon	458
25.1.1 Information about measurement and sampling	459
25.1.2 Examples for filemon	462
Chapter 26. The fileplace command	479
26.1 fileplace	480
26.1.1 Information about measurement and sampling	480
26.1.2 Examples for fileplace	481
26.1.3 Analyzing the physical report	483
Chapter 27. The lslv, lspv, and lsvg commands	501
27.1 lslv	502
27.2 lspv	502
27.3 lsvg	503
27.4 Examples for lslv, lspv, and lsvg	505
27.4.1 Using lslv	512

27.4.2	Using lspv	513
27.4.3	Using lsvg	515
27.4.4	Acquiring more disk information	516
Chapter 28.	The lvmstat command	519
28.1	lvmstat	520
28.1.1	Information about measurement and sampling	520
28.1.2	Examples for lvmstat	521
Part 6.	Network-related performance tools	531
Chapter 29.	atmstat, entstat, estat, fddistat, and tokstat commands	539
29.1	atmstat	540
29.1.1	Information about measurement and sampling	540
29.1.2	Examples for atmstat	541
29.2	entstat	546
29.2.1	Information about measurement and sampling	547
29.2.2	Examples for entstat	548
29.3	estat	552
29.3.1	Information about measurement and sampling	552
29.3.2	Examples for estat	552
29.4	fddistat	555
29.4.1	Information about measurement and sampling	556
29.4.2	Examples for fddistat	556
29.5	tokstat	560
29.5.1	Information about measurement and sampling	561
29.5.2	Examples for tokstat	562
Chapter 30.	TCP/IP packet tracing tools	567
30.1	Network packet tracing tools	568
30.2	iptrace	569
30.2.1	Information about measurement and sampling	571
30.3	ipreport	572
30.3.1	Information about measurement and sampling	573
30.4	ipfilter	573
30.4.1	Information about measurement and sampling	574
30.4.2	Protocols and header type options	574
30.5	Examples for iptrace, ipreport, and ipfilter	574
30.5.1	TCP packets	575
30.5.2	UDP packets	576
30.5.3	UDP domain name server requests and responses	577
30.6	Examples for ipreport	578
30.6.1	Using ipreport with tcpdump	578
30.6.2	Using ipreport with iptrace	579

30.7	Examples for ipfilter	582
30.7.1	Tracing TCP/IP traffic	583
30.7.2	NFS tracing	584
30.7.3	TCP tracing	585
30.7.4	UDP tracing	586
30.7.5	ICMP tracing	586
30.7.6	IPX tracing	586
30.7.7	ALL protocol tracing	587
30.8	tcpdump.	587
30.8.1	Information about measurement and sampling.	589
30.9	Examples for tcpdump.	594
30.10	trpt.	612
30.10.1	Information about measurement and sampling.	613
30.11	Examples for trpt.	613
30.11.1	Displaying all stored trace records	614
30.11.2	Displaying source and destination addresses.	615
30.11.3	Displaying packet-sequencing information	616
30.11.4	Displaying timers at each point in the trace	616
30.11.5	Printing trace records for a single protocol control block	617
Chapter 31. The netstat command.		619
31.1	netstat	620
31.1.1	Information about measurement and sampling.	622
31.1.2	Examples for netstat	624
Chapter 32. The nfso command.		645
32.1	nfso	646
32.1.1	Information about measurement and sampling.	648
32.2	Examples for nfso	648
32.2.1	Listing all of the tunables and their current values	648
32.2.2	Displaying characteristics of all tunables	649
32.2.3	Displaying and changing a tunable with the nfso command	651
32.2.4	Resetting a tunable value to its default	652
32.2.5	Displaying help information about a tunable	652
32.2.6	Permanently changing an nfso tunable.	652
32.2.7	Changing a tunable after reboot	653
Chapter 33. The nfsstat command.		655
33.1	nfsstat	656
33.1.1	Information about measurement and sampling.	656
33.2	Examples for nfsstat	657
33.2.1	NFS server RPC statistics.	657
33.2.2	NFS server NFS statistics.	659
33.2.3	NFS client RPC statistics	660

33.2.4 NFS client NFS statistics	661
33.2.5 Statistics on mounted file systems	662
Chapter 34. The no command	665
34.1 no	666
34.2 Examples for no	667
Part 7. Tracing performance problems	675
Chapter 35. The curt command	677
35.1 curt	678
35.1.1 Information about measurement and sampling	679
35.2 Examples for curt	680
Chapter 36. The gennames, genld, genkld, genkex, and gensyms commands	703
36.1 Offline generation tools	704
36.2 gennames	704
36.2.1 Information about measurement and sampling	705
36.2.2 Examples for gennames	705
36.3 genld	710
36.3.1 Information about measurement and sampling	710
36.3.2 Examples for genld	710
36.4 genkld	712
36.4.1 Information about measurement and sampling	712
36.4.2 Examples for genkld	712
36.5 genkex	713
36.5.1 Information about measurement and sampling	713
36.5.2 Examples for genkex	713
36.6 gensyms	715
36.6.1 Information about measurement and sampling	715
36.6.2 Examples for gensyms	715
Chapter 37. The locktrace command	719
37.1 locktrace	720
37.1.1 Information about measurement and sampling	720
37.1.2 Examples for locktrace	721
Chapter 38. The stripnm command	723
38.1 stripnm	724
38.1.1 Information about measurement and sampling	724
38.2 Examples for stripnm	725
Chapter 39. The splat command	729
39.1 splat	730

39.1.1	Information about measurement and sampling	732
39.2	Examples for splat	735
39.2.1	Execution summary	735
39.2.2	Gross lock summary	736
39.2.3	Per-lock summary	737
39.2.4	AIX kernel lock details	739
39.2.5	PThread synchronizer reports	749
Chapter 40.	The trace, trcnm, and trcrpt commands	759
40.1	trace	760
40.1.1	Information about measurement and sampling	764
40.1.2	Terminology used for trace	765
40.1.3	Ways to start and stop trace	767
40.1.4	Examples for trace	770
40.2	trcnm	775
40.2.1	Information about measurement and sampling	776
40.2.2	Examples for trcnm	776
40.3	trcrpt	777
40.3.1	Information about measurement and sampling	781
40.3.2	Examples for trcrpt	781
Part 8.	Additional performance topics	783
Chapter 41.	APIs for performance monitoring	785
41.1	Perfstat API	786
41.1.1	Compiling and linking	786
41.1.2	Subroutines	787
41.2	System Performance Measurement Interface	805
41.2.1	Compiling and linking	806
41.2.2	SPMI data organization	806
41.2.3	Subroutines	808
41.2.4	Examples for SPMI	813
41.3	Performance Monitor API	818
41.3.1	Performance Monitor data access	819
41.3.2	Compiling and linking	820
41.3.3	Subroutines	820
41.3.4	Examples for PM API	820
41.4	Resource Monitoring and Control	824
41.4.1	RMC commands	825
41.4.2	Information about measurement and sampling	826
41.4.3	Examples for RMC	828
41.5	Miscellaneous performance monitoring subroutines	842
41.5.1	Compiling and linking	842
41.5.2	Subroutines	842

41.5.3 Combined example	858
Chapter 42. Workload Manager tools	861
42.1 WLM tools overview	862
42.2 wlmstat	862
42.2.1 Information about measurement and sampling	864
42.2.2 Examples for wlmstat	865
42.3 wlmmon / wlmperf	872
42.3.1 Information about the xmwlm and xmtrend daemons	873
42.3.2 Information about measurement and sampling	875
42.3.3 Exploring the graphical windows	875
Chapter 43. Performance Toolbox Version 3 for AIX	891
43.1 Introduction	892
43.2 xmperf	894
43.2.1 Information about measurement and sampling	897
43.2.2 Examples	904
43.3 3D monitor	909
43.3.1 Information about measurement and sampling	912
43.3.2 Examples	915
43.4 jazizo	918
43.4.1 Syntax of xmtrend	918
43.4.2 Syntax of jazizo	919
43.4.3 Information about measurement and sampling	919
Part 9. Appendices	933
Appendix A. Source code examples	935
perfstat_dump_all.c	936
perfstat_dude.c	940
spmi_dude.c	949
spmi_data.c	953
spmi_file.c	959
spmi_traverse.c	961
dudestat.c	965
cwhet.c	968
Appendix B. Trace hooks	973
AIX 5L trace hooks	974
Abbreviations and acronyms	983
Related publications	987
IBM Redbooks	987

Other publications	987
Online resources	988
How to get IBM Redbooks	989
Index	991

Archived

Archived

Figures

1-1	Physical partition mapping	25
14-1	Sequential read-ahead.	242
30-1	Schematic flow during TCP open.	598
30-2	Schematic flow during TCP close	599
39-1	Lock states	742
40-1	The trace facility	766
42-1	Initial screen when wlmperf and wlmmon are started	876
42-2	The WLM_Console tab down menu.	877
42-3	The open log option from the tab down bar	877
42-4	The WLM table visual report	878
42-5	The CPU, memory, and disk I/O tab down menu	878
42-6	The bar-graph-style visual report.	880
42-7	The order of the snapshot visual report colored bulbs.	880
42-8	The snapshot visual report.	881
42-9	The Selected tab down menu	881
42-10	The time window for setting trend periods	882
42-11	The table visual report with trend values shown	883
42-12	The bar-graph style report showing a trend.	884
42-13	The snapshot visual report showing the trend	885
42-14	Advanced option under the Selected tab down menu	886
42-15	The Advanced Menu options shown in graphical form	887
42-16	The class/tier option from the selected tab down menu.	888
42-17	The snapshot report showing only the Red WLM class.	889
43-1	The initial xmpperf window.	898
43-2	The Mini Monitor window	898
43-3	Aged data moved to the left.	899
43-4	The Utilities tab down menu.	900
43-5	The Analysis tab down menus	901
43-6	The Controls tab down menu.	901
43-7	The Recording tab down menu	901
43-8	The Console Recording options.	902
43-9	Cautionary window when recording an instrument	902
43-10	Console Recording tab down menu: End Recording option	902
43-11	Options under the initial xmpperf window File tab down menu	903
43-12	The Playback window	903
43-13	The playback monitor.	904
43-14	Naming the user-defined console	904
43-15	Choose the Edit Console menu	905

43-16 Dynamic Data Supplier Statistics window	905
43-17 The Change Properties of a Value window	906
43-18 The final console monitoring CPU idle time	907
43-19 The Edit Console tab down menu	907
43-20 The Modify Instrument menu options.	908
43-21 The Style & Stacking menu option.	908
43-22 Menu options from the Edit Value tab down menu	908
43-23 An example of a CPU usage instrument	909
43-24 Initial 3dmon screen.	913
43-25 3-D window from 3dmon showing the statistics of a host	914
43-26 CPU statistics displayed by 3dmon after modifying 3dmon.cf	916
43-27 3dmon graph showing disk activity for multiple hosts	917
43-28 The jazizo opening window	921
43-29 The File tab down menu	921
43-30 The Open Recording File window in jazizo	922
43-31 Metric Selection window	923
43-32 The Metric Selection window showing metric selections	924
43-33 The Time Selection window	925
43-34 The Stop Hour and Start Hour tab down menus	925
43-35 Adjusting the month in the jazizo Time Selection window	926
43-36 Adjusting days in the jazizo Time Selection window	926
43-37 The jazizo window	927
43-38 The jazizo Edit tab down menu	928
43-39 The Graph Selection window of the jazizo program	928
43-40 The trend of the metric can be displayed by jazizo	929
43-41 The View tab down menu	930
43-42 The Report tab down menu	930
43-43 Tabular statistical output that can be obtained from jazizo	930
43-44 The File tab down menu when closing jazizo	931

Tables

1-1	TCP/IP layers and protocol examples	34
1-2	Network tunables minimum values for best performance	36
1-3	Other basic network tunables	37
2-1	Commands/tools, pathnames, and filesets	54
2-2	Performance tools by resource matrix	57
7-1	Files created by perfpnr	123
10-1	Current effective priority calculated where sched_R is four.	173
10-2	Current effective priority calculated where sched_R is 16.	174
10-3	The CPU decay factor using the default sched_D value of 16	175
10-4	The CPU decay factor using a sched_D value of 31	175
12-1	Machine faults	194
12-2	Signals	195
30-1	Some important protocols	570
30-2	Selection from /etc/services	570
30-3	ipfilter header types and options	574
34-1	Suggested minimum buffer and MTU sizes for adapters.	671
35-1	Minimum trace hooks required for curt	679
39-1	Trace hooks required for splat	732
39-2	PThread read/write lock report.	752
41-1	Interface types from if_types.h.	801
41-2	Column explanation	813
42-1	Output of winstat -v	867

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™

AIX®

DB2®

ESCON®

@server™

ibm.com®

IBM®

NetView®


Nways®

POWER3™

POWER4™

pSeries™

PTX®

Redbooks (logo) ™

Redbooks™

RS/6000®

Tivoli®

z/OS®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM Redbook takes an insightful look at the performance monitoring and tuning tools that are provided with AIX® 5L™. It discusses the usage of the tools as well as the interpretation of the results by using many examples.

This book is meant as a reference for system administrators and AIX technical support professionals so they can use the performance tools in an efficient manner and interpret the outputs when analyzing an AIX system's performance.

The individual performance tools discussed in this book are organized according to the resources for which they provide information:

- ▶ Part 1, “AIX 5L performance tools” on page 1 introduces the reader to the process of AIX performance analysis.
- ▶ Part 2, “Multi-resource monitoring and tuning tools” on page 67 discusses tools that provide information about multiple resources.
- ▶ Part 3, “CPU-related performance tools” on page 279 discusses tools that provides information about CPU resources.
- ▶ Part 4, “Memory-related performance tools” on page 363 discusses tools that provides information about system memory usage.
- ▶ Part 5, “Disk I/O–related performance tools” on page 455 discusses tools that provides information about disk I/O performance.
- ▶ Part 6, “Network-related performance tools” on page 531 discusses tools that provides information about network monitoring.
- ▶ Part 7, “Tracing performance problems” on page 675 explains AIX trace and trace-related tools that can be used to monitor all system resources.
- ▶ Part 8, “Additional performance topics” on page 783 discusses additional topics such as API for performance monitoring, Workload Manager tools, and performance toolbox for AIX.

The team that wrote this redbook

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.



Budi Darmawan is a Project Leader at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of performance management and database administration. Before joining the ITSO four years ago, Budi worked in the Integrated Technology Services Department of IBM Indonesia as a solution architect. His current interests are performance and availability management, Tivoli® systems management products, z/OS® systems management, and business intelligence.

Charles Kamers is a Technical Support Analyst for HSBC Bank Brazil. He has been worked in technical support, mainly for RS/6000® and pSeries™ servers, since 1999. His areas of expertise include AIX, PSSP, HACMP, Linux, and storage solutions. He holds a degree as a Data Processing Technologist.

Hennie Pienaar is a Senior Education Specialist for IBM South Africa and has worked at IBM for seven years. He has eight years of experience in AIX and UNIX. His areas of expertise include Tivoli, Linux, AIX, and security. He has taught extensively in these areas.

Janet Shiu is an HPC Technical Architect with 20 years of experience in high-performance computing. She holds a Ph.D. in Physics from the University of Pittsburgh. Her areas of expertise include architecting HPC solutions for various industry segments, project management, benchmarking, optimization, and performance tuning.

Thanks to the following people for their contributions to this project:

Keigo Matsubara, Betsy Thaggard
International Technical Support Organization, Austin Center

Luc Smolder
IBM Austin

Diana Gfoerer, Thomas Braunbeck, Stuart Lane, Bjorn Roden, Nigel Trickett
Original authors of the *AIX 5L Performance Tools Handbook*, SG24-6039

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM® Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for *AIX 5L Performance Tools Handbook*, SG24-6039-01
as created or updated on August 11, 2003.

August 2003, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- ▶ Kernel tunables commands that manipulate tunables stanzas for AIX initialization parameters
- ▶ Commands that relate to the /proc filesystems
- ▶ New commands, such as **vmo**, **ioo**, and **schedo**, that replace the **vm tune** and **sched tune** commands

Changed information

- ▶ Additional information about the **tprof** command
- ▶ Various enhancements of performance-related commands
- ▶ Direct link from the list of commands in Chapter 2, “Getting started” on page 53 into the appropriate section

Archived



Part 1

AIX 5L performance tools

This book discusses AIX 5L performance tools and their use. The original edition of this book was developed with AIX 5L Version 5.2 in mind, and we have extended the discussion to include several tool enhancements, such as the tunables and proc filesystem tools.

This project was performed at the ITSO Austin Center using several machines with AIX 5L operating systems. The configurations were:

- ▶ IBM @server pSeries 690, a Regatta server, partitioned with 4 CPUs and 4 GB of memory as our primary test machine. This machine is called *lpar05*.

- ▶ IBM RS/6000 model F80 as our Workload Manager node. This machine is called *wlmhost*.
- ▶ IBM RS/6000 43P, on which we installed AIX 5L version 5.1, as a comparison machine.

All of these machines were connected via an Ethernet LAN environment. We also attached the F80 with token-ring and ATM adapter for the network performance discussion.

The document is divided into these parts:

- ▶ Part 1, “AIX 5L performance tools” on page 1 is the primary discussion about AIX performance concepts and a listing of the available performance tools.
- ▶ Part 2, “Multi-resource monitoring and tuning tools” on page 67 discusses monitoring tools for multiple resources, which are useful in providing an initial indication of a problem and pinpointing the problem area.
- ▶ Specific performance tools for each major resources are discussed in:
 - Part 3, “CPU-related performance tools” on page 279
 - Part 4, “Memory-related performance tools” on page 363
 - Part 5, “Disk I/O-related performance tools” on page 455
 - Part 6, “Network-related performance tools” on page 531
- ▶ Part 7, “Tracing performance problems” on page 675 shows the available tracing tools for debugging performance problems
- ▶ Part 8, “Additional performance topics” on page 783 discusses miscellaneous topics such as:
 - APIs that can be used to create a performance monitoring program
 - Workload Manager tools
 - AIX Performance Toolbox

Introduction to AIX performance monitoring and tuning

The performance of a computer system is based on human expectations and the ability of the computer system to fulfill these expectations. The objective for performance tuning is to match expectations and fulfillment. The path to achieving this objective is a balance between appropriate expectations and optimizing the available system resources. The discussion consists of:

- ▶ What the human perceived as the performance is discussed in 1.1, “Performance expectation” on page 4.
- ▶ What can be actually tuned from the systems is categorized into CPU, memory, disk, and network as discussed in:
 - 1.2, “CPU performance” on page 5
 - 1.3, “Memory performance” on page 12
 - 1.4, “Disk I/O performance” on page 18
 - 1.5, “Network performance” on page 31
- ▶ The new tuning feature for AIX 5L Version 5.2 is discussed in 1.6, “Kernel tunables” on page 43 and 1.7, “The /proc file system” on page 46.

1.1 Performance expectation

The performance tuning process demands great skill, knowledge, and experience, and it cannot be performed by only analyzing statistics, graphs, and figures. The human aspect of perceived performance must not be neglected if results are to be achieved. Performance tuning will also usually have to take into consideration problem determination aspects as well as pure performance issues.

Expectations can often be classified as either:

Throughput expectations Throughput is a measure of the amount of work performed over a period of time.

Response time expectations Response time is the time elapsed between when a request is submitted and when the response from that request is returned.

The performance tuning process can be initiated for a number of reasons:

- ▶ To achieve optimal performance in a newly installed system
- ▶ To resolve performance problems resulting from the design (sizing) phase
- ▶ To resolve performance problems occurring in the runtime (production) phase

Performance tuning on a newly installed system usually involves setting some base parameters for the operating system and applications. The sections in this chapter describe the characteristics of different system resources and provide some advice regarding their base tuning parameters if applicable.

Limitations originating from the sizing phase either limit the possibility of tuning, or incur greater cost to overcome them. The system may not meet the original performance expectations because of unrealistic expectations, physical problems in the computer environment, or human error in the design or implementation of the system. In the worst case adding or replacing hardware might be necessary. It is therefore highly advised to be particularly careful when sizing a system to allow enough capacity for unexpected system loads. In other words, do not design the system to be 100 percent busy from the start of the project. More information about system sizing can be found in the redbook *Understanding IBM @server pSeries Performance and Sizing*, SG24-4810.

When a system in a productive environment still meets the performance expectations for which it was initially designed, but the demands and needs of the utilizing organization have outgrown the system's basic capacity, performance tuning is performed to avoid and/or delay the cost of adding or replacing hardware.

Remember that many performance-related issues can be traced back to operations performed by somebody with limited experience and knowledge who unintentionally restricted some vital logical or physical resource of the system.

1.2 CPU performance

This section gives an overview of the operations of the kernel and CPU. An understanding of the way processes and threads operate within the AIX environment is required to successfully monitor and tune AIX for peak CPU throughput.

Systems that experience performance problems are sometimes constrained less by hardware limitations than by the way applications are written or the way the operating system is tuned. Threads that are waiting on locks can cause a significant degradation in performance.

1.2.1 Initial advice

We recommend that you not make any changes to the CPU scheduling parameters until you have had experience with the actual workload. In some cases the workload throughput can benefit from adjusting the scheduling thresholds. See Chapter 10, “The schedo and schedtune commands” on page 165 for more details about monitoring and changing these values and parameters.

The discussion of the CPU performance and scheduling is divided into:

- ▶ 1.2.2, “Processes and threads” on page 6, which discusses the concepts of threads as the primary processing unit and shows how the priority of a thread is calculated.
- ▶ 1.2.3, “Scheduling” on page 7 discusses various scheduling policies and scheduling-related concepts.
- ▶ 1.2.4, “SMP performance” on page 9 discusses issues related to Symmetrical Multiprocessor (SMP) machines.

For more information about CPU scheduling, refer to:

- ▶ *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*, available online at:
http://publib16.boulder.ibm.com/pseries/en_US/aixbman/admnconc/admnconc.htm
- ▶ *AIX 5L Version 5.2 Performance Management Guide*, available online at:
http://publib16.boulder.ibm.com/doc_link/en_US/a_doc_lib/aixbman/prftungd/prftungd.htm

1.2.2 Processes and threads

The following defines the differences between threads and processes:

Processes	A process is an activity within the system that is started with a command, a shell script, or another process.
Threads	A thread is an independent flow of control that operates within the same address space as other independent flows of controls within a process. A kernel thread is a single sequential flow of control.

Kernel threads are owned by a process. A process has one or more kernel threads. The advantage of threads is that you can have multiple threads running in parallel on different CPUs on an SMP system.

Applications can be designed to have user level threads that are scheduled to work by the application or by the pthreads scheduler in libpthread. Multiple threads of control allow an application to service requests from multiple users at the same time. Application threads can be mapped to kernel threads in a 1:1 or an n:1 relation.

The kernel maintains the priority of the threads. A thread's priority can range from zero to 255. A zero priority is the most favored and 255 is the least favored. Threads can have a fixed or non-fixed priority. The priority of fixed priority threads does not change during the life of the thread, while non-fixed priority threads can have their maximum priority changed by changing its *nice* value with the **nice** or the **renice** commands.

Thread aging

When a thread is created, the CPU usage value is zero. As the thread accumulates more time on the CPU, the usage increments. The CPU usage of a process is initially zero. The CPU usage increases by one after each clock interrupt (every 10 ms) and will increment up to 120, which prevents high CPU usage threads from monopolizing the CPU.

The CPU usage can be shown with the **ps -ef** command, looking at the C column of the output (see 8.2.3, "Displaying the processes in order of being penalized" on page 133).

Every second, the scheduler ages the thread using the following formula:

$$\text{CPU usage} = \text{CPU usage} * (D/32)$$

Where D is the decay value as set by **schedo -o sched_D**

If the D parameter is set to 32, the thread usage will not decrease. The default of 16 will enable the thread usage to decrease, giving it more time on the CPU.

Calculating thread priority

The kernel calculates the priority for non-fixed priority threads using a formula that includes the following:

base priority	The <i>base priority</i> of a thread is 40.
nice value	The <i>nice value</i> defaults to 20 for foreground processes and 24 for background processes. This can be changed using the nice or renice command. See Chapter 20, “The nice and renice commands” on page 349.
r	The <i>CPU penalty factor</i> . The default for <i>r</i> is 16. This value can be changed with the schedo command.
D	The <i>CPU decay factor</i> . The default for <i>D</i> is 16. This value can be changed with the schedo command.
C	CPU usage as discussed in “Thread aging” on page 6.
p_nice	This is called the <i>niced priority</i> . It is calculated as from: $p_nice = \text{base priority} + \text{nice value}$
x_nice	The “ <i>extra nice</i> ” value. If the <i>niced priority</i> for a thread (<i>p_nice</i>) is larger than 60, then the following formula applies: $x_nice = p_nice * 2 - 60$ If the <i>niced priority</i> for a thread (<i>p_nice</i>) is equal or less than 60, the following formula applies: $x_nice = p_nice$
X	The <i>xnice factor</i> is calculated as: $(x_nice + 4) / 64$.

The thread priority is finally calculated based on the following formula:

$$\text{Priority} = (C * r/32 * X) + x_nice$$

Using this calculation method, note the following:

- ▶ With the default nice value of 20, the xnice factor is 1, no affect to the priority. When the nice value is bigger than 20, it had greater effect on the x_nice compared to the lower nice value.
- ▶ Smaller values of r reduce the impact of CPU usage to the priority of a thread; therefore the nice value has more of an impact on the system.

1.2.3 Scheduling

The following scheduling policies apply to AIX:

SCHED_RR	The thread is time-sliced at a fixed priority. If the thread is still running when the time slice expires, it is moved to the end of the queue of dispatchable threads. The queue the
-----------------	---

	thread will be moved to depends on its priority. Only root can schedule using this policy.
SCHED_OTHER	This policy only applies to non-fixed priority threads that run with a time slice. The priority gets recalculated at every clock interrupt. This is the default scheduling policy.
SCHED_FIFO	This is a non-preemptive scheduling scheme except for higher priority threads. Threads run to completion unless they are blocked or relinquish the CPU of their own accord. Only fixed priority threads use this scheduling policy. Only root can change the scheduling policy of threads to use SCHED_FIFO.
SCHED_FIFO2	Fixed priority threads use this scheduling policy. The thread is put at the head of the run queue if it was only asleep for a short period of time.
SCHED_FIFO3	Fixed priority threads use this scheduling policy. The thread is put at the head of the run queue whenever it becomes runnable, but it can be preempted by a higher priority thread.

The following section describes important concepts in scheduling.

Run queues

Each CPU has a dedicated run queue. A run queue is a list of runnable threads, sorted by thread priority value. There are 256 thread priorities (zero to 255). There is also an additional global run queue where new threads are placed.

When the CPU is ready to dispatch a thread, the global run queue is checked before the other run queues are checked. When a thread finishes its time slice on the CPU, it is placed back on the runqueue of the CPU it was running on. This helps AIX to maintain processor affinity. To improve the performance of threads that are running with SCHED_OTHER policy and are interrupt driven, you can set the environmental variable called RT_GRQ to ON. This will place the thread on the global run queue. Fixed priority threads will be placed on the global run queue if you run `schedo -o fixed_pri_global=1`.

Time slices

The CPUs on the system are shared among all of the threads by giving each thread a certain slice of time to run. The default time slice of one clock tick (10 ms) can be changed using `schedo -o timeslice`. Sometimes increasing the time slice improves system throughput due to reduced context switching. The `vmstat` and `sar` commands show the amount of context switching. In a high value of context switches, increasing the time slice can improve performance. This parameter should, however, only be used after a thorough analysis.

Mode switching

There are two modes that a CPU operates in: *kernel mode* and *user mode*. In user mode, programs have read and write access to the user data in the process private region. They can also read the user text and shared text regions, and have access to the shared data regions using shared memory functions. Programs also have access to kernel services by using system calls.

Programs that operate in kernel mode include interrupt handlers, kernel processes, and kernel extensions. Code operating in this mode has read and write access to the global kernel address space and to the kernel data in the process region when executing within the context of a process. User data within the process address space must be accessed using kernel services.

When a user program access system calls, it does so in kernel mode. The concept of user and kernel modes is important to understand when interpreting the output of commands such as **vmstat** and **sar**.

1.2.4 SMP performance

In an SMP system, all of the processors are identical and perform identical functions:

- ▶ Any processor can run any thread on the system. This means that a process or thread ready to run can be dispatched to any processor, except the processes or threads bound to a specific processor using the **bindprocessor** command.
- ▶ Any processor can handle an external interrupt except interrupt levels bound to a specific processor using the **bindintcpu** command. Some SMP systems use a first fit interrupt handling in which an interrupt always gets directed to CPU0. If there are multiple interrupts at a time, the second interrupt is directed to CPU1, the third interrupt to CPU2, and so on. A process bound to CPU0 using the **bindprocessor** command may not get the necessary CPU time to run with best performance in this case.
- ▶ All processors can initiate I/O operations to any I/O device.

Cache coherency

All processors work with the same virtual and real address space and share the same real memory. However, each processor may have its own cache, holding a small subset of system memory. To guarantee cache coherency the processors use a snooping logic. Each time a word in the cache of a processor is changed, this processor sends a broadcast message over the bus. The processors are “snooping” on the bus, and if they receive a broadcast message about a modified word in the cache of another processor, they need to verify if they hold this changed address in their cache. If they do, they invalidate this entry in their

cache. The broadcast messages increase the load on the bus, and invalidated cache entries increase the number of cache misses. Both reduce the theoretical overall system performance, but hardware systems are designed to minimize the impact of the cache coherency mechanism.

Processor affinity

If a thread is running on a CPU and gets interrupted and redispached, the thread is placed back on the same CPU (if possible) because the processor's cache may still have lines that belong to the thread. If it is dispatched to a different CPU, the thread may have to get its information from main memory. Alternatively, it can wait until the CPU where it was previously running is available, which may result in a long delay.

AIX automatically tries to encourage processor affinity by having one run queue per CPU. Processor affinity can also be forced by binding a thread to a processor with the **bindprocessor** command. A thread that is bound to a processor can run only on that processor, regardless of the status of the other processors in the system. Binding a process to a CPU must be done with care, as you may reduce performance for that process if the CPU to which it is bound is busy and there are other idle CPUs in the system.

Locking

Access to I/O devices and real memory is serialized by hardware. Besides the physical system resources, such as I/O devices and real memory, there are logical system resources, such as shared kernel data, that are used by all processes and threads. As these processes and threads are able to run on any processor, a method to serialize access to these logical system resources is needed. The same applies for parallelized user code.

The primary method to implement resource access serialization is the usage of locks. A process or thread has to obtain a lock prior to accessing the shared resource. The process or thread has to release this lock after the access is completed. Lock and unlock functions are used to obtain and release these locks. The lock and unlock operations are atomic operations, and are implemented so that neither interrupts nor threads running on other processors affect the outcome of the operation. If a requested lock is already held by another thread, the requesting thread has to wait until the lock becomes available.

There are two ways for a thread to wait for a lock:

- ▶ Spin locks

A spin lock is suitable for a lock held only for a very short time. The thread waiting on the lock enters a tight loop wherein it repeatedly checks for the availability of the requested lock. No useful work is done by the thread at this time, and the processor time used is counted as time spent in system (kernel)

mode. To prevent a thread from spinning forever, it may be converted into a sleeping lock. An upper limit for the number of times to loop can be set using:

- The **schedo -o maxspin** command
The **maxspin** parameter is the number of times to spin on a kernel lock before sleeping. The default value of the **n** parameter for multiprocessor systems is 16384, and 1 (one) for uniprocessor systems. Refer to Chapter 10, “The **schedo** and **schedtune** commands” on page 165 for more details about the **schedo** command.
- The **SPINLOOPTIME** environment variable
The value of **SPINLOOPTIME** is the number of times to spin on a user lock before sleeping. This environment variable applies to the locking provided by **libpthreads.a**.
- The **YIELDLOOPTIME** environment variable
Controls the number of times to yield the processor before blocking on a busy user lock. The processor is yielded to another kernel thread, assuming there is another runnable kernel thread with sufficient priority. This environment variable applies to the locking provided by **libpthreads.a**.

► **Sleeping locks**

A sleeping lock is suitable for a lock held for a longer time. A thread requesting such a lock is put to sleep if the lock is not available. The thread is put back to the run queue if the lock becomes available. There is an additional overhead for context switching and dispatching for sleeping locks.

AIX provides two *types* of locks, which are:

► **Read-write lock**

Multiple readers of the data are allowed, but write access is mutually exclusive. The read-write lock has three states:

- Exclusive write
- Shared read
- Unlocked

► **Mutual exclusion lock**

Only one thread can access the data at a time. Other threads, even if they want only to read the data, have to wait. The mutual exclusion (mutex) lock has two states:

- Locked
- Unlocked

Both types of locks can be spin locks or sleeping locks.

Programmers in a multiprocessor environment should decide on the number of locks for shared data. If there is a single lock then lock contention (threads waiting on a lock) can occur often. If this is the case, more locks will be required. However, this can be more expensive because CPU time must be spent locking and unlocking, and there is a higher risk for a deadlock.

As locks are necessary to serialize access to certain data items, the heavy usage of the same data item by many threads may cause severe performance problems. In 19.5.2, “Examples for tprof” on page 329, we show an example of such a problem caused by a user-level application.

Refer to the *AIX 5L Version 5.2 Performance Management Guide* for further information about multiprocessing.

1.3 Memory performance

In a multi-user, multi-processor environment, the careful control of system resources is paramount. System memory, whether paging space or real memory, when not carefully managed can result in poor performance and even program and application failure. The AIX operating system uses the Virtual Memory Manager (VMM) to control real memory and paging space on the system.

1.3.1 Initial advice

We recommend that you do not make any VMM changes until you have had experience with the actual workload. Note that many parameters of the VMM can be monitored and tuned with the `vmo` command, described in Chapter 14, “The `vmo`, `ioo`, and `vmtune` commands” on page 229.

The discussion in this section is about:

- ▶ 1.3.2, “Memory segments” on page 13
- ▶ 1.3.3, “Paging mechanism” on page 14
- ▶ 1.3.4, “Memory load control mechanism” on page 15
- ▶ 1.3.5, “Paging space allocation policies” on page 15
- ▶ 1.3.6, “Memory leaks” on page 17
- ▶ 1.3.7, “Shared memory” on page 17

To learn more about how the VMM works, refer to:

- ▶ *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*
- ▶ *AIX 5L Version 5.2 Performance Management Guide*

1.3.2 Memory segments

A segment is 256 MB of contiguous virtual memory address space into which an object can be mapped. Virtual memory segments are partitioned into fixed sizes known as pages. Each page is 4096 bytes (4 KB) in size. A page in a segment can be in real memory or on disk where it is stored until it is needed. Real memory is divided into 4-KB page frames.

Simply put, the function of the VMM is to manage the allocation of real memory page frames and to resolve references from a program to virtual memory pages. Typically, this happens when pages are not currently in memory or do not exist when a process makes the first reference to a page of its data segment.

The amount of virtual memory used can exceed the size of the real memory of a system. The function of the VMM from a performance point of view is to:

- ▶ Minimize the processor use and disk bandwidth resulting from paging
- ▶ Minimize the response degradation from paging for a process

Virtual memory segments can be of three types:

- ▶ Persistent segments

Persistent segments are used to hold file data from the local filesystems. Because pages of a persistent segment have a permanent disk storage location, the VMM writes the page back to that location when the page has been changed if it can no longer be kept in memory. When a persistent page is opened for deferred update, changes to the file are not reflected on permanent storage until an fsync subroutine operation is performed. If no fsync subroutine operation is performed, the changes are discarded when the file is closed. No I/O occurs when a page of a persistent segment is selected for placement on the free list if that page has not been modified. If the page is referenced again later, it is read back in.

- ▶ Working segments

These segments are transitory and only exist during use by a process. Working segments have no permanent storage location, so they are stored in paging space when real memory pages must be freed.

- ▶ Client segments

These segments are saved and restored over the network to their permanent locations on a remote file system rather than being paged out to the local system. CD-ROM page-ins and compressed pages are classified as client segments. JFS2 pages are also mapped into client segments.

1.3.3 Paging mechanism

The VMM maintains a list of free memory page frames available to satisfy a *page fault*. This list is known as the *free list*. The VMM uses a *page replacement* algorithm. This algorithm is used to determine which pages in virtual memory will have their page frames reassigned to the free list.

When the number of pages in the free list becomes low, the *page stealer* is invoked. The page stealer is a mechanism that evaluates the Page Frame Table (PFT) entries looking for pages to steal. The PFT contains flags that indicate which pages have been referenced and which have been modified.

If the page stealer finds a page in the PFT that has been referenced, then it will not steal the page, but rather will reset the reference flag. The next time that the page stealer passes this page in the PFT, if it has not been referenced, it will be stolen. Pages that are not referenced when the page stealer passes them the first time are stolen.

When the modify flag is set on a page that has not been referenced, it indicates to the page stealer that the page has been modified since it was placed in memory. In this instance, a page out is called before the page is stolen. Pages that are part of a working segment are written to paging space, while pages of persistent segments are written to their permanent locations on disk.

There are two types of page fault:

- ▶ a new page fault, where the page is referenced for the first time
- ▶ a repage fault, where pages have already been paged out before

The stealer keeps track of the pages paged out, by using a history buffer that contains the IDs of the most recently paged-out pages. The history buffer also serves the purpose of maintaining a balance between pages of persistent segments and pages of working segments that get paged out to disk. The size of the history buffer is dependent on the amount of memory in the system; a memory size of 512 MB requires a 128 KB history buffer.

When a process terminates, its working storage is released and pages of memory are freed up and put back on the free list. Files that have been opened by the process can, however, remain in memory.

On an SMP system, the **1rud** kernel process is responsible for page replacement. This process is dispatched to a CPU when the minfree parameter threshold is reached. The minfree and maxfree parameters are set using the **vmo** command; see “The page replacement algorithm” on page 232 for more details. In the uniprocessor environment, page replacement is handled directly within the scope of the thread running.

The page replacement algorithm is most effective when the number of repages is low. The perfect replacement algorithm would eliminate repage faults completely and would steal pages that are not going to be referenced again.

1.3.4 Memory load control mechanism

If the number of active virtual memory pages exceeds the amount of real memory pages, paging space is used for those pages that cannot be kept in real memory. If an application accesses a page that was paged out, the VMM loads this page from the paging space into real memory. If the number of free real memory pages is low at this time, the VMM also needs to free another page in real memory before loading the accessed page from paging space. If the VMM only finds computational pages to free, it is forced to page out those pages to paging space. In the worst case the VMM always needs to page out a page to paging space before loading another page from paging space into memory. This condition is called *thrashing*. In a thrashing condition, processes encounter a page fault almost as soon as they are dispatched. None of the processes make any significant progress and the performance of the system deteriorates.

The operating system has a memory load control mechanism that detects when the thrashing condition is about to start. Once thrashing is detected, the system starts to suspend active processes and delay the start of any new processes. The memory load control mechanism is disabled by default on systems with more than 128 MB of memory. For more information about the load control mechanism and the `schedo` command, refer to 10.2.3, “Tuning memory parameters” on page 171.

1.3.5 Paging space allocation policies

The operating system supports three paging space allocation policies:

- ▶ Late Paging Space Allocation (LPSA)

With the LPSA, a paging slot is only allocated to a page of virtual memory when that page is first touched. The risk involved with this policy is that when the process touches the file, there may not be sufficient pages left in paging space.

- ▶ Early Paging Space Allocation (EPSA)

This policy allocates the appropriate number of pages of paging space at the time that the virtual memory address range is allocated. This policy ensures that processes do not get killed when the paging space of the system gets low. To enable EPSA, set the environment variable `PSALLOC=early`. Setting this policy ensures that when the process needs to page out, pages will be available. The recommended paging space size when adopting the EPSA policy is at least four times the size of real memory.

► Deferred Paging Space Allocation (DPSA)

This is the default policy in AIX 5L Version 5.2. The allocation of paging space is delayed until it is necessary to page out, so no paging space is wasted with this policy. Only once a page of memory is required to be paged out will the paging space be allocated. This paging space is reserved for that page until the process releases it or the process terminates. This method saves huge amounts of paging space. To disable this policy, the **vmo** command's **defps** parameter can be set to 0 (zero) with **vmo -o defps=0**. If the value is set to zero then the late paging space allocation policy is used.

Tuning paging space thresholds

When paging space becomes depleted, the operating system attempts to release resources by first warning processes to release paging space, and then by killing the processes. The **vmo** command is used to set the thresholds at which this activity will occur. The **vmo** tunables that affect paging are:

- npswarn** The operating system sends the SIGDANGER signal to all active processes when the amount of paging space left on the system goes below this threshold. A process can either ignore the signal or it can release memory pages using the `disclaim()` subroutine.
- npskill** The operating system will begin killing processes when the amount of paging space left on the system goes below this threshold. When the **npskill** threshold is reached, the operating system sends a SIGKILL signal to the youngest process. Processes that are handling a SIGDANGER signal and processes that are using the EPSA policy are exempt from being killed.
- nokilluid** By setting the value of the **nokilluid** value to 1 (one), the root processes will be exempt from being killed when the **npskill** threshold is reached. User identifications (UIDs) lower than the number specified by this parameter are not killed when the **npskill** parameter threshold is reached.

For more information about the setting these parameters, refer to Chapter 14, "The **vmo**, **ioo**, and **vmtune** commands" on page 229.

When a process cannot be forked due to a lack of paging space, the scheduler will make five attempts to fork the process before giving up and putting the process to sleep. The scheduler delays 10 clock ticks between each retry. By default, each clock tick is 10 ms. This results in 100 ms between retries. The **schedo** command has a **pacefork** value that can be used to change the number of times the scheduler will retry a fork.

To monitor the amount of paging space, use the `lsp` command. The `-s` flag should be issued rather than the `-a` flag of the `lsp` command because the former includes pages in paging space reserved by the EPSA policy.

1.3.6 Memory leaks

Systems have been known to run out of paging space because of memory leaks in long-running programs that are interactive applications. A memory leak is a program error where the program repeatedly allocates memory, uses it, and then neglects to free it. The `svmon` command is useful in detecting memory leaks. Use the `svmon` command with the `-i` flag to look for processes or groups of processes whose working segments are continually growing. For more information about the `svmon` command, refer to Chapter 24, “The `svmon` command” on page 387.

1.3.7 Shared memory

Memory segments can be shared between processes. Using shared memory avoids buffering and system call overhead. Applications reduce the overhead of read and write system calls by manipulating pointers in these memory segments. Both files and data in shared segments can be shared by multiple processes and threads, but the synchronization between processes or threads must be done at the application level.

By default, each shared memory segment or region has an address space of 256 MB, and the maximum number of shared memory segments that the process can access at the same time is limited to 11. Using extended shared memory increases this number to more than 11 segments and allows shared memory regions to be any size from 1 byte up to 256 MB. Extended shared memory is available to processes that have the variable `EXTSHM` set to `ON` (that is, `EXTSHM=ON` in their process environment). The restrictions of extended shared memory are:

- ▶ I/O is restricted in the same way as for memory regions.
- ▶ Raw I/O is not supported.
- ▶ They cannot be used as I/O buffers where the unpinning of buffers occurs in an interrupt handler.
- ▶ They cannot be pinned using the `plock()` subroutine.

1.4 Disk I/O performance

A lot of attention is required when the disk subsystem is designed and implemented. For example, you will need to consider the following:

- ▶ Bandwidth of disk adapters and system bus
- ▶ Placement of logical volumes on the disks
- ▶ Configuration of disk layouts
- ▶ Operating system settings, such as striping or mirroring
- ▶ Performance implementation of other technologies, such as SSA

1.4.1 Initial advice

We recommend that you do not make any changes to the default disk I/O parameters until you have had experience with the actual workload. Note, however, that you should always monitor the I/O workload and will very probably need to balance the physical and logical volume layout after runtime experience.

There are two performance-limiting aspects of the disk I/O subsystem that must be considered:

- ▶ Physical limitations
- ▶ Logical limitations

A poorly performing disk I/O subsystem usually will severely penalize overall system performance.

Physical limitations concern the throughput of the interconnecting hardware. Logical limitations concern limiting both the physical bandwidth and the resource serialization and locking mechanisms built into the data access software¹. Note that many logical limitations on the disk I/O subsystem can be monitored and tuned with the `ioo` command. See Chapter 14, “The `vmo`, `ioo`, and `vmtune` commands” on page 229 for details.

For further information, refer to:

- ▶ *AIX 5L Version 5.2 Performance Management Guide*
- ▶ *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*
- ▶ *AIX 5L Version 5.2 System Management Guide: Operating System and Devices*
- ▶ *RS/6000 SP System Performance Tuning Update, SG24-5340*

¹ Usually to ensure data integrity and consistency (such as file system access and mirror consistency updating).

1.4.2 Disk subsystem design approach

For many systems, the overall performance of an application is bound by the speed at which data can be accessed from disk and the way the application reads and writes data to the disks. Designing and configuring a disk storage subsystem for performance is a complex task that must be carefully thought out during the initial design stages of the implementation. Some of the factors that must be considered include:

- ▶ Performance versus availability

A decision must be made early on as to which is more important; I/O performance of the application or application integrity and availability. Increased data availability often comes at the cost of decreased system performance and vice versa. Increased availability also may result in larger amounts of disk space being required.

- ▶ Application workload type

The I/O workload characteristics of the application should be fairly well understood prior to implementing the disk subsystem. Different workload types most often require a different disk subsystem configuration in order to provide acceptable I/O performance.

- ▶ Required disk subsystem throughput

The I/O performance requirements of the application should be defined up front, as they will play a large part in dictating both the physical and logical configuration of the disk subsystem.

- ▶ Required disk space

Prior to designing the disk subsystem, the disk space requirements of the application should be well understood.

- ▶ Cost

While not a performance-related concern, overall cost of the disk subsystem most often plays a large part in dictating the design of the system. Generally, a higher-performance system costs more than a lower-performance one.

1.4.3 Bandwidth-related performance considerations

The bandwidth of a communication link, such as a disk adapter or bus, determines the maximum speed at which data can be transmitted over the link. When describing the capabilities of a particular disk subsystem component, performance numbers typically are expressed in maximum or peak throughput, which often do not realistically describe the true performance that will be realized in a real world setting. In addition, each component most will likely have different bandwidths, which can create bottlenecks in the overall design of the system.

The bandwidth of each of the following components must be taken into consideration when designing the disk subsystem:

▶ Disk devices

The latest SCSI and SSA disk drives have maximum sustained data transfer rates of 14-20 MB per second. Again, the real world expected rate will most likely be lower depending on the data location and the I/O workload characteristics of the application. Applications that perform a large amount of sequential disk reads or writes will be able to achieve higher data transfer rates than those that perform primarily random I/O operations.

▶ Disk adapters

The disk adapter can become a bottleneck depending on the number of disk devices that are attached and their use. While the SCSI-2 specification allows for a maximum data transfer rate of 20 MBps, adapters based on the UltraSCSI specification are capable of providing bandwidth of up to 40 MBps. The SCSI bus used for data transfer is an arbitrated bus. In other words, only one initiator or device can be sending data at any one time. This means the theoretical maximum transfer rate is unlikely to be sustained. By comparison, the IBM SSA adapters use a non-arbitrated loop protocol, which also supports multiple concurrent peer-to-peer data transfers on the loop. The current SSA adapters are capable of supporting maximum theoretical data transfer rates of 160 MBps.

▶ System bus

The system bus architecture used can further limit the overall bandwidth of the disk subsystem. Just as the bandwidth of the disk devices is limited by the bandwidth of the disk adapter to which they are attached, the speed of the disk adapter is limited by the bandwidth of the system bus. The industry standard PCI bus is limited to a theoretical maximum of either 132 MBps (32-bit @ 33MHz) or 528 MBps (64-bit @ 66MHz).

1.4.4 Disk design

A disk consists of a set of flat, circular rotating platters. Each platter has one or two sides on which data is stored. Platters are read by a set of non-rotating, but positionable, read or read/write heads that move together as a unit. The following terms are used when discussing disk device block operations:

Sector

An addressable subdivision of a track used to record one block of a program or data. On a disk, this is a contiguous, fixed-size block. Every sector of every disk is exactly 512 bytes.

Track

A circular path on the surface of a disk on which information is recorded and from which recorded information is read; a

contiguous set of sectors. A track corresponds to the surface area of a single platter swept out by a single head while the head remains stationary.

Head A positionable entity that can read and write data from a given track located on one side of a platter. Usually a disk has a small set of heads that move from track to track as a unit.

Cylinder The tracks of a disk that can be accessed without repositioning the heads. If a disk has n number of vertically aligned heads, a cylinder has n number of vertically aligned tracks.

Disk access times

The three components that make up the access time of a disk are:

Seek A seek is the physical movement of the head at the end of the disk arm from one track to another. The time for a seek is the time needed for the disk arm to accelerate, to travel over the tracks to be skipped, to decelerate, and finally to settle down and wait for the vibrations to stop while hovering over the target track. The total time the seeks take is variable. The average seek time is used to measure the disk capabilities.

Rotational This is the time that the disk arm has to wait while the disk is rotating underneath until the target sector approaches. Rotational latency is, for all practical purposes except sequential reading, a random function with values uniformly between zero and the time required for a full revolution of the disk. The average rotational latency is taken as the time of a half revolution. To determine the average latency, you must know the number of revolutions per minute (RPM) of the drive. By converting the RPMs to revolutions per second and dividing by 2, we get the average rotational latency.

Transfer The data transfer time is determined by the time it takes for the requested data block to move through the read/write arm. It is linear with respect to the block size. The average disk access time is the sum of the averages for seek time and rotational latency plus the data transfer time (normally given for a 512-byte block). The average disk access time generally overestimates the time necessary to access a disk; typical disk access time is 70 percent of the average.

Disks per adapter bus or loop

Discussions of disk, logical volume, and file system performance sometimes lead to the conclusion that the more drives you have on your system, the better the disk I/O performance. This is not always true because there is a limit to the amount of data that can be handled by a disk adapter, which can become a bottleneck. If all your disk drives are on one disk adapter and your hot file systems are on separate physical volumes, you might benefit from using multiple disk adapters. Performance improvement will depend on the type of access.

Using the proper number of disks per adapter is essential. For both SCSI and SSA adapters the maximum number of disks per bus or loop should not exceed four if maximum throughput is needed and can be utilized by the applications. For SCSI the limiting factor is the bus, and for SSA it is the adapter.

The major performance issue for disk drives is usually application-related; that is, whether large numbers of small accesses (random) or smaller numbers of large accesses (sequential) will be made. For random access, performance generally will be better using larger numbers of smaller-capacity drives. The opposite situation exists for sequential access (use faster drives or use striping with a larger number of drives).

Physical disk buffers

The Logical Volume Manager (LVM) uses a construct called a *pbuf* (physical buffer) to control a pending disk I/O. A single pbuf is used for each I/O request, regardless of the number of pages involved. AIX creates extra pbufs when a new physical volume is added to the system. When striping is used, you need more pbufs because one I/O operation causes I/O operations to more disks and, therefore, more pbufs. When striping and mirroring is used, even more pbufs are required. Running out of pbufs reduces performance considerably because the I/O process is suspended until pbufs are available again. Increase the number of pbufs with the `ioo` command (see “I/O tuning parameters” on page 245); however, pbufs are pinned so that allocating many pbufs increases the use of memory.

A special note should be given to adjusting the number of pbufs on systems with many disks attached or available with the `ioo` command. The number of pbufs per active disk should be twice the queue depth of the disk or 32, whatever is greater. The default maximum number of pbufs should not exceed a total of 65536.

The script in Example 1-1 on page 23 extracts the information for each disk and calculates a recommendation for `ioo -o hd_pbuf_cnt`. The script does not take into account multiple Serial Storage Architecture (SSA) pdisks or hdisks using `vpath`. It uses the algorithm shown in Example 1-2 on page 23.

Note: The script in Example 1-1 cannot be used for disks with multiple connections.

Example 1-1 ioo_calc_puf.sh

```
1 #!/bin/ksh
2 integer max_pbuf_count=65535
3 integer hd_pbuf_cnt=128
4 integer current_hd_pbuf_cnt=$(ioo -o hd_pbuf_cnt|awk '{print $3;exit}')
5 lsdev -Cc disk -Fname|
6 while read disk;do
7     integer queue_depth=$(lsattr -El $disk -aqueue_depth -Fvalue)
8     ((pbuf_to_add=queue_depth*2))
9     if (( pbuf_to_add < 32));then
10         pbuf_to_add=32
11     fi
12     if (( (hd_pbuf_cnt+pbuf_to_add) > max_pbuf_count));then
13         ((pbuf_to_add=max_pbuf_count-hd_pbuf_cnt))
14     fi
15     ((hd_pbuf_cnt+=pbuf_to_add))
16 done
17 if (( current_hd_pbuf_cnt < hd_pbuf_cnt ));then
18     print "Run ioo -o hd_pbuf_cnt $hd_pbuf_cnt to change from
19 $current_hd_pbuf_cnt to $hd_pbuf_cnt"
20 else
21     print "The current hd_pbuf_cnt ($current_hd_pbuf_cnt) is OK"
22 fi
```

The algorithm in Example 1-2 is used for setting pbufs.

Example 1-2 Algorithm used for setting pbufs

```
max_pbuf_count = 65535
hd_pbuf_cnt 128
for each disk {
    pbuf_to_add = queue_depth * 2
    if ( pbuf_to_add < 32)
        pbuf_to_add = 32
    if ( (hd_pbuf_cnt + pbuf_to_add) > max_pbuf_count)
        pbuf_to_add = max_pbuf_count - hd_pbuf_cnt
    hd_pbuf_cnt += pbuf_to_add
}
```

Note that more buffers might have to be increased on a large server system. On large server systems, you should always monitor the utilization with the **ioo** command and adjust the parameter values appropriately. File system buffers for LVM require that the change is made before the filesystem is mounted. See “I/O

tuning parameters” on page 245 for more details about monitoring and changing these values and parameters.

1.4.5 Logical Volume Manager concepts

Many modern UNIX operating systems implement the concept of a Logical Volume Manager (LVM) that can be used to logically manage the distribution of data on physical disk devices. The AIX LVM is a set of operating system commands, library subroutines, and other tools used to control physical disk resources by providing a simplified logical view of the available storage space. Unlike other LVM offerings, the AIX LVM is an integral part of the base AIX operating system provided at no additional cost.

Within the LVM, each disk or physical volume (PV) belongs to a volume group (VG). A volume group is a collection of 1 to 32 physical volumes (1 to 128 in the case of a big volume group), which can vary in capacity and performance. A physical volume can belong to only one volume group at a time. A maximum of 255 volume groups can be defined per system.

When a volume group is created, the physical volumes within the volume group are partitioned into contiguous, equal-sized units of disk space known as physical partitions. Physical partitions are the smallest unit of allocatable storage space in a volume group. The physical partition size is determined at volume group creation, and all physical volumes that are placed in the volume group inherit this size. The physical partition size can range from 1 MB to 1024 MB, but must be a power of two. If not specified, the default physical partition size in AIX is 4 MB for disks up to 4 GB, but it must be larger for disks greater than 4 GB due to the fact that the LVM, by default, will only track up to 1016 physical partitions per disk (unless you use the `-t` option with `mkvg`, which reduces the maximum number of physical volumes in the volume group). In AIX 5L Version 5.2, the minimum PP size needed is determined by the operating system if the default size of 4 MB is specified.

Use of LVM policies

Deciding on the physical layout of an application is one of the most important decisions to be made when designing a system for optimal performance. The physical location of the data files is critical to ensuring that no single disk, or group of disks, becomes a bottleneck in the I/O performance of the application. In order to minimize their impact on disk performance, heavily accessed files should be placed on separate disks, ideally under different disk adapters. There are several ways to ensure even data distribution among disks and adapters, including operating system level data striping, hardware data striping on a Redundant Array of Independent Disks (RAID), and manually distributing the application data files among the available disks.

The disk layout on a server system is usually very important to determine the possible performance that can be achieved from disk I/O.

The AIX LVM provides a number of facilities or *policies* for managing both the performance and availability characteristics of logical volumes. The policies that have the greatest impact on performance are intra-disk allocation, inter-disk allocation, I/O scheduling, and write-verify policies.

Intra-disk allocation policy

The intra-disk allocation policy determines the actual physical location of the physical partitions on disk. A disk is logically divided into the following five concentric areas as shown in Figure 1-1:

- ▶ Outer edge
- ▶ Outer middle
- ▶ Center
- ▶ Inner middle
- ▶ Inner edge

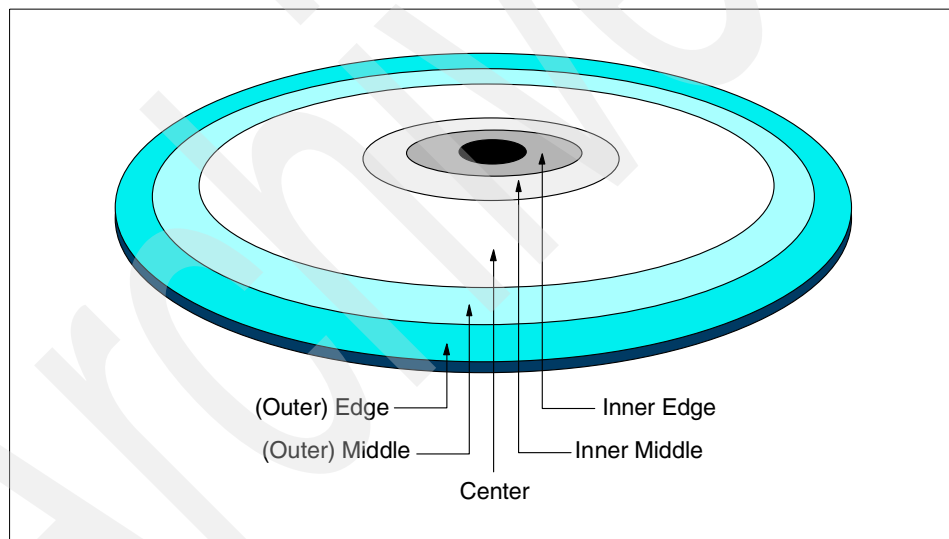


Figure 1-1 Physical partition mapping

Due to the physical movement of the disk actuator, the outer and inner edges typically have the largest average seek times and are a poor choice for application data that is frequently accessed. The center region provides the fastest average seek times and is the best choice for paging space or applications that generate a significant amount of random I/O activity. The outer and inner middle regions provide better average seek times than the outer and inner edges, but worse seek times than the center region.

As a general rule, when designing a logical volume strategy for performance, the most performance-critical data should be placed as close to the center of the disk as possible. There are, however, two notable exceptions:

- ▶ Applications that perform a large amount of sequential reads or writes experience higher throughput when the data is located on the outer edge of the disk due to the fact that there are more data blocks per track on the outer edge of the disk than the other disk regions.
- ▶ Logical volumes with Mirrored Write Consistency (MWC) enabled should also be located at the outer edge of the disk, as this is where the MWC cache record is located.

When the disks are set up in a RAID5 configuration, the intra-disk allocation policy will not have any benefits to performance.

Inter-disk allocation policy

The inter-disk allocation policy is used to specify the number of disks that contain the physical partitions of a logical volume. The physical partitions for a given logical volume can reside on one or more disks in the same volume group depending on the setting of the *range* option. The range option can be set by using the **smitty mklv** command and changing the *RANGE of physical volumes* menu option.

- ▶ The maximum range setting attempts to spread the physical partitions of a logical volume across as many physical volumes as possible in order to decrease the average access time for the logical volume.
- ▶ The minimum range setting attempts to place all of the physical partitions of a logical volume on the same physical disk. If this cannot be done, it will attempt to place the physical partitions on as few disks as possible. The minimum setting is used for increased availability only, and should not be used for frequently accessed logical volumes. If a non-mirrored logical volume is spread across more than one drive, the loss of any of the physical drives will result in data loss. In other words, a non-mirrored logical volume spread across two drives will be twice as likely to experience a loss of data as one that resides on only one drive.

The physical partitions of a given logical volume can be mirrored to increase data availability. The location of the physical partition copies is determined by setting the *Strict* option with the **smitty mklv** command called *Allocate each logical partition copy*. When *Strict = y*, each physical partition copy is placed on a different physical volume. When *Strict = n*, the copies can be on the same physical volume or different volumes. When using striped and mirrored logical volumes in AIX 4.3.3 and above, there is an additional partition allocation policy known as *superstrict*. When *Strict = s*, partitions of one mirror cannot share the

same disk as partitions from a second or third mirror, further reducing the possibility of data loss due to a single disk failure.

In order to determine the data placement strategy for a mirrored logical volume, the settings for both the range and Strict options must be carefully considered. As an example, consider a mirrored logical volume with range setting of minimum and a strict setting of yes. The LVM would attempt to place all of the physical partitions associated with the primary copy on one physical disk, with the mirrors residing on either one or two additional disks, depending on the number of copies of the logical volume (2 or 3). If the strict setting were changed to no, all of the physical partitions corresponding to both the primary and mirrors would be located on the same physical disk.

I/O-scheduling policy

The default for logical volume mirroring is that the copies should use different disks. This is both for performance and data availability. With copies residing on different disks, if one disk is extremely busy, then a read request can be completed using the other copy residing on a less busy disk. Different I/O scheduling policies can be set for logical volumes. The different I/O scheduling policies are as follows:

- | | |
|-----------------------------|--|
| Sequential | The sequential policy results in all reads being issued to the primary copy. Writes happen serially, first to the primary disk; only when that is completed is the second write initiated to the secondary disk. |
| Parallel | The parallel policy balances reads between the disks. On each read, the system checks whether the primary is busy. If it is not busy, the read is initiated on the primary. If the primary is busy, the system checks the secondary. If it is not busy, the read is initiated on the secondary. If the secondary is busy, the read is initiated on the copy with the fewest number of outstanding I/Os. Writes are initiated concurrently. |
| Parallel/sequential | The parallel/sequential policy always initiates reads on the primary copy. Writes are initiated concurrently. |
| Parallel/round-robin | The parallel/round-robin policy is similar to the parallel policy except that instead of always checking the primary copy first, it alternates between the copies. This results in equal utilization for reads even when there is never more than one I/O outstanding at a time. Writes are initiated concurrently. |

Write-verify policy

When the write-verify policy is enabled, all write operations are validated by immediately performing a follow-up read operation of the previously written data. An error message will be returned if the read operation is not successful. The use of write-verify enhances the integrity of the data but can drastically degrade the performance of disk writes.

Mirror write consistency (MWC)

The Logical Volume Device Driver (LVDD) always ensures data consistency among mirrored copies of a logical volume during normal I/O processing. For every write to a logical volume, the LVDD² generates a write request for every mirror copy. If a logical volume is using mirror write consistency, then requests for this logical volume are held within the scheduling layer until the MWC cache blocks can be updated on the target physical volumes. When the MWC cache blocks have been updated, the request proceeds with the physical data write operations. If the system crashes in the middle of processing, a mirrored write (before all copies are written) MWC will make logical partitions consistent after a reboot.

MWC Record The MWC Record consists of one disk sector. It identifies which logical partitions may be inconsistent if the system is not shut down correctly.

MWC Check The MWC Check (MWCC) is a method used by the LVDD to track the last 62 distinct Logical Track Groups (LTGs) written to disk. By default, an LTG is 32 4-KB pages (128 KB). AIX 5L supports LTG sizes of 128 KB, 256 KB, 512 KB, and 1024 KB. MWCC only makes mirrors consistent when the volume group is varied back online after a crash by examining the last 62 writes to mirrors, picking one mirror, and propagating that data to the other mirrors. MWCC does not keep track of the latest data; it only keeps track of LTGs currently being written. Therefore, MWC does not guarantee that the latest data will be propagated to all of the mirrors. It is the application above LVM that has to determine the validity of the data after a crash.

There are three different states for the MWC:

Disabled (off) MWC is not used for the mirrored logical volume. To maintain consistency after a system crash, the logical volumes file system must be manually mounted after reboot, but only after the **syncvg** command has been used to synchronize the physical partitions that belong to the mirrored logical partition.

² The scheduler layer (part of the bottom half of LVDD) schedules physical requests for logical operations and handles mirroring and the MWC cache.

- Active** MWC is used for the mirrored logical volume and the LVDD will keep the MWC record synchronized on disk. Because every update will require a repositioning of the disk write head to update the MWC record, it can cause a performance problem. When the volume group is varied back online after a system crash, this information is used to make the logical partitions consistent again.
- Passive** MWC is used for the mirrored logical volume but the LVDD will not keep the MWC record synchronized on disk. Synchronization of the physical partitions that belong to the mirrored logical partition will be updated after IPL. This synchronization is performed as a background task (**syncvg**). The passive state of MWC only applies to big volume groups. Big volume groups can accommodate up to 128 physical volumes and 512 logical volumes. To create a big volume group, use the **mkvg -B** command. To change a regular volume group to a big volume group, use the **chvg -B** command.

The type of mirror consistency checking is important for maintaining data accuracy even when using MWC. MWC ensures data consistency, but not necessarily data accuracy.

Log logical volume

The log logical volume should be placed on a different physical volume from the most active file system. Placing it on a disk with the lowest I/O utilization will increase parallel resource usage. A separate log can be used for each file system. However, special consideration should be taken if multiple logs must be placed on the same physical disk, which should be avoided if possible.

The general rule to determine the appropriate size for the JFS log logical volume is to have 4 MB of JFS log for each 2 GB of file system space. The JFS log is limited to a maximum size of 256 MB.

Note that when the size of the log logical volume is changed, the **logform** command must be run to reinitialize the log before the new space can be used.

nointegrity

The mount option **nointegrity** bypasses the use of a log logical volume for the file system mounted with this option. This can provide better performance as long as the administrator knows that the **fsck** command might have to be run on the file system if the system goes down without a clean shutdown.

```
mount -o nointegrity /filesystem
```

To make the change permanent, either add the option to the options field in `/etc/filesystems` manually or do it with the `chfs` command as follows (in this case for the file system):

```
chfs -a options=nointegrity,rw /filesystem
```

JFS2 inline log

In AIX 5L, log logical volumes can be either of JFS or JFS2 types, and are used for JFS and JFS2 file systems respectively. The JFS2 file system type allows the use of an inline journaling log. This log section is allocated within the JFS2 itself.

Paging space

If paging space is needed in a system, performance and throughput always suffer. The obvious conclusion is to eliminate paging to paging space as much as possible by having enough real memory available for applications when they need it. Paging spaces are accessed in a round-robin fashion, and the data stored in the logical volumes is of no use to the system after a reboot/IPL.

The current default paging-space-slot-allocation method, Deferred Page Space Allocation (DPSA), delays allocation of paging space until it is necessary to page out the page.

Some rules of thumb when it comes to allocating paging space logical volumes are:

- ▶ Use the disk or disks that are least utilized.
- ▶ Do not allocate more than one paging space logical volume per physical disk.
- ▶ Avoid sharing the same disk with log logical volumes.
- ▶ If possible, make all paging spaces the same size.

Because the data in a page logical volume cannot be reused after a reboot/IPL, the MWC is disabled for mirrored paging space logical volumes when the logical volume is created.

Recommendations for performance optimization

As with any other area of system design, when deciding on the LVM policies, a decision must be made as to which is more important; performance or availability. The following LVM policy guidelines should be followed when designing a disk subsystem for performance:

- ▶ When using LVM mirroring:
 - Use a parallel write-scheduling policy.
 - Allocate each logical partition copy on a separate physical disk by using the Strict option of the inter-disk allocation policy.
- ▶ Disable write-verify.

- ▶ Allocate heavily accessed logical volumes near the center of the disk.
- ▶ Use an intra-disk allocation policy of maximum in order to spread the physical partitions of the logical volume across as many physical disks as possible.

1.5 Network performance

Tuning network utilization is a complex and sometimes very difficult task. You need to know how applications communicate and how the network protocols work on AIX and other systems involved in the communication. The only general recommendation for network tuning is that Interface Specific Network Options (ISNO) should be used and buffer utilization should be monitored. Some basic network tunables for improving throughput can be found in Table 1-2 on page 36. Note that with network tuning, indiscriminately using buffers that are too large can reduce performance.

To learn more about how the different protocols work, refer to:

- ▶ Chapter 34, “The no command” on page 665
- ▶ Chapter 32, “The nfso command” on page 645
- ▶ *AIX 5L Version 5.2 Performance Management Guide*
- ▶ *AIX 5L Version 5.2 System Management Guide: Communications and Networks*
- ▶ *AIX 5L Version 5.2 System Management Guide: Operating System and Devices*
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376
- ▶ *RS/6000 SP System Performance Tuning Update*, SG24-5340
- ▶ <http://www.rs6000.ibm.com/support/sp/perf>
- ▶ Appropriate Request For Comment (RFC) at <http://www.rfc-editor.org/>

There are also excellent books available on the subject, and a good starting point is RFC 1180 A TCP/IP Tutorial. A short overview of the TCP/IP protocols can be found in 1.5.2, “TCP/IP protocols” on page 33. Information about the network tunables, including network adapter tunables, is provided in 1.5.3, “Network tunables” on page 34.

1.5.1 Initial advice

The knowledge of the network topology used is necessary to understand and detect possible performance bottlenecks on the network. This includes information about the routers and gateways used, the Maximum Transfer Unit

(MTU) used on the network path between the systems, and the current load on the networks used. This information should be well documented, and access to these documents needs to be guaranteed at any time.

AIX offers a wide range of tools to monitor networks, network adapters, network interfaces, and system resources used by the network software. These tools are covered in detail in Chapter 6, “Network-related performance tools” on page 531. Use these tools to gather information about your network environment when everything is functioning correctly. This information will be very useful in case a network performance problem arises, because a comparison between the monitored information of the poorly performing network and the earlier well-performing network helps to detect the problem source. The information gathered should include:

- ▶ Configuration information from the server and client systems

A change in the system configuration can be the cause of a performance problem. Sometimes such a change may be done by accident, and finding the changed configuration parameter to correct it can be very difficult. The **snap -a** command can be used to gather system configuration information. Refer to the *AIX 5L Version 5.2 Commands Reference*, SBOF-1877 for more information about the **snap** command.

- ▶ The system load on the server system

Poor performance on a client system is not necessarily a network problem. If case the server system is short on local resources, such as CPU or memory, it may be unable to answer the client’s request in the expected time. The **perfpmr** tool can be used to gather this information. Refer to Chapter 7, “The perfpmr command” on page 115 for more information.

- ▶ The system load on the client system

The same considerations for the server system apply to the client system. A shortage of local resources, such as CPU or memory, can slow down the client’s network operation. The **perfpmr** tool can be used to gather this information; refer to Chapter 7, “The perfpmr command” on page 115 for more information.

- ▶ The load on the network

The network usually is a resource shared by many systems. Poor performance between two systems connected to the network may be caused by an overloaded network, and this overload could be caused by other systems connected to the network. There are no native tools in AIX to gather

information about the load on the network itself. Tools such as Sniffer, DatagLANce Network Analyzer, and Nways® Workgroup Manager can provide such information. Detailed information about the network management products IBM offers can be found at:

<http://www.networking.ibm.com/netprod.html>

However, tools such as **ping** or **tracert** can be used to gather turnaround times for data on the network. The **ftp** command can be used to transfer a large amount of data between two systems using `/dev/zero` as input and `/dev/null` as output, and registering the throughput. This is done by opening an **ftp** connection, changing to binary mode, and then executing the **ftp** sub command that transfers 10000 * 32 KB over the network. :

```
put "| dd if=/dev/zero bs=32k count=10000" /dev/null
```

► Network interface throughput

The commands **atmstat**, **estat**, **entstat**, **fddistat**, and **tokstat** can be used to gather throughput data for a specific network interface. The first step would be to generate a load on the network interface. Use the example above, **ftp** using **dd** to do a put. Without the `count=10000` the **ftp put** command will run until it is interrupted.

While **ftp** is transferring data, issue the command sequence:

```
entstat -r en2;sleep 100;entstat en2>/tmp/entstat.en2
```

It is used to reset the statistics for the network interface, in our case `en2` (**entstat -r en2**), wait 100 seconds (**sleep 100**), and then gather the statistics for the interface (**entstat en2>/tmp/entstat.en2**). Refer to Chapter 29, “atmstat, entstat, estat, fddistat, and tokstat commands” on page 539 for details on these commands.

► Output of network monitoring commands on both the server and client

The output of the commands should be part of the data gathered by the **perfpnr** tool. However, the **perfpnr** tool may change, so it is advised to control the data gathered by **perfpnr** to ensure that the outputs of the **netstat** and **nfsstat** commands are included.

1.5.2 TCP/IP protocols

Application programs send data by using one of the Internet Transport Layer Protocols, either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). These protocols receive the data from the application, divide it into smaller pieces called packets, add a destination address, and then pass the packets along to the next protocol layer, the Internet Network layer.

The Internet Network layer encloses the packet in an Internet Protocol (IP) datagram, adds the datagram header and trailer, decides where to send the

datagram (either directly to a destination or else to a gateway), and passes the datagram on to the Network Interface layer.

The Network Interface layer accepts IP datagrams and transmits them as frames over a specific network hardware, such as Ethernet or token-ring networks.

For more detailed information about the TCP/IP protocol, refer *AIX 5L Version 5.2 System Management Guide: Communications and Networks*, and *TCP/IP Tutorial and Technical Overview*, GG24-3376.

To interpret the data created by programs such as the **iptrace** and **tcpdump** commands, formatted by **ipreport**, and summarized with **ipfilter**, you need to understand how the TCP/IP protocols work together. See Chapter 30, “TCP/IP packet tracing tools” on page 567. Table 1-1 is a short, top-down reminder of TCP/IP protocols hierarchy.

Table 1-1 TCP/IP layers and protocol examples

TCP/IP Layer	Protocol Examples
Application	Telnet, FTP, SMTP, LPD
Transport	TCP, UDP
Internet Network	IP, ICMP, IGMP, ARP, RARP
Network Interface	Ethernet, token-ring, ATM, FDDI, SP Switch
Hardware	Physical network

1.5.3 Network tunables

In most cases you need to adjust some network tunables on server systems. Most of these settings concern different network protocol buffers. You can set these buffer sizes systemwide with the **no** command (refer to Chapter 34, “The no command” on page 665), or use the Interface Specific Network Options³ (ISNO) for each network adapter. For more details about ISNO, see *AIX 5L Version 5.2 System Management Guide: Communications and Networks* and *AIX 5L Version 5.2 Commands Reference*, SBOF-1877.

The change will only apply to the specific network adapter if you have enabled ISNO with the **no** command as in the following example:

```
no -o use_isno=1
```

³ There are five ISNO parameters for each supported interface; `rfc1323`, `tcp_nodelay`, `tcp_sendspace`, `tcp_recvspace`, and `tcp_msdfilt`. When set, the values for these parameters override the systemwide parameters of the same names that had been set with the **no** command. When ISNO options are not set for a particular interface, systemwide options are used. Options set by an application for a particular socket using the *setsockopt* subroutine override the ISNO options and systemwide options set by using the **chdev**, **ifconfig**, and **no** commands.

If different network adapter types with a big difference of MTU sizes are used in the system, using ISNO to tune each network adapter for best performance is the preferred way. For example with Ethernet adapters using an MTU of 1500 and an ATM adapter using an MTU of 65527 installed.

Document the current values before making any changes, especially if you use ISNO to change the individual interfaces. Example 1-3 shows how to use the **lsattr** command to check the current settings for an network interface, in this case token-ring:

Example 1-3 Using lsattr to check adapter settings

```
# lsattr -H -E1 tr0 -F"attribute value"
attribute      value

mtu            1492
mtu_4          1492
mtu_16         1492
mtu_100        1492
remmtu         576
netaddr        10.3.2.164
state          up
arp            on
allcast        on
hwloop         off
netmask        255.255.255.0
security       none
authority
broadcast
netaddr6
alias6
prefixlen
alias4
rfc1323        0
tcp_nodelay
tcp_sendspace 16384
tcp_recvspace 16384
tcp_mssdf1t
```

The highlighted part in the Example 1-3 output indicates the ISNO options. Before applying ISNO settings to interfaces by using the **chdev** command, you can use **ifconfig** to set them on each adapter. Should you for some reason need to reset them and are unable to log in to the system, the values will not be permanent and will not be activated after IPL. For this reason it is not recommended to set ISNO values using **ifconfig** in any system startup scripts that are started by init.

Network buffer tuning

The values in Table 1-2 are settings that have proved to give the highest network throughput for each network type. A general rule is to set the TCP buffer sizes to 10 times the MTU size, but as can be seen in the following table, this is not always true for all network types.

Table 1-2 Network tunables minimum values for best performance

Device	Speed Mbit	MTU	tcp sendspace	tcp ^a recvspace	sb_max	rfc 1323
Ethernet	10	1500	16384	16384	32768	0
Ethernet	100	1500	16384	16384	32768	0
Ethernet	1000	1500	131072	65536	131072	0
Ethernet	1000	9000	131072	65536	262144	0
Ethernet	1000	9000	262144	131072	262144	1
ATM	155	1500	16384	16384	131072	0
ATM	155	9180	65536	65536	131072	1
ATM	155	65527	655360	655360	1310720	1
FDDI	100	4352	45056	45056	90012	0
SPSW	-	65520	262144	262144	1310720	1
SPSW2	-	65520	262144	262144	1310720	1
HiPPI	-	65536	655360	655360	1310720	1
HiPS	-	65520	655360	655360	1310720	1
ESCON®	-	4096	40960	40960	81920	0
Token-ring	4	1492	16384	16384	32768	0
Token-ring	16	1492	16384	16384	32768	0
Token-ring	16	4096	40960	40960	81920	0
Token-ring	16	8500	65536	65536	131072	0

a. If an application sends only a small amount of data and then waits for a response, the performance may degrade if the buffers are too large, especially when using large MTU sizes. It might be necessary to either tune the sizes further or disable the Nagle algorithm by setting `tcp_nagle_limit` to 0 (zero).

Other network tunable considerations

Table 1-3 shows some other network tunables that should be considered and other ways to calculate some of the values in Table 1-2 on page 36.

Table 1-3 Other basic network tunables

tunable name	Comment
thewall	Use the default or if network errors occur ^a , set manually to a higher value. no -o thewall shows the current setting.
tcp_pmtu_discover	Disable Path Maximum Transfer Unit (PMTU) discovery by setting this option to 0 (zero) if the server communicates with more than 64 other systems ^b . This option enables TCP to dynamically find the largest size packet to send through the network, which will be as big as the smallest MTU size in the network.
sb_max	<p>Could be set to slightly less than thewall, or at two to four times the size of the largest value for tcp_sendspace, tcp_recvspace, udp_sendspace, and udp_recvspace.</p> <p>This parameter controls how much buffer space is consumed by buffers that are queued to a sender's socket or to a receiver's socket. A socket is just a queuing point, and it represents the file descriptor for a TCP session. tcp_sendspace, tcp_recvspace, udp_sendspace, and udp_recvspace parameters cannot be set larger than sb_max.</p> <p>The system accounts for socket buffers used based on the size of the buffer, not on the contents of the buffer. For example, if an Ethernet driver receives 500 bytes into a 2048-byte buffer and then this buffer is placed on the applications socket awaiting the application reading it, the system considers 2048 bytes of buffer to be used. It is common for device drivers to receive buffers into a buffer that is large enough to receive the adapter's maximum size packet. This often results in wasted buffer space, but it would require more CPU cycles to copy the data to smaller buffers. Because the buffers often are not 100 percent full of data, it is best to have sb_max to be at least twice as large as the TCP or UDP receive space. In some cases for UDP it should be much larger.</p> <p>Once the total buffers on the socket reach the sb_max limit, no more buffers will be allowed to be queued to that socket.</p>
tcp_sendspace	This parameter mainly controls how much buffer space in the kernel (mbuf) will be used to buffer data that the application sends. Once this limit is reached, the sending application will be suspended until TCP sends some of the data, and then the application process will be resumed to continue sending.

tunable name	Comment
tcp_recvspace	This parameter has two uses. First, it controls how much buffer space may be consumed by receive buffers. Second, TCP uses this value to inform the remote TCP how large it can set its transmit window to. This becomes the "TCP Window size." TCP will never send more data than the receiver has buffer space to receive the data into. This is the method by which TCP bases its flow control of the data to the receiver.
udp_sendspace	Always less than udp_recvspace but never greater than 65536 because UDP transmits a packet as soon as it gets any data and IP has an upper limit of 65536 bytes per packet.
udp_recvspace	Always greater than udp_sendspace and sized to handle as many simultaneous UDP packets as can be expected per UDP socket. For single parent/multiple child configurations, set udp_recvspace to udp_sendspace times the maximum number of child nodes if UDP is used, or at least 10 times udp_sendspace.
tcp_mssdflt	This setting is used for determining MTU sizes when communicating with remote networks. If not changed and MTU discovery is not able to determine a proper size, communication degradation ^c may occur. The default value for this option is 512 bytes and is based on the convention that all routers should support 576 byte packets. Calculate a proper size by using the following formula; $MTU - (IP + TCP\ header)^d$.
ipqmaxlen	Could be set to 512 when using file sharing with applications such as GPFS.
tcp_nagle_limit	Could be set to 0 to disable the Nagle Algorithm when using large buffers.
fasttimo	Could be set to 50 if transfers take a long time due to delayed ACKs.
rfc1323	This option enables TCP to use a larger window size, at the expense of a larger TCP protocol header. This enables TCP to have a 4 GB window size. For adapters that support a 64K MTU (frame size), you must use RFC1323 to gain the best possible TCP performance.

- a. It is set automatically by calculating the amount of memory available.
- b. In a heterogeneous environment the value determined by MTU discovery can be way off.
- c. When setting this value, make sure that all routing equipment between the sender and receiver can handle the MTU size; otherwise they will fragment the packets.

d. The size depends on the original MTU size and if RFC1323 is enabled or not. If RFC1323 is enabled, then the IP and TCP header is 52 bytes, if RFC1323 is not enabled, the IP and TCP header is 40 bytes.

To document all network interfaces and important device settings, you can manually check all interface device drivers with the `lsattr` command as is shown in Example 1-4.

Basic network adapter settings

Network adapters should be set to utilize the maximum transfer capability of the current network given available system memory. On large server systems (such as database server or Web servers with thousands of concurrent connections), you might need to set the maximum values allowed for network device driver queues if you use Ethernet or token-ring network adapters. However, note that each queue entry will occupy memory at least as large as the MTU size for the adapter.

To find out the maximum possible setting for a device, use the `lsattr` command as shown in the following examples. First find out the attribute names of the device driver buffers/queues that the adapter uses. (These names can vary for different adapters.) Example 1-4 is for an Ethernet network adapter interface using the `lsattr` command.

Example 1-4 Using lsattr on an Ethernet network adapter interface

```
# lsattr -El ent0
busmem      0x1ffac000      Bus memory address          False
busintr     5               Bus interrupt level         False
intr_priority 3             Interrupt priority          False
rx_que_size 512            Receive queue size          False
tx_que_size 8192           Software transmit queue size True
jumbo_frames no             Transmit jumbo frames       True
media_speed Auto_Negotiation Media Speed (10/100/1000 Base-T Ethernet) True
use_alt_addr no             Enable alternate ethernet address True
alt_addr    0x0000000000000000 Alternate ethernet address  True
trace_flag  0              Adapter firmware debug trace flag True
copy_bytes  2048           Copy packet if this many or less bytes True
tx_done_ticks 1000000       Clock ticks before TX done interrupt True
tx_done_count 64            TX buffers used before TX done interrupt True
receive_ticks 50            Clock ticks before RX interrupt True
receive_bds  6             RX packets before RX interrupt True
receive_proc 16            RX buffers before adapter updated True
rxdesc_count 1000          RX buffers processed per RX interrupt True
stat_ticks  1000000       Clock ticks before statistics updated True
rx_checksum yes           Enable hardware receive checksum True
flow_ctrl   yes           Enable Transmit and Receive Flow Control True
slih_hog    10            Interrupt events processed per interrupt True
```

Example 1-5 shows what it might look like on a token-ring network adapter interface using the **lsattr** command.

Example 1-5 Using lsattr on a token-ring network adapter interface

```
# lsattr -E1 tok0
busio      0x7fffc00 Bus I/O address          False
busintr    3          Bus interrupt level         False
xmt_que_size 16384     TRANSMIT queue size         True
rx_que_size 512       RECEIVE queue size          True
ring_speed 16        RING speed                   True
attn_mac   no        Receive ATTENTION MAC frame True
beacon_mac no        Receive BEACON MAC frame    True
use_alt_addr no       Enable ALTERNATE TOKEN RING address True
alt_addr   0x       ALTERNATE TOKEN RING address True
full_duplex yes     Enable FULL DUPLEX mode     True
```

To find out the maximum possible setting for a device attribute, use the **lsattr** command with the **-R** option on each of the adapters' queue attributes as in Example 1-6.

Example 1-6 Using lsattr to find out attribute ranges for a network adapter interface

```
# lsattr -R1 ent0 -a tx_que_size
512...16384 (+1)
# lsattr -R1 ent0 -a rx_que_size
512
# lsattr -R1 tok0 -a xmt_que_size
32...16384 (+1)
# lsattr -R1 tok0 -a rx_que_size
32...512 (+1)
```

In the example output, for the Ethernet adapter the maximum values for `tx_que_size` and `rx_que_size` are 16384 and 512. For the token-ring adapter the maximum values in the example output above for `xmt_que_size` and `rx_que_size` is are also 16384 and 512. When only one value is shown it means that there is only one value to use and it cannot be changed. When an ellipsis (...) separates values it means an interval between the values surrounding the dotted line in increments shown at the end of the line within parenthesis, such as in the example above (+1), which means by increments of one.

To change the values so that they will be used the next time the device driver is loaded, use the **chdev** command as shown in Example 1-7. Note that with the **-P** attribute, the changes will be effective after the next IPL.

Example 1-7 Using chdev to change a network adapter interface attributes

```
# chdev -l ent0 -a tx_que_size=16384 -a rx_que_size=512 -P
ent0 changed
```

```
# chdev -l tok0 -a xmt_que_size=16384 -a rx_que_size=512 -P
tok0 changed
```

The commands **atmstat**, **entstat**, **fddistat**, and **tokstat** can be used to monitor the use of transmit buffers for a specific network adapter. Refer to Chapter 29, “atmstat, entstat, estat, fddistat, and tokstat commands” on page 539 for more details about these commands.

The MTU sizes for a network adapter interface can be examined by using the **lsattr** command and the **mtu** attribute as in Example 1-8, which shows the **tr0** network adapter interface.

Example 1-8 Using lsattr to examine the possible MTU sizes for a network adapter

```
# lsattr -R -a mtu -l tr0
60...17792 (+1)
```

The minimum MTU size for token-ring is 60 bytes and the maximum size is just over 17 KB. Example 1-9 shows the allowable MTU sizes for Ethernet (**en0**).

Example 1-9 Using lsattr to examine the possible MTU sizes for Ethernet

```
# lsattr -R -a mtu -l en0
60...9000 (+1)
```

Note that 9000 as a maximum MTU size is only valid for Gigabit Ethernet; 1500 is still the maximum for 10/100 Ethernet.

Resetting network tunables to their default

Should you need to set all **no** tunables back to their default value, the following commands are one way to do it:

```
no -a | awk '{print $1}' | xargs -t -i no -d {}
no -o extendednetstats=0
```

Attention: The default value for the network option **extendednetstats** is 1 (one) to enable the collection of extended network statistics. Normally these extended network statistics should be disabled using the command **no -o extendednetstats=0**. Refer to Chapter 31, “The netstat command” on page 619 and Chapter 34, “The no command” on page 665 for more information about the effects of the **extendednetstats** option.

Some high-speed adapters have **ISNO** parameters set by default in the ODM database. Review the *AIX 5L Version 5.2 System Management Guide: Communications and Networks* for individual adapters default values, or use the **lsattr** command with the **-D** option as in Example 1-10 on page 42.

Example 1-10 Using lsattr to list default values for a network adapter

# lsattr -HD -l ent0			
attribute	deflt	description	user_settable
busmem	0	Bus memory address	False
busintr		Bus interrupt level	False
intr_priority	3	Interrupt priority	False
rx_que_size	512	Receive queue size	False
tx_que_size	8192	Software transmit queue size	True
jumbo_frames	no	Transmit jumbo frames	True
media_speed	Auto_Negotiation	Media Speed (10/100/1000 Base-T Ethernet)	True
use_alt_addr	no	Enable alternate ethernet address	True
alt_addr	0x000000000000	Alternate ethernet address	True
trace_flag	0	Adapter firmware debug trace flag	True
copy_bytes	2048	Copy packet if this many or less bytes	True
tx_done_ticks	1000000	Clock ticks before TX done interrupt	True
tx_done_count	64	TX buffers used before TX done interrupt	True
receive_ticks	50	Clock ticks before RX interrupt	True
receive_bds	6	RX packets before RX interrupt	True
receive_proc	16	RX buffers before adapter updated	True
rxdesc_count	1000	RX buffers processed per RX interrupt	True
stat_ticks	1000000	Clock ticks before statistics updated	True
rx_checksum	yes	Enable hardware receive checksum	True
flow_ctrl	yes	Enable Transmit and Receive Flow Control	True
slh_hog	10	Interrupt events processed per interrupt	True

The deflt column shows the default values for each attribute. Example 1-11 shows how to use them on an Ethernet network adapter interface.

Example 1-11 Using lsattr to list default values for a network interface

# lsattr -HD -l en0			
attribute	deflt	description	user_settable
mtu	1500	Maximum IP Packet Size for This Device	True
remmtu	576	Maximum IP Packet Size for REMOTE Networks	True
netaddr		Internet Address	True
state	down	Current Interface Status	True
arp	on	Address Resolution Protocol (ARP)	True
netmask		Subnet Mask	True
security	none	Security Level	True
authority		Authorized Users	True
broadcast		Broadcast Address	True
netaddr6	N/A		True
alias6	N/A		True
prefixlen	N/A		True
alias4	N/A		True
rfc1323	N/A		True
tcp_nodelay	N/A		True

tcp_sendspace	N/A	True
tcp_recvspace	N/A	True
tcp_mssdf1t	N/A	True

Default values should be listed in the `def1t` column for each attribute. If no value is shown, it means that there is no default setting.

1.6 Kernel tunables

Starting with AIX 5L Version 5.2, there is a more consistent method of tuning the AIX kernel parameters. Rather than having commands that work in different ways, new commands such as **schedo**, **vmo**, and **ioo** were added and some old commands such as **no** and **nfso** were enhanced. Also, the tuning capabilities are now implemented in System Management Interface (SMIT) panels. The parameter values are now stored in stanza files in the directory `/etc/tunables`.

The discussion here consists of:

- ▶ 1.6.1, “Tunables commands” on page 43
- ▶ 1.6.2, “Tunable files” on page 45

1.6.1 Tunables commands

The commands that manipulate these tuning parameters are:

- ▶ New commands: **vmo** and **ioo** (which replaced **vmtune**) and **schedo** (which replaced **schedtune**). The **ioo**, **vmo**, and **schedo** commands reside in `/usr/sbin` and are part of the `bos.perf.tune` fileset, which is installable from the AIX base installation media.

See Chapter 14, “The **vmo**, **ioo**, and **vmtune** commands” on page 229 for further information about **vmo** and **ioo** commands.

See 10.1, “**schedo**” on page 166 for further information about the **schedo** tuning command.

- ▶ Enhanced old commands, such as **no** and **nfso**. The **no** command resides in `/usr/sbin` and is part of the `bos.net.tcp.client` fileset, which is installable from the AIX base installation media. The **nfso** command resides in `/usr/sbin` and is part of the `bos.net.nfs.client` fileset, which is installable from the AIX base installation media.

See Chapter 32, “The **nfso** command” on page 645 for further information about the **nfso** tuning command.

See Chapter 34, “The **no** command” on page 665 for further information about the **no** tuning command.

The **no**, **nfso**, **vmo**, **ioo**, and **schedo** tuning commands all support this syntax:

```
command [-p|-r] {-o tunable[=newvalue]}
command [-p|-r] {-d tunable}
command [-p|-r] -D
command [-p|-r] -a
command -h tunable
command -L [tunable]
```

Flags

- p** Applies changes for current and next reboot.
- r** Applies changes for next reboot only.
- o tunable[=newvalue]** Displays the value or sets tunable to newvalue.
- d tunable** Resets tunable to default value.
- D** Resets all tunables to their default value.
- a** Displays current, reboot (when used in conjunction with -r) or permanent (when used in conjunction with -p) value for all tunable parameters.
- h [tunable]** Displays help about tunable parameter.
- L [tunable]** Lists the characteristics of one or all tunables, one per line.
- x [tunable]** Provides a comma-separated result similar to the -L, appropriate for loading into a spreadsheet.

Permanent kernel-tuning changes are achieved by centralizing the reboot values for all tunable parameters in the `/etc/tunables/nextboot` stanza file. When a system is rebooted, the values in the `/etc/tunables/nextboot` file are applied automatically.

The following commands are used to manipulate the nextboot file and other files containing a set of tunable parameter values:

- ▶ **tunsave**: Saves tunable values to a stanza file.
- ▶ **tunrestore**: Restores all of the parameters from a file.
- ▶ **tuncheck**: Must be used to validate a file created manually.
- ▶ **tundefault**: Used to force reset of all tuning parameters to their default value.
- ▶ **tunchange**: Updates stanzas in tunables files.

The **tunsave**, **tunrestore**, **tuncheck**, **tundefault**, and **tunchange** commands reside in `/usr/sbin` and are part of the `bos.perf.tune` fileset, which is installable from the AIX base installation media. For more information about these commands, refer to Chapter 15, “Kernel tunables commands” on page 255.

1.6.2 Tunable files

All of the tunable parameters manipulated by the tuning commands (**no**, **nfso**, **vmo**, **ioo**, and **schedo**) have been classified into six categories:

Dynamic	The parameter can be changed at any time.
Static	The parameter can never be changed.
Reboot	The parameter can only be changed during reboot.
Bosboot	The parameter can only be changed by running bosboot and rebooting the machine.
Connect	The parameter only applies to future socket connections, changes of this type of parameter automatically restart inetd
Mount	The parameter changes are only effective for future file systems or directory mounts.
Incremental	The parameter can only be incremented, except at boot time.

The main page for each of the five tuning commands contains the complete list of all parameters manipulated by each of the commands, and for each parameter, its type, range, default value, and any dependencies on other parameters.

These files under `/etc/tunables` are used for storing these tunable parameters:

nextboot	This file is automatically applied at boot time. The bosboot command also gets the value of Bosboot-type tunables from this file. It contains all tunable settings made permanent. See Example 1-12 on page 46.
lastboot	This file is automatically generated at boot time. It contains the full set of tunable parameters with their values after the last boot. Default values are marked with <code># DEFAULT VALUE</code> .

The `lastboot.log` is automatically generated at boot time. It contains the logging of the creation of the `lastboot` file, such as:

- ▶ Any parameter changes made
- ▶ Any parameter changes that could not be made (for example, if the `nextboot` file was created manually and not validated with **tuncheck**)

Tunable files can be created and modified using one of the following options:

- ▶ Using **smit** or Web-based System Manager to:
 - Modify the next reboot value for tunable parameters

- Ask to save all current values for next boot
- Ask to use an existing tunable file at the next reboot

All of these actions will update the `/etc/tunables/nextboot` file. A new file in the `/etc/tunables` directory can also be created to save all current or all nextboot values.

- ▶ Using the tuning commands (`ioo`, `vmo`, `schedo`, `no`, or `nfso`) with the `-p` or `-r` options, which will update the `/etc/tunables/nextboot` file.
- ▶ A new file in the `/etc/tunables` directory can also be created directly with an editor or copied from another machine. Running `tuncheck [-r | -p] -f` must then be done on that file.
- ▶ Using the `tunsave` command to create or overwrite files in the `/etc/tunables` directory.
- ▶ Using the `tunrestore -r` command to update the nextboot file.

An example of the nextboot file is provided in Example 1-12.

Example 1-12 nextboot file

```
vmo:
    maxfree = "128"
    minfree = "120"

ioo:
    maxpgahead = "8"

no:
    ipforwarding = "0"

nfso:
    nfs_v3_vm_bufs = "5000"

schedo:
    sched_R = "16"
```

1.7 The `/proc` file system

The `/proc` file system is created with the initial installation of AIX 5L Version 5.1 and later. It is a pseudo file system that maps processes and kernel data structures to corresponding files and contains state information about processes and threads in the system.

Example 1-13 shows the output of the **mount** and **df** commands showing /proc.

Example 1-13 proc filesystems attributes

```
lpar05:/>> mount
node          mounted          mounted over    vfs      date          options
-----
/dev/hd4      /
/dev/hd2      /usr
/dev/hd9var   /var
/dev/hd3      /tmp
/dev/hd1      /home
/proc        /proc            procfs         Apr 21 16:45 rw
/dev/hd10opt  /opt
/dev/lv00     /home/db2inst1  jfs            Apr 21 16:45 rw,log=/dev/loglv00
/dev/lv02     /home/db2as     jfs            Apr 21 16:45 rw,log=/dev/loglv00
/dev/lv03     /home/db2fenc1  jfs            Apr 21 16:45 rw,log=/dev/loglv00
/dev/lv06     /work           jfs            Apr 21 16:46 rw,log=/dev/loglv02

lpar05:/>> df -k
Filesystem    1024-blocks    Free %Used    Iused %Iused Mounted on
/dev/hd4      32768          13164 60%         1675 11% /
/dev/hd2      4882432        2446384 50%         54536 5% /usr
/dev/hd9var   491520         420880 15%          679 1% /var
/dev/hd3      327680         205596 38%          155 1% /tmp
/dev/hd1      32768          7000 79%          344 5% /home
/proc        -              -      -           -    - /proc
/dev/hd10opt  425984         405724 5%           282 1% /opt
/dev/lv00     1048576        974164 8%           435 1% /home/db2inst1
/dev/lv02     131072         117880 11%          75 1% /home/db2as
/dev/lv03     32768          31700 4%           17 1% /home/db2fenc1
/dev/lv06     1048576        1015612 4%           17 1% /work
```

Each process is assigned a directory entry in the /proc file system with a name identical to its process ID. In this directory, several files and subdirectories are created corresponding to internal process control data structures. Most of these files are read-only, but some of them can also be written to and be used for process control purposes. In addition, if a process becomes a zombie, most of its associated /proc files disappear from the directory structure.

The /proc files contain data that presents the state of processes and threads in the system. This state is constantly changing while the system is operating. To lessen the load on system performance caused by reading /proc files, the /proc filesystem does not stop system activity while gathering the data for those files. A single read of a /proc file generally returns a coherent and fairly accurate representation of process or thread state. However, because the state changes as the process or thread runs, multiple reads of /proc files may return representations that show different data and therefore appear to be inconsistent with each other.

An atomic representation is a representation of the process or thread at a single and discrete point in time. If you want an atomic snapshot of process or thread state, stop the process and thread before reading the state. There is no guarantee that the data is an atomic snapshot for successive reads of /proc files for a running process. In addition, a representation is not guaranteed to be atomic for any I/O applied to the address space file. The contents of any process address space might be simultaneously modified by a thread of that process or any other process in the system.

Important: Multiple structure definitions are used to describe the /proc files. A /proc file may contain additional information other than the definitions presented here. In future releases of the operating system, these structures may grow by the addition of fields at the end of the structures.

The content of the /proc/pid directory is shown in Example 1-14.

Example 1-14 Content of /proc/pid

```
lpar05:/proc/454698>> ls -la
total 8
-rw----- 1 root    system    0 Apr 24 13:20 as
-r----- 1 root    system    128 Apr 24 13:20 cred
--w----- 1 root    system    0 Apr 24 13:20 ctl
lr-x----- 22 root    system    0 Apr  2 16:34 cwd ->
/usr/WebSphere/AppServer/
dr-x----- 1 root    system    0 Apr 24 13:20 fd
dr-xr-xr-x 1 root    system    0 Apr 24 13:20 lwp
-r----- 1 root    system    0 Apr 24 13:20 map
dr-x----- 1 root    system    0 Apr 24 13:20 object
-r--r--r-- 1 root    system    448 Apr 24 13:20 psinfo
-r----- 1 root    system    12288 Apr 24 13:20 sigact
-r----- 1 root    system    1520 Apr 24 13:20 status
-r--r--r-- 1 root    system    0 Apr 24 13:20 sydent
```

The following are the files and directories that exist for each process in the /proc filesystem:

/proc/pid	Directory for the process PID.
/proc/pid/as	Address space of process PID.
/proc/pid/cred	Contains a description of the credentials associated with the process.
/proc/pid/ctl	Control file for process PID.
/proc/pid/cwd	A link that provides access to the current working directory of the process. Any process can access the

	current working directory of the process through this link, provided it has the necessary permissions.
/proc/pid/fd	Contains files for all open file descriptors of the process.
/proc/pid/map	Address space map info for process PID.
/proc/pid/object	Directory for objects for process PID.
/proc/pid/psinfo	Process status info for process PID.
/proc/pid/sigact	Signal actions for process PID.
/proc/pid/status	Status of process PID.
/proc/pid/sysent	System call information for process PID.

Some of the files relate to a specific threads within the process. Those are:

/proc/pid/lwp/tid	Directory for thread TID
/proc/pid/lwp/tid/lwpctl	Control file for thread TID
/proc/pid/lwp/tid/lwpinfo	Process status info for thread TID
/proc/pid/lwp/tid/lwpstatus	Status of thread TID

The pseudo file named *as* enables you to access the address space of the process, and as it can be seen by the *rw* (read/write) access flags, you can read and write to the memory belonging to the process.

It should be understood that only the user regions of the process's address can be written to under */proc*. Also, a copy of the address space of the process is made while tracing under */proc*. This is the address space that can be modified. This is done when the *as* file is closed; the original address space is unmodified.

The *cred* file provides information about the credentials associated with this process. Writing to the *ctl* file enables you to control the process; for example, to stop or to resume it. The *map* file allows access to the virtual address map of the process. Information usually shown by the *ps* command can be found in the *psinfo* file, which is readable for all system users. The current status of all signals associated with this process is recorded in the *sigact* file. State information for this process, such as the address and size of the process heap and stack (among others), can be found in the *status* file. Finally, the *sysent* file allows you to check for the system calls available to this process.

The object directory contains files with names as they appear in the *map* file. These files correspond to files mapped in the address space of the process. The content of the */proc/pid/object* directory is shown in Example 1-15 on page 50.

Example 1-15 Content of /proc/pid/object

```
lpar05:/proc/454698/object>> ls -la
total 38760
dr-x----- 1 root    system      0 Apr 24 13:58 .
dr-xr-xr-x  1 root    system      0 Apr 24 13:58 ..
-rwxr-xr-x  1 root    system    45835 Nov 11 10:15 a.out
-r--r--r--  1 bin     bin      5926092 Mar  6 23:38 jfs.10.5.16392
-r-xr-xr-x  1 bin     bin      6785519 Sep 19 2002 jfs.10.5.4132
-r-xr-xr-x  1 bin     bin      10993 Sep 15 2002 jfs.10.5.4144
-r--r--r--  1 bin     bin     909148 Sep 20 2002 jfs.10.5.4159
-r-xr-xr-x  1 bin     bin     60890 Sep 15 2002 jfs.10.5.4188
-rwxr-xr-x  1 root    system   2548621 Nov 11 10:15 jfs.10.5.530543
-rw-r--r--  1 root    system    35088 Nov 11 19:52 jfs.10.5.539133
-rwxr-xr-x  1 root    system   134782 Nov 11 10:15 jfs.10.5.540408
-rwxr-xr-x  1 root    system   364159 Nov 11 10:15 jfs.10.5.540410
-rwxr-xr-x  1 root    system  2716599 Nov 11 10:15 jfs.10.5.540414
-rwxr-xr-x  1 root    system    71838 Nov 11 10:15 jfs.10.5.540417
-rwxr-xr-x  1 root    system     9738 Nov 11 10:15 jfs.10.5.540418
-rwxr-xr-x  1 root    system   45336 Nov 11 10:15 jfs.10.5.540419
-rwxr-xr-x  1 root    system  104961 Nov 11 10:15 jfs.10.5.540420
-rwxr-xr-x  1 root    system   45835 Nov 11 10:15 jfs.10.5.604167
```

The a.out file always represents the executable binary file for the program running in the process itself because the example program is written in C and must use the C runtime library, as indicated by the other file references. To get the actual corresponding file names from the symbolic file, use the **ls** command to get the *major* and *minor* device numbers, and the *inode number* that can be queried using the **ncheck** command. The example that we use checks whether jfs.10.5.4132 is used to find a file belonging to an inode in a specific file system.

Example 1-16 To check inode number and correspondent file

```
lpar05:/proc/454698/object>> ls -l /dev/hd2
brw-rw---- 1 root    system      10,  5 Apr  2 15:03 /dev/hd2

lpar05:/proc/454698/object>> ncheck -i 4132 /dev/hd2
/dev/hd2:
4132    /ccs/lib/libc.a
```

The lwp directory has subdirectory entries for each kernel thread running in the process. The term lwp stands for lightweight process and is the same as the term thread used in the AIX documentation. It is used in the context of the /proc file system to keep a common terminology with the /proc implementation of other operating systems. The names of the subdirectories are the thread IDs. The program has threads, as shown in the output of the **ps** command. Therefore, only the content of one of these thread directoris is shown in Example 1-17 on page 51.

Example 1-17 Displaying threads with the ps command

```
lpar05:/proc/454698/object>> ps -mo THREAD -p 454698
USER  PID  PPID    TID S  CP PRI SC   WCHAN          F    TT BND COMMAND
... ( lines omitted)...
-    -    -    979177 S   0  60  1 f10000879000ef40  8410400  - - -
-    -    -    983271 S   0  60  1 f10000879000f040  8410400  - - -
-    -    -    991463 S   0  60  1 f1000089c1684a00   400400  - - -
-    -    -    995567 S   0  60  1 f1000089c16dc200   400400  - - -
-    -    -    999661 S   0  60  1 f1000089c1684200   400400  - - -
-    -    -    1003761 S  0  60  1 f1000089c16dca00   400400  - - -
-    -    -    1007857 S  0  60  1 f10000879000f640  8410400  - - -
-    -    -    1024171 S  0  60  1 f10000879000fa40  8410400  - - -
lpar05:/proc/454698/lwp>> cd 1024171
lpar05:/proc/454698/lwp/1024171>> ls -la
total 0
dr-xr-xr-x  1 root    system          0 Apr 24 16:25 .
dr-xr-xr-x  1 root    system          0 Apr 24 16:25 ..
--w-----  1 root    system          0 Apr 24 16:25 lwpctl
-r--r--r--  1 root    system         120 Apr 24 16:25 lwpsinfo
-r-----  1 root    system         1200 Apr 24 16:25 lwpstatus
```

The `lwpctl`, `lwpsinfo`, and `lwpstatus` files contain thread-specific information to control this thread, for the `ps` command, and about the state, similar to the corresponding files in the `/proc/pid` directory. As an example of what can be obtained from reading these files, Example 1-18 shows the content of the `cred` file.

Example 1-18 Using the od command to show the content of the cred file

```
lpar05:/proc/454698>> ls -l cred
-r-----  1 root    system         128 Apr 24 16:45 cred
lpar05:/proc/454698>> od -x cred
0000000  0000 0000 0000 0000 0000 0000 0000 0000
*
0000160  0000 0000 0000 0007 0000 0000 0000 0000
0000200  0000 0000 0000 0002 0000 0000 0000 0003
0000220  0000 0000 0000 0007 0000 0000 0000 0008
0000240  0000 0000 0000 000a 0000 0000 0000 000b
0000260
```

The output in the leftmost column shows the byte offset of the file in octal representation. The remainder of the lines are the actual content of the file in hexadecimal notation. Even if the directory listing shows the size of the file to be 128 bytes or 0200 bytes in octal, the actual output is 0260 or 176 bytes in size. This is due to the dynamic behavior of the last field in the corresponding structure. The digit 7 in the 0160 line specifies the number of groups the user ID running this process belongs to. Because every user ID is at least part of its

primary group, but possibly belongs to a number of other groups that cannot be known in advance, only space for the primary group is reserved in the *cred* data structure. In this case, the primary group ID is zero because the user ID running this process is root. Reading the complete content of the file, nevertheless, reveals all of the other group IDs the user currently belongs to. The group IDs in this case (2, 3, 7, 8, 0xa (10), and 0xb (11)) map to the groups bin, sys, security, cron, audit, and lp. This is exactly the set of groups the user ID root belongs to by default.

The `/proc/pid#/fd` directory contains files for all the open file descriptors of the process. As seen in the example, each entry is a decimal number that corresponds to an open file descriptor in the process. Any directories are displayed as links. Example 1-19 shows the directory layout for a process.

Example 1-19 Using the ls command

```
lpar05:/proc/454698/fd>> ls -l
total 149032
... ( lines omitted)...
c----- 1 root system 21, 1 Apr 24 17:17 0
-rw-r--r-- 1 root system 0 Apr 2 17:06 1
-r-xr-xr-x 1 root system 6538 Jul 9 2002 10
-r--r--r-- 1 root system 2425671 Nov 11 19:38 100
-r--r--r-- 1 root system 135580 Nov 11 21:50 101
-r--r--r-- 1 root system 24815 Nov 11 22:35 102
-r--r--r-- 1 root system 15052 Nov 11 19:39 103
-r--r--r-- 1 root system 417110 Oct 11 2001 104
-r--r--r-- 1 root system 1201599 Sep 24 2002 105
-r--r--r-- 1 root system 41007 Nov 11 21:49 106
-r--r--r-- 1 root system 557450 Nov 11 19:01 107
-r--r--r-- 1 root system 1154258 Nov 11 19:06 108
-r--r--r-- 1 root system 82298 Nov 11 19:03 109
-r-xr-xr-x 1 root system 315235 Oct 9 2002 11
-r--r--r-- 1 root system 1883329 Oct 18 2002 110
-r--r--r-- 1 root system 3173 Nov 11 19:02 111
-r--r--r-- 1 root system 124331 Nov 11 19:02 112
-r--r--r-- 1 root system 174782 Nov 11 19:03 113
-r--r--r-- 1 root system 383415 Nov 11 19:03 114
-r--r--r-- 1 root system 342484 Sep 3 2002 115
-r--r--r-- 1 root system 67138 Mar 1 2001 116
```

Getting started

This chapter is intended as a starting point. It contains listings of all of the common and most useful AIX tools for resolving and monitoring performance issues. The quick-lookup tables in this chapter are intended to assist the user in finding the required command for monitoring a certain system resource and to provide the user with information about which AIX fileset a tool might belong to.

When facing a performance problem on a system, an approach must be chosen in order to analyze and resolve the problem. The **topas** command is an AIX performance monitoring tool that gives an overview of all of the system resources and can therefore very well be used as a starting point for performance analysis.

The discussions in this chapter are:

- ▶ 2.1, “Tools and filesets” on page 54
- ▶ 2.2, “Tools by resource matrix” on page 57
- ▶ 2.3, “Performance tuning approach” on page 60, which shows the user the recommended approach to resolving a performance problem, starting with **topas**, and guides the user through the performance analysis task.

2.1 Tools and filesets

The intention of this section is to give you an list of all the performance tools discussed in this book together with the path that is used to call the command and the fileset the tool is part of.

Many of the performance tools are located in filesets that obviously would contain them, such as `bos.perf.tools` or `perfagent.tools`. However, some are located in filesets that are not quite as obvious. You will often find that this fileset is not installed on a system because it does not obviously contain performance tools.

One example is the `vmtune` and `schedtune` commands, which are both part of the `bos.adt.samples` fileset. However, starting with AIX 5.2, `schedtune` and `vmtune` are only scripts that call the new commands `vmo`, `ioo`, and `schedo`. For more information see Chapter 14, “The `vmo`, `ioo`, and `vmtune` commands” on page 229 and Chapter 10, “The `schedo` and `schedtune` commands” on page 165.

Table 2-1 lists the tools discussed in this book, their full path name, and their fileset information.

Table 2-1 *Commands/tools, pathnames, and filesets*

Command / Tool	Full path name	Fileset name / URL
3D monitor	/usr/bin/3dmon	perfmgr.network
alstat	/usr/bin/alstat	bos.perf.tools
atmstat	/usr/bin/atmstat	devices.common.IBM.atm.rte
bindintcpu	/usr/sbin/bindintcpu	devices.chrp.base.rte
bindprocessor	/usr/sbin/bindprocessor	bos.mp
curt	/usr/bin/curt	bos.perf.tools
emstat	/usr/bin/emstat	bos.perf.tools
entstat	/usr/bin/entstat	devices.common.IBM.ethernet.rte
estat	/usr/lpp/ssp/css/css	ssp.css
fddistat	/usr/bin/fddistat	devices.common.IBM.fddi.rte
fdpr	/usr/bin/fdpr	perfagent.tools
filemon	/usr/bin/filemon	bos.perf.tools
fileplace	/usr/bin/fileplace	bos.perf.tools
genkex	/usr/bin/genkex	bos.perf.tools

Command / Tool	Full path name	Fileset name / URL
genkld	/usr/bin/genkld	bos.perf.tools
genld	/usr/bin/genld	bos.perf.tools
gennames	/usr/bin/gennames	bos.perf.tools
gprof	/usr/bin/gprof	bos.adt.prof
ioo	/usr/sbin/ioo	bos.perf.tune
iostat	/usr/bin/iostat	bos.acct
ipcs	/usr/bin/ipcs	bos.rte.control
ipfilter	/usr/bin/ipfilter	bos.perf.tools
ipreport	/usr/sbin/ipreport	bos.net.tcp.server
iptrace	/usr/sbin/iptrace	bos.net.tcp.server
jazizo (PTX)	/usr/bin/jazizo	perfmgr.analysis.jazizo
locktrace	/usr/bin/locktrace	bos.perf.tools
lsiv	/usr/sbin/lsiv	bos.rte.lvm
lspv	/usr/sbin/lspv	bos.rte.lvm
lsvg	/usr/sbin/lsvg	bos.rte.lvm
lvmstat	/usr/sbin/lvmstat	bos.rte.lvm
netpmon	/usr/bin/netpmon	bos.perf.tools
netstat	/usr/bin/netstat	bos.net.tcp.client
nfso	/usr/sbin/nfso	bos.net.nfs.client
nfsstat	/usr/sbin/nfsstat	bos.net.nfs.client
nice	/usr/bin/nice	bos.rte.control
no	/usr/sbin/no	bos.net.tcp.client
PDT	/usr/sbin/perf/diag_tool	bos.perf.diag_tool
perfpmr	-	ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/
Perfstat API	-	bos.perf.libperfstat
Performance Monitor API	-	bos.pmapi.lib

Command / Tool	Full path name	Fileset name / URL
pprof	/usr/bin/pprof	bos.perf.tools
prof	/usr/bin/prof	bos.adt.prof
ps	/usr/bin/ps	bos.rte.control
renice	/usr/bin/renice	bos.rte.control
Resource Monitoring and Control	-	rsct.core.*
rmss	/usr/bin/rmss	bos.perf.tools
sar	/usr/sbin/sar	bos.acct
schedo	/usr/sbin/schedo	bos.perf.tune
splat	/usr/bin/splat	bos.perf.tools
System Performance Measurement Interface	-	perfagent.tools, perfagent.server
stripnm	/usr/bin/stripnm	bos.perf.tools
svmon	/usr/bin/svmon	bos.perf.tools
tcpdump	/usr/sbin/tcpdump	bos.net.tcp.server
time	/usr/bin/time	bos.rte.misc_cmds
timex	/usr/bin/timex	bos.acct
tokstat	/usr/bin/tokstat	devices.common.IBM.tokenring.rte
topas	/usr/bin/topas	bos.perf.tools
tprof	/usr/bin/tprof	bos.perf.tools
trace	/usr/bin/trace	bos.sysmgt.trace
trcnm	/usr/bin/trcnm	bos.sysmgt.trace
trcrpt	/usr/bin/trcrpt	bos.sysmgt.trace
trpt	/usr/sbin/trpt	bos.net.tcp.server
truss	/usr/bin/truss	bos.sysmgt.serv_aid
vmstat	/usr/bin/vmstat	bos.acct
vmo	/usr/sbin/vmo	bos.perf.tune

Command / Tool	Full path name	Fileset name / URL
wlmmon	/usr/bin/wlmmon	perfagent.tools
wlmpref	/usr/bin/wlmpref	perfmgr.analysis.jazizo
wlmstat	/usr/sbin/wlmstat	bos.rte.control
xmpref (PTX®)	/usr/bin/xmpref	perfmgr.network

2.2 Tools by resource matrix

Table 2-2 contains a list of the AIX monitoring and tuning tools and what system resources (CPU, Memory, Disk I/O, Network I/O) they obtain statistics for. Tools that are used by **trace**, that post-process the **trace** output, or that are directly related to **trace**, are denoted in the Trace Tools column. Tools that are useful for application development are checked in the Application column.

Table 2-2 Performance tools by resource matrix

Command	CPU	Memory	Disk I/O	Network I/O	Trace Tools	Application
alstat	x					
atmstat				x		
bindintcpu	x					
bindprocessor	x					
curt	x				x	
emstat	x					
entstat				x		
estat				x		
fddistat				x		
fdpr						x
filemon			x		x	
fileplace			x			
genkex					x	
genkld					x	
genld					x	

Command	CPU	Memory	Disk I/O	Network I/O	Trace Tools	Application
gennames					x	
gprof	x					x
ioo			x			
iostat	x		x			
ipcs		x				x
ipfilter				x		
ipreport				x		
iptrace				x		
locktrace	x				x	
lsiv			x			
lspv			x			
lsvg			x			
lvmstat			x			
netpmn	x			x	x	
netstat				x		
nfso				x		
nfsstat				x		
nice	x					
no				x		
PDT	x	x	x	x		
perfpmr	x	x	x	x	x	
Perfstat API	x	x	x	x		
Performance Monitor API	x					x
pprof	x				x	
prof	x					x
ps	x	x				

Command	CPU	Memory	Disk I/O	Network I/O	Trace Tools	Application
Performance Toolbox Version 3 for AIX	x	x	x	x		x
renice	x					
Resource Monitoring and Control	x	x	x	x		
rmss		x				
sar	x	x	x	x		
schedo	x	x				
splat	x				x	x
System Performance Measurement Interface	x	x	x	x		
stripanm						
svmon		x				
tcpdump				x		
time	x					
timex	x					
tokstat				x		
topas	x	x	x	x		
tprof	x				x	x
trace	x	x	x	x	x	
trcnm					x	
trcrpt					x	
trpt				x		x
truss					x	
vmstat	x	x	x			

Command	CPU	Memory	Disk I/O	Network I/O	Trace Tools	Application
vmo		x				
wlmmmon	x	x	x	x		
wlmpperf	x	x	x	x		
wlmstat	x	x	x	x		

2.3 Performance tuning approach

In this section, we discuss a typical initial approach to solve a performance problem. To determine which of the monitored performance values are high in a particular environment, it is necessary to gather the performance data on the system during an optimal performance state. This *baseline* performance information is very useful to have in case of a performance problem on the system. The **perfpmr** command can be used to gather this information. However, a screen snapshot of **topas** provides a brief overview of all of the major performance data that makes it easier to compare the values gathered on the well-performing system to the values shown if performance is low.

Note: In the following sections we rate the values of the **topas** output such as a high number of system calls. High, in this context, means that the value shown on the **topas** output of the currently low-performing system is higher than the value of the baseline performance data.

However, the values shown in the outputs of **topas** in the following sections do not necessarily reflect a performance problem. The outputs in our examples are only used to highlight the fields of interest.

In any case all four major resources (CPU, memory, disk I/O, and network) need to be checked when the performance of a system is analyzed.

2.3.1 CPU bound system

The output of **topas** in Example 2-1 shows the fields that are used to decide whether the system is CPU bound.

Example 2-1 topas output with highlighted CPU statistics

```

Topas Monitor for host:   wlmhost           EVENTS/QUEUES  FILE/TTY
Fri May 11 11:28:06 2001  Interval: 2    Cswitch      64  Readch      353
                               Sysca11      211 Writech     7836

```

Kernel	0.6					Reads	16	Rawin	0
User	99.3	#####				Writes	6	Ttyout	0
Wait	0.0					Forks	0	Igets	0
Idle	0.0					Execs	0	Namei	8
						Runqueue	4.0	Dirblk	0
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	Waitqueue	0.0		
tr0	8.3	6.1	9.2	0.3	8.0				
lo0	0.0	0.0	0.0	0.0	0.0	PAGING		MEMORY	
						Faults	0	Real,MB	511
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	Steals	0	% Comp	46.5
hdisk0	0.0	2.0	0.0	0.0	2.0	PgspIn	0	% Noncomp	53.6
hdisk1	0.0	0.0	0.0	0.0	0.0	PgspOut	0	% Client	49.6
						PageIn	0		
WLM-Class (Active)		CPU%	Mem%	Disk-I/O%		PageOut	0	PAGING SPACE	
Unmanaged		0	23	0		Sios	0	Size,MB	1024
Unclassified		0	0	0				% Used	13.1
						NFS (calls/sec)		% Free	86.8
Name	PID	CPU%	PgSp	Class		ServerV2	0		
dc	43564	25.0	0.3	System		ClientV2	0	Press:	
dc	21566	25.0	0.3	System		ServerV3	0	"h" for help	
dc	41554	25.0	0.3	VPs		ClientV3	0	"q" to quit	
dc	23658	24.2	0.3	System					

The fields of interest are:

- Kernel The CPU time spent in system (kernel) mode. The **tprof** or **trace** commands can be used for further problem determination of why the system spends more time than normal in system mode.
- User The CPU time spent in user mode. If the consumption is much higher than shown in the baseline, a user process may be looping. The output of **topas** may show this process in the process part (PID field for process ID). In case there are many active processes on the system and more than one looping user process, the **tprof** or **trace** command can be used to find these looping processes.
- Cswitch The number of context switches per second; this may vary. However, if this value is high, then the CPU system time also should be higher than normal. The **trace** command can be used for further investigation on the context switches.
- Syscall The number of system calls per second. If this value is higher than usual, the CPU system time also should be higher than normal. The **tprof** or **trace** commands can be used for further investigation on the system calls.
- Forks The number of fork system calls per second. See Execs below.
- Execs The number of exec system calls per second. If the number of fork or exec system calls is high, then the CPU system time also should

be higher than normal. A looping shell script that executes a number of commands may be the cause for the high fork and exec system calls. It may not be easy to find this shell script using the `ps` command. The AIX `trace` facility can be used for further investigation.

Runqueue The number of processes ready to run. If this number is high, either the number of programs run on the system increased (the load put on the system by the users), or there are fewer CPUs to run the programs. The `sar -P ALL` command should be used to see how all CPUs are used.

PID The process ID. Useful in case of a runaway process that causes CPU user time to be high. If there is a process using an unusually high amount of CPU time, the `tprof -t` command can be used to gather information about this process. If it is a runaway process, killing this process will reduce the high CPU usage and may solve the performance problem.

2.3.2 Memory bound system

The output of `topas` in Example 2-2 shows the fields that are used to decide whether the system is memory bound.

Example 2-2 topas output with highlighted memory statistics

```

Topas Monitor for host:  wlmhost          EVENTS/QUEUES  FILE/TTY
Fri May 11 11:28:06 2001  Interval:  2    Cswitch       64  Readch       353
                               Syscall       211  Writech      7836
Kernel   0.6  |                               Reads          16  Rawin         0
User     99.3 | #####                               Writes          6  Ttyout        0
Wait      0.0 |                               Forks           0  Igets         0
Idle      0.0 |                               Execs           0  Namei         8
                               Runqueue      4.0  Dirblk        0
                               Waitqueue     0.0
Network  KBPS  I-Pack  O-Pack  KB-In  KB-Out
tr0      8.3   6.1    9.2    0.3   8.0
lo0      0.0   0.0    0.0    0.0   0.0
PAGING          MEMORY
Faults          0  Real,MB       511
Steals          0  % Comp        46.5
PgspIn          0  % Noncomp     53.6
PgspOut         0  % Client      49.6
PageIn          0
PageOut         0  PAGING SPACE
Unmanaged       0   23           0  Sios          0  Size,MB      1024
Unclassified    0   0            0                % Used       13.1
                               NFS (calls/sec) % Free       86.8
Name            PID CPU% PgSp Class
dc              43564 25.0 0.3 System
dc              21566 25.0 0.3 System
ServerV2        0
ClientV2        0  Press:
ServerV3        0  "h" for help

```

```
dc      41554 25.0  0.3 VPs          ClientV3      0  "q" to quit
dc      23658 24.2  0.3 System
```

These are the fields of interest:

Steals This is the number of page steals per second by the VMM. If the system needs real memory, the VMM scans for the least referenced pages to free them. The `vmstat` command provides a statistic about the number of pages scanned. If a page to be stolen contains changed data, this page need to be written back to disk. Refer to `PgspOut` below. If the `Steals` value gets high, further investigation is necessary. There could be a memory leak in the system or an application. The `ps` command can be used for a brief monitoring of memory usage of processes. The `svmon` command can be used to gather more-detailed memory usage information about the processes suspected to leak memory.

PgspIn This is the number of paging space page ins per second. These are previously stolen pages read back from disk into real memory.

PgspOut This is the number of paging space page outs per second. If a page is selected to be stolen and the data in this page is changed, then the page must be written to paging space. (An unchanged page does not need to be written back.)

% Used The amount of used paging space. A good balanced system should not page; at least the page outs should be 0 (zero). Because of memory fragmentation, the amount of paging space used will increase on a newly started system over time. (It should be notable for the first few days.) However, if the amount of paging space used increases constantly, a memory leak may be the cause, and further investigations using `ps` and `svmon` are necessary. The load on the disks holding the paging space will increase if paging space ins (read from disk) and paging space outs (write to disk) increase.

2.3.3 Disk I/O bound system

The output of `topas` in Example 2-3 shows the fields that are used to decide whether the system is disk I/O bound.

Example 2-3 topas output with highlighted disk I/O statistics

```
Topas Monitor for host:  wlmhost          EVENTS/QUEUES  FILE/TTY
Fri May 11 11:28:06 2001  Interval:  2    Cswitch       64  Readch       353
                               Syscall       211 Writech      7836
Kernel   0.6  |                               | Reads        16  Rawin        0
User     99.3 |#####|                               | Writes        6  Ttyout       0
```

Wait	0.0						Forks	0	Igets	0
Idle	0.0						Execs	0	Namei	8
							Runqueue	4.0	Dirblk	0
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out		Waitqueue	0.0		
tr0	8.3	6.1	9.2	0.3	8.0					
lo0	0.0	0.0	0.0	0.0	0.0					
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ		PAGING		MEMORY	
hdisk0	0.0	2.0	0.0	0.0	2.0		Faults	0	Real,MB	511
hdisk1	0.0	0.0	0.0	0.0	0.0		Steals	0	% Comp	46.5
							PgspIn	0	% Noncomp	53.6
							PgspOut	0	% Client	49.6
							PageIn	0		
WLM-Class (Active)		CPU%	Mem%	Disk-I/O%			PageOut	0	PAGING SPACE	
Unmanaged		0	23	0			Sios	0	Size,MB	1024
Unclassified		0	0	0					% Used	13.1
							NFS (calls/sec)		% Free	86.8
Name	PID	CPU%	PgSp	Class			ServerV2	0		
dc	43564	25.0	0.3	System			ClientV2	0	Press:	
dc	21566	25.0	0.3	System			ServerV3	0	"h" for help	
dc	41554	25.0	0.3	VPs			ClientV3	0	"q" to quit	
dc	23658	24.2	0.3	System						

These are the fields of interest:

Wait The CPU idle time during which the system had at least one outstanding I/O to disk (whether local or remote) and asynchronous I/O is not in use. An I/O causes the process to block (or sleep) until the I/O is complete.

Disk The name of the physical device.

Busy% The percentage of time that the disk drive was active. A high busy percentage could be caused by random disk access. The disk's throughput may be low even if the percentage busy value is high. If this number is high for one or multiple devices, the **iostat** command can be used to gather more precise information. In case of paging activity the disk holding the paging logical volumes are used more than normal and the cause for the higher paging activity should be investigated. The **filemon** command can be used to gather information about the logical volume accessed to keep the disks busy and the process accessing the logical volume. The **fileplace** command can be used to gather information about the accessed files. All of this information can be used to redesign the layout of the logical volume and the file system. The **trace** command can be used to gather information about the application's access pattern to the data on disk, which may be useful in case a redesign of the application is possible.

KBPS	The total throughput of the disk in kilobytes per second. This value is the sum of KB-Read and KB-Writ. If this value is high, the iostat , filemon , and fileplace commands can be used to gather detailed data. A redesign of the logical volume or volume group may be necessary to improve I/O throughput.
TPS	The number of transfers per second or I/O requests to a disk drive.
KB-Read	The number of kilobytes read per second. Refer to the field KBPS. The system's total number of read system calls per second is shown in the Reads field. The system's total number of read characters per second is shown in the Readch field. Both Reads and Readch can be used to estimate the data block size transferred per read.
KB-Writ	The number of kilobytes written per second. Refer to the field KBPS. The system total number of write system calls per second is shown in the Writes field. The system total number of written characters per second is shown in the Writch field. Both Writes and Writch can be used to estimate the data block size transferred per write.

2.3.4 Network I/O bound system

The following output of **topas** in Example 2-4 shows the fields that are used to decide whether the system is network I/O bound.

Example 2-4 topas output with highlighted network I/O and nfs statistics

Topas Monitor for host: lpar05						EVENTS/QUEUES		FILE/TTY			
Thu May 15 17:35:41 2003						Interval: 2		Cswitch	2867	Readch	11.8M
						Syscall		2000	Writch	11.8M	
Kernel	2.7	#				Reads	757	Rawin	0		
User	4.2	#				Writes	758	Ttyout	0		
Wait	0.0					Forks	0	Igets	0		
Idle	93.0	#####				Execs	0	Namei	0		
						Runqueue		0.0	Dirblk	0	
						Waitqueue		0.0			
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	PAGING		MEMORY			
en0	6808.5	6027	9026	270.0	13347.0	Faults	0	Real,MB	2047		
lo0	0.0	0	0	0.0	0.0	Steals	0	% Comp	15.2		
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	PgspIn	0	% Noncomp	1.8		
hdisk0	0.0	0.0	0	0.0	0.0	PgspOut	0	% Client	2.3		
hdisk1	0.0	0.0	0	0.0	0.0	PageIn	0				
Name	PID	CPU%	PgSp	Owner		PageOut	0	PAGING SPACE			
ftp	315570	4.9	0.6	root		Sios	0	Size,MB	512		
xmgc	90156	0.1	0.0	root				% Used	1.5		
dd	307250	0.0	0.1	root		NFS (calls/sec)		% Free	98.4		
dd	274486	0.0	0.1	root		ServerV2	0				

syncd	127062	0.0	0.6	root	ClientV2	0	Press:
rmcd	225418	0.0	2.2	root	ServerV3	0	"h" for help
telnetd	278752	0.0	0.6	root	ClientV3	0	"q" to quit
IBM.Servi	319652	0.0	1.1	root			

These are the fields of interest for network performance:

Network	Shows the network interface.
KBPS	Transferred amount of data over the interface in KB per second. This is the sum of KB-In and KB-Out. If this is lower than expected, further investigation is necessary. Network-related resource bottlenecks such as CPU, disk I/O, or memory could be the cause. Tools and procedures to put maximum load on the network and reach the maximum possible transfer rates should be in place. The ftp put command shown in 1.5, "Network performance" on page 31 can be used. The netstat command as well as the interface statistics commands atmstat , entstat , estat , fddistat , and tokstat can be used to monitor network resources on the local system. The netpmon command provides detailed usage statistics for all network-related functions of the system. However, a monitoring of the remote systems as well as the network may be necessary to detect possible throughput limiting problems there.
I-Pack	Received packets per second. With the value of received bytes per second (KB-In), the average packet size can be calculated.
O-Pack	Sent packets per second. With the value of sent bytes per second (KB-Out) the average packet size can be calculated.
KB-In	Amount of data received on the interface per second.
KB-In	Amount of data sent on the interface per second.

Note: Detecting the root cause of a low network throughput is not easy. A shortage of resources on the local system can be the cause, such as an mbuf low condition (**netstat -m**), a busy CPU that prevents the execution of network code at the necessary speed, or slow disk I/O unable to deliver the necessary data fast enough. Test tools and procedures that use only a small amount of local resources to produce a high network load can help to detect problems on the network or the remote systems.

For NFS performance, **topas** shows only the number of NFS server and client calls for both NFS V2 and NFS V3. This data can only provide a quick overview of the NFS usage. The **nfsstat** command should be used to get more details about the NFS operations used and to gather RPC statistics.

Multi-resource monitoring and tuning tools

This part describes tools for monitoring and tuning multiple system resources. The commands listed are not specific to CPU, disk, memory, or network resources. They may be used across one or more of those resources. Some of the commands may report on CPU, the Virtual Memory Manager (VMM), and disk I/O, while others may report statistics on CPU and network activities. Refer to the sections referenced below for specific information about the individual tools.

- ▶ Monitoring tools:
 - The **iostat** command described in Chapter 4, “The iostat command” on page 81 is used to monitor system input/output device loading by

observing the time the physical disks are active in relation to their average transfer rates. It also reports on CPU use.

- The **netpmon** command described in Chapter 5, “The netpmon command” on page 93 is used to monitor a trace of system events on network activity and performance and the CPU consumption of network activities.
- The **PDT** tool described in Chapter 6, “Performance Diagnostic Tool (PDT)” on page 105 attempts to identify performance problems automatically by collecting and integrating a wide range of performance, configuration, and availability data.
- The **perfpnr** command described in Chapter 7, “The perfpnr command” on page 115 is a set of utilities that builds a test case by running many of the commands featured in this book. The test case contains the necessary information to assist in analyzing performance issues.
- The **ps** command described in Chapter 8, “The ps command” on page 127 is used to produce a list of processes on the system with specific information about, for instance, the CPU use of these processes.
- The **sar** command described in Chapter 9, “The sar command” on page 139 is used to report on CPU use, I/O, and other system activities.
- The **topas** command described in Chapter 11, “The topas command” on page 179 is used to monitor a broad spectrum of system resources such as CPU use, CPU events and queues, memory and paging use, disk performance, network performance, and NFS statistics. It also reports system resource consumption by processes assigned to different Workload Manager (WLM) classes.
- The **truss** command described in Chapter 11, “The topas command” on page 179 is used to track a process’s system calls, received signals, and incurred machine faults.
- The **vmstat** command described in Chapter 13, “The vmstat command” on page 211 is used to report statistics about kernel threads, virtual memory, disks, and CPU activity.
- ▶ Tuning tools:
 - The **fdpr** command described in Chapter 3, “The fdpr command” on page 71 is used for improving execution time and real memory use of user-level application programs and libraries.
 - The **schedo** command described in Chapter 10, “The schedo and schedtune commands” on page 165 is used to set criteria of thrashing, process suspension, time slices, and the length of time that threads can spin on locks. A sample compatibility script called **schedtune** exists.
 - The **vmo** and **ioo** command described in Chapter 14, “The vmo, ioo, and vmtune commands” on page 229 is used to change the characteristics of

the Virtual Memory Manager (VMM) such as page replacement, persistent file reads and writes, file system buffer structures (bufstructs), Logical Volume Manager (LVM) buffers, raw input/output, paging space, parameters, page deletes, and memory pinned parameters. A sample compatibility script called **vmtune** exists.

Archived

The fdpr command

The **fdpr** (Feedback Directed Program Restructuring) command is a performance tuning utility for improving execution time and real memory use of user-level application programs and libraries. The **fdpr** command can perform different actions to achieve these goals, such as removing unnecessary instructions and reordering of code and data. The **fdpr** program optimizes the executable image of a program by collecting information about the behavior of the program while the program is used for some typical workload, and then creates a new version of the program that is optimized for that workload.

fdpr resides in `/usr/bin` and is part of the `perfagent.tools` fileset, which is installable from the AIX base installation media.

3.1 fdpr

The **fdpr** command builds an optimized executable program in three distinct phases:

- ▶ Phase 1: Create an instrumented executable program.
- ▶ Phase 2: Run the instrumented program and create the profile data.
- ▶ Phase 3: Generate the optimized executable program file.

If not specified, all three phases are run. This is equal to the `-123` flags.

Depending on the phase to be executed by the **fdpr** command, the syntax of the **fdpr** command can be as follows:

- ▶ Most common use:

```
fdpr -p ProgramFile -x Command
```

- ▶ Syntax to use with phase 1 and 3 flags:

```
fdpr -p ProgramFile [ -M Segnum ] [ -fd Fdesc ] [ -o OutputFile ]  
[ -armemberArchiveMemberList ] [ OptimizationFlags ] [ -map ] [ -disasm ]  
[ -profcount ] [-v ] [ -s [ -1 | -3 ] ] [ -x WorkloadCommand ]
```

- ▶ Syntax to use with phase 2 flag:

```
fdpr -p ProgramFile [ -M Segnum ] [ -fd Fdesc ] [ -o OutputFile ]  
[ -armemberArchiveMemberList ] [ OptimizationFlags ] [ -map ] [ -disasm ]  
[ -profcount ] [-v ] [ -s [ -2 | -12 | -23 ] ] -x WorkloadCommand
```

The following is the syntax for the optimization flags:

```
[ [ -Rn ] | [ -R0 | -R1 | -R2 | -R3 ] ] [ -nI ] [ -tb ] [ -pc ] [ -pp ] [ -bt ]  
[ -toc ] [ -02 ] [ -03 ] [ -nop ] [ -opt_fdpr_glue ] [ -inline ] [ -i_resched ]  
[ -killed_regs ] [ -RD ] [ -full_saved_regs_calls ] [ -trunc_tb ] [ -tocload  
| -aggressive_tocload ] [ -regs_release ] [ -ret_prologs ] [ -volatile_regs ]  
[ -propagate ] [ -regs_redo ] [ -ptrgl_opt ] [ -dcbt_opt ]
```

Optimization flags

-1, -2, -3

Specifies the phase to run. The default is to run all three phases (-123). The `-s` flag must be used when running separate phases so that the succeeding phases can access the required intermediate files. The phases must be run in order (for example, -1, then -2, then -3, or -1, then -23). The `-2` flag must be used along with the invocation flag `-x`.

-M SegNum

Specifies where to map shared memory for profiling. The default is `0x30000000`. Specify an alternate shared memory address if the program to be reordered or any of the command strings invoked with the `-x` flag use

conflicting shared memory addresses. Typical alternative values are 0x40000000, 0x50000000, and so on up to 0xC0000000.

- fd Fdesc** Specifies which file descriptor number is to be used for the profile file that is mapped to the above shared memory area. The default of Fdesc is set to 1999.
- o OutFile** Specifies the name of the output file from the optimizer. The default is ProgramFile.f DPR
- p ProgramFile** Contains the name of the executable program file, shared object file, or shared library containing shared objects or executables to optimize. This program must be an unstriped executable.
- x Command** Specifies the command used for invoking the instrumented program. All of the arguments after the -x flag are used for the invocation. The -x flag is required when the -s flag is used with the -2 flag.
- armember amList** Lists archive members to be optimized within a shared archive file specified by the -p flag. If -armember is not specified, all members of the archive file are optimized. The entries in amList should be separated by spaces.
- profcoun t** Prints the profiling counters into a suffixed .counters file.
- disasm** Prints the disassembled version of the input program into a suffixed .dis file.
- map** Prints a map of basic blocks with their respective old and new addresses into a suffixed .map file.
- s** Specifies that temporary files created by the **fdpr** command cannot be removed. This flag must be used when running **fdpr** in separate phases.
- v** Enables verbose output.
- Rn** Copies input to output instead of invoking the optimizer. The -Rn flag cannot be used with the -R0, -R1, -R2, or -R3 flags.
- R0,-R1,-R2, -R3** Specifies the level of optimization. -R3 is the most aggressive optimization. The default is -R0. Refer to *AIX 5L Version 5.2 Commands Reference*, SBOF-1877, for more information about the optimization levels.
- nl** Does not permit branch reversing.
- tb** Forces the restructuring of traceback tables in reordered code. If -tb is omitted, traceback tables are automatically

	included only for C++ applications using a try and catch mechanism.
-pc	Preserves CSECT boundaries. Effective only with -R1 and -R3.
-pp	Preserves procedures boundaries. Effective only with -R1 and -R3.
-toc	Enable TOC pointer modifications. Effective only with -R0 and -R2.
-bt	Enables branch table modifications. Effective only with -R0 and -R2.
-O3	Switches on the following optimization flags: -nop, -opt_fdpr_glue, -inline, -i_resched, -killed_regs, -RD, -aggressive_tocload, -regs_release, -ret_prologs.
-inline	Performs inlining of hot functions.
-nop	Removes NOP instructions from reordered code.
-opt_fdpr_glue	Optimizes hot BBs in FDPR glue during code reordering.
-killed_regs	Avoids storing instructions for registers within callee functions' prologs that are later killed by the calling function.
-regs_release	Eliminates store/restore instructions in the function's prolog/epilog for non-frequently used registers within the function.
-tocload	Replaces an indirect load instruction via the TOC with an add immediate instruction.
-aggressive_tocload	Performs the -tocload optimization, and reduces the TOC size by removing redundant TOC entries.
-RD	Performs static data reordering in the .data and .bss sections.
-i_resched	Performs instruction rescheduling after code reordering.
-ret_prologs	Optimizes functions prologs that terminate with a conditional branch instruction directly to the function's epilog.

Attention: The **fdpr** command applies advanced optimization techniques that may result in programs that do not behave as expected. Programs that are reordered using this tool should be used with due caution and should be rigorously retested with, at a minimum, the same test suite used to test the original program in order to verify expected functionality. The reordered program is not supported by IBM.

3.1.1 Information about measurement and sampling

The **fdpr** command builds an optimized executable by applying advanced optimization techniques using three distinct phases to optimize the source executable. These three phases are:

- ▶ In phase one, **fdpr** creates an instrumented executable program. The source executable is saved as `__ProgramFile.save`, and a new and instrumented version, named `__ProgramFile.instr`, is built.
- ▶ In Phase two, **fdpr** runs the instrumented version of the executable, and profiling data is collected. This profiling data is stored in the file named `__ProgramFile.prof`. The executable needs to be run with typical input data to reflect normal use and to enable **fdpr** to find the code parts to improve.
- ▶ In Phase three, **fdpr** uses the profiled information collected in phase two to reorder the executable. This reordering includes tasks such as:
 - Packing together highly executed code sequences
 - Recoding conditional branches to improve hardware branch prediction
 - Moving less-used code sections out of line
 - Inlining of hot functions
 - Removing NOP instructions from reordered code

The compiler flag `-qfdpr` can be used to have the compiler add additional information into the executable that assists **fdpr** in reordering the executable. However, if the `-qfdpr` compiler flag is used, only those object modules compiled with this flag are reordered by **fdpr**. The reordered executable generated by **fdpr** provides a certain degree of debugging capability. Refer to *AIX 5L Version 5.2 Commands Reference* for more information about the **fdpr** command.

3.2 Examples for fdpr

Example 3-1 shows a source code of the C program that will be optimized using **fdpr**.

Example 3-1 C program used to show code instrumentation by fdpr

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>

main(argc, argv, envp)
    int argc;
    char **argv;
    char **envp;
{
    int x;
    x=atoi(argv[1]);
    if (x) {
        printf ("then part\n");
    } else {
        fprintf (stderr, "else part\n");
    } /* endif */
    exit (0);
}
```

This program converts the parameter passed to it into an integer and, depending on the value, the then or else part of the if instruction is executed. For easy identification of the then and else part in the assembler code, a printf in the then part and an fprintf in the else part is used. The code is compiled using:

```
cc -qfdpr -qlist c.c
```

The shell script in Example 3-2 is used to instrument the program.

Example 3-2 Shell script c.sh used to instrument the program with fdpr

```
#!/usr/bin/ksh
let x=0
while [ $x -lt 5000 ]
do
    ./a.out $x 2>/dev/null 1>/dev/null
    let x=x+1
done
```

The program a.out is called and the loop counter \$x is passed as the parameter. This way the else part of the example program gets executed only once and the then part gets executed 4999 times.

Example 3-3 shows how the **fdpr** command is used to optimize the program. The output indicates that the code is being reordered.

Example 3-3 Running fdpr

```
$ fdpr -p a.out -R3 -disasm -x ./c.sh
FDPR 5.2.0: The fdpr tool has the potential to alter the expected
behavior of a program. The resulting program will not be supported by IBM.
The user should refer to the fdpr document
for additional information.
Reading Input Executable File...
Recognizing CSECTs in Executable File...
Identifying the Basic Blocks...
## 54 Basic Blocks identified. ##
Instrumenting Input Executable File...
trampoline size 116
instrumented 100% of code
trampoline code and 1 trampolines were built, max 116 entries in 000001D0
10000150 <-- code[0] <-- 100005EC: size = 000004A0
100005F0 <-- trmp[0] <-- 100007BC: size = 000001D0, used 75% ( 88 in 116)
100007C0 <-- code[1] <-- 100012EC: size = 00000B2C
Writing output file /home/res1/fdpr_D/___a.out.instr...
Recognizing CSECTs in Executable File...
Identifying the Basic Blocks...
## 54 Basic Blocks identified. ##
Reading Profiling Information...
Maximal profiling counter 5000
Average profiling counter 1574.074219
2 NOP instructions found
Printing disassembly code into file ___a.out.save.dis...
Reordering the Code...
999)conditional jumps removed due to new code order (total executions removed..
The Code Reordering completed...
Writing output file /home/res1/fdpr_D/a.out.fdpr...
```

The result of the disassembled instruction in Example 3-4 shows that **fdpr** captures branch information for optimization.

Example 3-4 Content of ___a.out.save.dis

```
.__start {PR} (0x10000150):      # New BB: .....
.....more lines ...
0x10000380: 0x48000079: b1      0x100003f8      /* .atoi */
0x10000384:      # New BB: 0x10000384(size 16 proc 2 exec 5000)
      0x10000384: 0x80410014: l      r2,20(r1)
      0x10000388: 0x2c030000: cmpi   cr0,r3,0
      0x1000038c: 0x90610040: st     r3,64(r1)
      0x10000390: 0x41820014: beq    cr0,0x100003a4 # 12,bit2
0x10000394:      # New BB: 0x10000394(size 8 proc 2 exec 4999)
      0x10000394: 0x387f0010: cal   r3,16(r31)
```

```

0x10000398: 0x48000089: b1      0x10000420      /* .printf */
0x1000039c:      # New BB: 0x1000039c(size 8 proc 2 exec 4999)
0x1000039c: 0x80410014: l      r2,20(r1)
0x100003a0: 0x48000018: b      0x100003b8
L11:      # New BB: 0x100003a4(size 16 proc 2 exec 1)
0x100003a4: 0x80620050: l      r3,80(r2)      /* _iob */
0x100003a8: 0x389f001c: cal     r4,28(r31)
0x100003ac: 0x38630040: cal     r3,64(r3)
0x100003b0: 0x48000099: b1      0x10000448      /* .fprintf */
0x100003b4:      # New BB: 0x100003b4(size 4 proc 2 exec 1)
0x100003b4: 0x80410014: l      r2,20(r1)
L12:      # New BB: 0x100003b8(size 8 proc 2 exec 5000)
0x100003b8: 0x38600000: lil     r3,0x0
0x100003bc: 0x480000b5: b1      0x10000470      /* .exit */
.....more lines ....

```

The alternate shell script in Example 3-5 is now used to instrument the program.

Example 3-5 Alternate shell script c.sh2 to instrument the program

```

#!/usr/bin/ksh
let x=1
while [ $x -lt 5000 ]
do
    ./a.out 0 2>/dev/null 1>/dev/null
    let x=x+1
done
./a.out 1

```

This shell script runs the **./a.out 0** command 4999 times and **./a.out 1** only once. Using this shell script with the **fdpr** command to instrument the small C program shown on Example 3-1 on page 76. We use the following command:

```
$fdpr -p a.out -R3 -disasm -x ./c.sh2
```

Now the output **__a.out.save.dis** is shown in Example 3-6. It shows that **fdpr** captures different branch information for optimizing and restructuring the code.

Example 3-6 Content of __a.out.save.dis with c.sh2

```

__start {PR} (0x10000150):      # New BB.....
.... more lines .....
0x10000388: 0x2c030000: cmpi     cr0,r3,0
0x1000038c: 0x90610040: st      r3,64(r1)
0x10000390: 0x41820014: beq     cr0,0x100003a4 # 12,bit2
0x10000394:      # New BB: 0x10000394(size 8 proc 2 exec 1)
0x10000394: 0x387f0010: cal     r3,16(r31)
0x10000398: 0x48000089: b1      0x10000420      /* .printf */
0x1000039c:      # New BB: 0x1000039c(size 8 proc 2 exec 1)
0x1000039c: 0x80410014: l      r2,20(r1)

```

```
0x100003a0: 0x48000018: b      0x100003b8
L11: # New BB: 0x100003a4(size 16 proc 2 exec 4999)
0x100003a4: 0x80620050: l      r3,80(r2)      /* _iob */
0x100003a8: 0x389f001c: cal     r4,28(r31)
0x100003ac: 0x38630040: cal     r3,64(r3)
0x100003b0: 0x48000099: bl      0x10000448    /* .fprintf */
0x100003b4: # New BB: 0x100003b4(size 4 proc 2 exec 4999)
0x100003b4: 0x80410014: l      r2,20(r1)
L12: # New BB: 0x100003b8(size 8 proc 2 exec 5000)
..... more line.....
```

Keep in mind that the performance gain from **fdpr** depends on the way the program is run during instrumentation. The degree of performance improvement from the **fdpr**-optimized executable depends largely on how closely the production workload is imitated by the instrumented program.

Archived

The iostat command

The **iostat** command is used for monitoring system input/output device load by observing the time the physical disks are active in relation to their average transfer rates. The **iostat** command generates reports that can be used to determine an imbalanced system configuration to better balance the I/O load between physical disks and adapters.

The primary purpose of the **iostat** tool is to detect I/O bottlenecks by monitoring the disk utilization (% `tm_act` field). **iostat** can also be used to identify CPU problems, assist in capacity planning, and provide insight into solving I/O problems. Armed with both **vmstat** and **iostat**, you can capture the data required to identify performance problems related to CPU, memory, and I/O subsystems.

iostat resides in `/usr/bin` and is part of the `bos.acct` fileset, which is installable from the AIX base installation media.

4.1 iostat

The syntax of the **iostat** command is:

```
iostat [-s] [-a] [-d|-t] [-T] [-m] [PhysicalVolume ...] [Interval [Count ]]
```

Flags

-a	specifies adapter throughput report
-s	specifies system throughput report
-t	specifies tty/cpu report only
-T	specifies time stamp
-d	displays only the disk utilization report
-m	reports Path statistics by device and for all paths

The following conditions exist:

- ▶ The **-t** and **-d** are mutually exclusive; they cannot both be specified.
- ▶ The **-s** and **-a** flags can both be specified to display both the system and adapter throughput reports.
- ▶ If the **-a** flag is specified with the **-t** flag, the tty and CPU report is displayed followed by the adapter throughput report. Disk utilization reports of the disks connected to the adapters will not be displayed after the adapter throughput report.
- ▶ If the **-a** flag is specified with the **-d** flag, the tty and CPU report will not be displayed. If the **PhysicalVolume** parameter is specified, the disk utilization report of the specified Physical volume will be printed under the corresponding adapter to which it belongs.

Parameters

Interval

Specifies the update period — the amount of time between each report — in seconds. The first report contains statistics for the time since system startup (boot). Each subsequent report contains statistics collected during the interval since the previous report.

Count

Specifies the number of iterations. This can be specified in conjunction with the **Interval** parameter. If **Count** is specified, the value of **Count** determines the number of reports generated at **Interval** seconds apart. If the **Interval** is specified without the **Count** parameter, the command generates reports continuously.

PhysicalVolume

Specifies disks or paths. This can specify one or more alphabetic or alphanumeric physical volumes. If the **PhysicalVolume** parameter is specified, the tty and CPU

reports are displayed and the disk report contains statistics for the specified drives. If a specified logical drive name is not found, the report lists the specified name and displays the message `Disk is not Found`.

If no logical drive names are specified, the report contains statistics for all configured disks and CD-ROMs. If no drives are configured on the system, no disk report is generated. The first character in the `PhysicalVolume` parameter cannot be numeric.

4.1.1 Information about measurement and sampling

The `iostat` command generates four types of reports:

- ▶ tty and CPU utilization
- ▶ Disk utilization
- ▶ System throughput
- ▶ Adapter throughput

Note: The first set of `iostat` output contains the cumulative data from the last boot to the start of the `iostat` command.

Each subsequent sample in the report covers the time since the previous sample. All statistics are reported each time the `iostat` command is run. The report consists of a tty and CPU header row followed by a row of tty and CPU statistics. CPU statistics are calculated systemwide as averages among all processors.

The `iostat` command keeps a history of disk input/output activity shown in “Enabling disk input/output statistics” on page 90. Information about the disks and which disks are attached to which adapters are stored in the Object Database Manager (ODM).

Measurement is done as specified by the parameters in the command line issued by the user.

4.2 Examples for iostat

The following sections show reports generated by `iostat`.

4.2.1 System throughput report

This system throughput report is generated if the `-s` flag is specified and provides statistics for the entire system. It has the format shown in Example 4-1. The fields `Kbps`, `tps`, `Kb_read`, and `Kb_wrtn` are accumulated totals for the entire system.

Example 4-1 System throughput report

```
# iostat -s
tty:      tin          tout  avg-cpu:  % user   % sys    % idle   % iowait
          0.0          0.0           33.0    11.8     10.1     45.1

System: lpar05

          Kbps      tps    Kb_read  Kb_wrtn
          2774.1    367.1    18156    9592

Disks:    % tm_act   Kbps    tps    Kb_read  Kb_wrtn
hdisk0    92.4    451.9    100.9    1000    3520
hdisk1    88.2    447.9   100.2    964   3516
hdisk3    76.7    1090.7    94.7    9632    1278
hdisk5     0.0      0.0      0.0      0      0
hdisk6     0.0      0.0      0.0      0      0
hdisk8     0.0      0.0      0.0      0      0
hdisk9     0.0      0.0      0.0      0      0
hdisk2    74.1    783.6    71.3    6560    1278
hdisk10    0.0      0.0      0.0      0      0
hdisk7     0.0      0.0      0.0      0      0
cd0        0.0      0.0      0.0      0      0
```

The following values are displayed:

► **Statistic for tty**

`tin` Shows the total number of characters read by the system for all ttys.

`tout` Shows the total number of characters written by the system to all ttys.

You will see few input characters and many output characters. On the other hand, applications such as `vi` result in a smaller difference between the number of input and output characters. Analysts using modems for asynchronous file transfer may notice the number of input characters exceeding the number of output characters. Naturally, this depends on whether the files are being sent or received relative to the measured system.

Because the processing of input and output characters consumes CPU resources, look for a correlation between increased TTY activity and CPU utilization. If such a relationship exists, evaluate ways to improve the performance of the TTY subsystem. Steps that could be taken include changing the application program, modifying TTY port parameters during file transfer, or perhaps upgrading to a faster or more efficient asynchronous communications adapter.

► Average CPU usage

% user

Shows the percentage of CPU resources spent in user mode. A UNIX process can execute in user or system mode. When in user mode, a process executes within its own code and does not require kernel resources. On an SMP system, the % user is averaged across all CPUs.

% sys

Shows the percentage of CPU utilization that occurred while executing at the system level (kernel). On an SMP system, the % sys is averaged across all CPUs. This includes CPU resources consumed by kernel processes (kprocs) and others that need access to kernel resources. For example, the reading or writing of a file requires kernel resources to open the file, seek a specific location, and read or write data. A UNIX process accesses kernel resources by issuing system calls. A high number of system calls in relation to user utilization can be caused by applications inefficiently performing disk I/O or misbehaving shell scripts such as a shell script stuck in a loop, which can generate a large number of system calls. If you encounter this, look for penalized processes. Run the `ps -eaf` command and look under the **C** column for processes that are penalized. Refer to 8.2.3, “Displaying the processes in order of being penalized” on page 133 for more information.

Typically, the CPU is pacing (the system is CPU bound) if the sum of user and system time exceeds 90 percent of CPU resources on a single-user system or 80 percent on a multi-user system. This condition could mean that the CPU is the limiting factor in system performance.

A factor when evaluating CPU performance is the size of the run queue (provided by the `vmstat` command, see 13.2.1, “Virtual memory activity” on page 213). In general, as the run queue increases, users will notice degradation (an increase) in response time.

<code>% idle</code>	Shows the percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request. The <code>% idle</code> column shows the percentage of CPU time spent idle, or waiting, without pending local disk I/O. If there are no processes on the run queue, the system dispatches a special kernel process called <code>wait</code> . On an SMP system, the <code>% idle</code> is averaged across all CPUs.
<code>% iowait</code>	Shows the percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request. On an SMP system, the <code>% iowait</code> is averaged across all CPUs.

The `iowait` state is different from the `idle` state in that at least one process is waiting for local disk I/O requests to complete. Unless the process is using asynchronous I/O, an I/O request to disk causes the calling process to block (or sleep) until the request is completed. Once a process's I/O request completes, it is placed on the run queue. On systems running a primary application, a high I/O wait (`iowait`) percentage may be related to workload. In this case, there may be no way to overcome the problem.

When you see a high `iowait` percentage, you need to investigate the I/O subsystem to try to eliminate any potential bottlenecks. It could be insufficient memory, in which case the disk(s) containing paging space may be busy while paging and you are likely to see a higher run queue as threads are waiting for the CPU. An inefficient I/O subsystem configuration, or an application handling input/output inefficiently can also result in higher `%iowait`.

A `%iowait` percentage is not necessarily a bad thing. For example, if you are copying a file, you will want to see the disk as busy as possible. In this scenario, a higher `%tm_act` with good disk throughput would be desirable over a disk that is only 50 `%tm_act`.

If an application is writing sequential files, then the write behind algorithm will write pages to disk. With large sequential writes, the `%iowait` will be higher, but the busy disk does not block the application because the application has already written to memory. The application is free to continue processing and is not waiting on the disk. Similarly, when sequential reads are performed, the

%iowait can increase as the pages are read in, but this does not effect the application because only the pages that are already read into memory are made available to the application and read ahead is not dependant on the application.

Understanding the I/O bottleneck and improving the efficiency of the I/O subsystem requires more data than **iostat** can provide. However, typical solutions might include:

- Limiting the number of active logical volumes and file systems placed on a particular physical disk. The idea is to balance file I/O evenly across all physical disk drives.
 - Spreading a logical volume across multiple physical disks. This is useful when a number of different files are being accessed. Use the `lslv -m` command to see how volume groups are placed on physical disks.
 - Creating multiple Journaled File System (JFS) logs for a volume group and assigning them to specific file systems. (This is beneficial for applications that create, delete, or modify a large number of files, particularly temporary files.)
 - Backing up and restoring file systems to reduce fragmentation. Fragmentation causes the drive to seek excessively and can be a large portion of overall response time.
 - Adding additional drives and rebalancing the existing I/O subsystem.
- ▶ Disk activity status

% tm_act

Indicates the percentage of time the physical disk was active (bandwidth utilization for the drive). The % tm_act column shows the percentage of time the volume was active. This is the primary indicator of a bottleneck. Any % tm_act over 70 percent may be considered a potential bottleneck.

A drive is active during data transfer and command processing, such as seeking to a new location. The disk-use percentage is directly proportional to resource contention and inversely proportional to performance. As disk use increases, performance decreases and the time it takes for the system to respond to user requests increases. In general, when a disk's use (% tm_act) exceeds 70 percent, processes may be waiting longer than necessary for I/O to complete because most UNIX processes block (or sleep) while waiting for their I/O requests to complete.

Kbps	Indicates the amount of data transferred (read or written) to the drive in KB per second.
tps	Indicates the number of transfers per second that were issued to the physical disk. A transfer is an I/O request at the device driver level to the physical disk. As physical I/O (read or write, to or from the disk) is expensive in terms of performance, in order to reduce the amount of physical I/O to the disk(s), multiple logical requests (reads and writes from the application) can be combined into a single physical I/O. A transfer is of an indeterminate size.
Kb_read	The total number of KB read.
Kb_wrtn	The total number of KB written. Kb_read and Kb_wrtn combined should not exceed 70 percent of the disk or adapter's throughput to avoid saturation.

With the `-s` flag is specified, a system-header row is displayed followed by a line of statistics for the entire system. The hostname of the system is printed in the system-header row. It provides the statistics since boot time.

If you run `iostat` specifying an interval, for example `iostat -s 5` to display statistics every five seconds, or you run `iostat` specifying an interval and a count, for example `iostat -s 2 5` to display five reports of statistics every two seconds, then the first report will represent the I/O activity since boot time and the subsequent reports will reflect the amount of I/O on the system over the last interval.

What the report is telling us

The above report shows 45.1 percent `iowait`. This should be investigated further. By looking at `% tm_act`, we know we are having performance hits on `hdisk0`, `hdisk1`, `hdisk2`, and `hdisk3`. This is because `% tm_act` is more than 70 percent. We need to run `filemon`, refer to “Analyzing the physical volume reports” on page 464 to see why the disks are busy. For example, some files may have a lot of I/O, or disks may be seeking. The `vmstat` command (refer to 13.2.1, “Virtual memory activity” on page 213) may report high paging.

4.2.2 tty and CPU utilization report

The first report generated by the `iostat` command is the tty and CPU utilization report. The CPU values are global averages among all processors. The I/O wait state is defined systemwide and not per processor.

This information is updated at regular intervals by the kernel (typically 60 times per second). The `tty` report provides a collective account of characters per second received from all terminals on the system as well as the collective count of characters output per second to all terminals on the system. Example 4-2 shows the `tty` and CPU utilization report.

Example 4-2 tty and CPU utilization report

```
# iostat -t
tty:      tin          tout  avg-cpu: % user   % sys   % idle  % iowait
          1.5          9846.3      26.9    1.3    70.6    1.1
```

4.2.3 Disk utilization report

The disk utilization report, generated by the `iostat` command, provides statistics on a per physical disk basis. Statistics for CD-ROM devices are also reported.

A disk header column is displayed followed by a column of statistics for each disk that is configured. If the `PhysicalVolume` parameter is specified, only those names specified are displayed. Example 4-3 shows the disk utilization report.

Example 4-3 Disk utilization report

```
# iostat -d
Disks:      % tm_act   Kbps    tps    Kb_read  Kb_wrtn
hdisk0      92.4       451.9   100.9   1000     3520
hdisk1      88.2       447.9   100.2   964      3516
hdisk3      76.7      1090.7   94.7    9632     1278
hdisk5       0.0        0.0     0.0     0         0
hdisk6       0.0        0.0     0.0     0         0
hdisk8       0.0        0.0     0.0     0         0
hdisk9       0.0        0.0     0.0     0         0
hdisk2      74.1       783.6   71.3    6560     1278
hdisk10      0.0        0.0     0.0     0         0
hdisk7       0.0        0.0     0.0     0         0
cd0         0.0        0.0     0.0     0         0
```

If `iostat -d` is run as is, then the statistics since boot time are displayed.

If you run `iostat` specifying an interval, for example `iostat -d 5` to display statistics every five seconds, or you run `iostat` specifying an interval and a count, such as `iostat -d 2 5` to display five reports of statistics every two seconds, then the first report will represent the I/O activity since boot time and the subsequent reports will reflect the amount of I/O on the system over the last interval.

4.2.4 Disk utilization report for MPIO

For Enterprise Storage Server (ESS) machines, the vpaths will be treated as disks and hdisks will be treated as Paths. Internally, the vpaths are actually disks and hdisks are the paths to them. For multi-path input-output (MPIO) enabled devices, the path name will be represented as Path0, Path1, Path2 and so on. The numbers 0, 1, 2, and so on are the path IDs provided by the **lspath** command. Since paths to a device can be attached to any adapter, the adapter report will report the path statistics under each adapter. The disk name will be a prefix to all of the paths. For all MPIO-enabled devices, the adapter report will print the path names as hdisk10_Path0, hdisk0_Path1, and so on. For all ESS Machines, the adapter report will print the path names as vpath0_hdisk3, vpath10_hdisk25, and so on.

If you use **iostat -m**, you can see input/output statistics on MPIO as shown in Example 4-4. However, we do not have redundant path in our setup. Therefore only a single path is identified for both SCSI drives.

Example 4-4 Output of iostat -m

```
lpar05:/>> iostat -m
```

tty:	tin	tout	avg-cpu:	% user	% sys	% idle	% iowait
	0.2	12.4		8.8	3.7	57.8	29.8
Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn		
hdisk1	0.0	0.2	0.0	12631	13276		
Paths:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn		
Path0	0.0	0.3	0.0	25262	26552		
Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn		
hdisk0	43.4	1301.4	96.3	64838983	142530689		
Paths:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn		
Path0	43.4	2602.9	192.5	129677967	285061379		

Enabling disk input/output statistics

To improve performance, the collection of disk input/output statistics may have been disabled. For large system configurations where a large number of disks is configured, the system can be configured to avoid collecting physical disk input/output statistics when the **iostat** command is not executing. If the system is configured in this manner, then the first disk report displays the message `Disk History Since Boot Not Available` instead of the disk statistics. Subsequent interval reports generated by the **iostat** command contain disk statistics collected during the report interval. Any tty and CPU statistics after boot are unaffected if a system management command is used to re-enable disk

statistics-keeping. The first **iostat** command report displays activity from the interval starting at the point that disk input/output statistics were enabled.

To enable the collection of this data, enter:

```
chdev -l sys0 -a iostat=true
```

To display the current settings, enter:

```
lsattr -E -l sys0 -a iostat
```

If disk input/output statistics are enabled, the **lsattr** command will display:

```
iostat true Continuously maintain DISK I/O history True
```

If disk input/output statistics are disabled, the **lsattr** command will display:

```
iostat false Continuously maintain DISK I/O history True.
```

Note: Some system resources are consumed in maintaining disk I/O history for the **iostat** command.

4.2.5 Adapter throughput report

If the **-a** flag is specified, an adapter-header row is displayed followed by a line of statistics for the adapter. This will be followed by a disk-header row and the statistics of all of the disks and CD-ROMs connected to the adapter. The adapter throughput report shown in Example 4-5 is generated for all of the disk adapters connected to the system. Each adapter statistic reflects the performance of all of the disks attached to it.

Example 4-5 Adapter throughput report

```
# iostat -a
tty:      tin      tout  avg-cpu:  % user   % sys    % idle   % iowait
          1.8      7989.0      21.9    1.2     76.0     0.9

Adapter:      Kbps    tps    Kb_read  Kb_wrtn
scsi2         4.5     0.5    14429    920

Paths/Disks:  % tm_act  Kbps    tps    Kb_read  Kb_wrtn
hdisk0_Path0  0.0      4.5     0.5    14429    920
hdisk1_Path0  0.0      0.0     0.0     0        0

Adapter:      Kbps    tps    Kb_read  Kb_wrtn
scsi0         0.0     0.0     0        0

Paths/Disks:  % tm_act  Kbps    tps    Kb_read  Kb_wrtn
cd0           0.0      0.0     0.0     0        0
```

If **iostat -a** is run as is, then the statistics since boot time are displayed.

If you run **iostat** specifying an interval, for example **iostat -a 5** to display statistics every five seconds, or you run **iostat** specifying an interval and a count, for example **iostat -a 2 5** to display five reports of statistics every two seconds, then the first report represents the I/O activity since boot time and the subsequent reports reflect the amount of I/O on the system over the last interval.

Tip: It is useful to run **iostat** when your system is under load and performing normally. This gives a baseline to determine future performance problems with the disk, CPU, and tty subsystems.

You should run **iostat** again when:

- ▶ Your system is experiencing performance problems.
- ▶ You make hardware or software changes to the disk subsystem.
- ▶ You make changes to the AIX Operating System, such as installing, upgrades, and changing the disk tuning parameters using **ioo**.
- ▶ You make changes to your application.

The netpmon command

The **netpmon** command makes use of the trace utility to monitor network activity. Because of this, only root and members of the system group can run this command. The **netpmon** command reports on network activity over the monitoring period.

Note: The **netpmon** command does not work with NFS 3 and is only supported on POWER-based platforms.

The **netpmon** command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

5.1 netpmon

The syntax of the **netpmon** command is:

```
netpmon [ -o File ] [ -d ] [ -T n ] [ -P ] [ -t ] [ -v ] [ -O ReportType ... ]  
[-i Trace_File -n Gennames_File ]
```

Flags

- d** Starts the **netpmon** command, but defers tracing until the **trcon** command has been executed by the user. By default, tracing is started immediately.
- i Trace_File** Reads trace records from the trace file produced with the **trace** command instead of a live system. The trace file must be rewritten first in raw format using the **trcpt -r** command. This flag cannot be used without the **-n** flag.
- n Gennames_File** Reads necessary mapping information from the file **Gennames_File** produced by the **gennames** command. This flag is mandatory when the **-i** flag is used.
- o File** Writes the reports to the specified **File** instead of to standard output.
- O ReportType ...** Produces the specified report types. Valid report type values are:
- | | |
|-----|--------------------------------------|
| cpu | CPU use |
| dd | Network device-driver I/O |
| so | Internet socket call I/O |
| nfs | NFS I/O |
| all | All of the above (the default value) |
- P** Pins monitor process in memory. This flag causes the **netpmon** text and data pages to be pinned in memory for the duration of the monitoring period. This flag can be used to ensure that the real-time **netpmon** process does not run out of memory space when running in a memory-constrained environment.
- t** Prints CPU reports on a per-thread basis.
- T n** Sets the kernel's trace buffer size to **n** bytes. The default size is 64000 bytes. The buffer size can be increased to accommodate larger bursts of events, if any. (A typical event record size is on the order of 30 bytes.)

Note: The trace driver in the kernel uses double buffering, so actually two buffers of size n bytes will be allocated. These buffers are pinned in memory, so they are not subject to paging.

-v Prints extra information in the report. All processes and all accessed remote files are included in the report instead of only the 20 most active processes and files.

5.1.1 Information about measurement and sampling

Once **netpmon** is started, it runs in the background until it is stopped by issuing the **trcstop** command. The **netpmon** command reports on network-related activity over the monitoring period. If the default settings are used, the **trace** command is invoked automatically by the **netpmon** command. Alternately, **netpmon** has an option **-d** flag to switch the trace on at a later time using the **trcon** command. When the trace is stopped by issuing the **trcstop** command, the **netpmon** command outputs its report and exits. Reports are either displayed on standard output by default or can be redirected to a file with the **-f** flag.

The **netpmon** command monitors a trace of a specific number of trace hooks. The trace hooks include NFS, **dstokdd**, and **ethchandd**. When the **netpmon** command is issued with the **-v** flag, the trace hooks used by **netpmon** are listed. Alternatively, you can run the **trcevgrp -l netpmon** command to receive a list of trace hooks that are used by **netpmon**.

The **netpmon** command can also be used offline with the **-i** flag specifying the trace file and a **-n** flag to specify the **gennames** file. The **gennames** command is used to create this file. Refer to 36.2, “gennames” on page 704 for more information about **gennames**.

Reports are generated for the CPU use, the network device driver I/O, Internet socket calls, and Network File System (NFS) I/O information.

CPU use

The **netpmon** command reports on the CPU use by threads and interrupt handlers. The command differentiates between CPU use on network-related activity and other CPU use.

Network Device Driver I/O

The **netpmon** command monitors I/O statistics through network adapters.

Internet Socket Calls

The **netpmon** command monitors the read, recv, recvfrom, write, send, and sendto subroutines on the Internet socket. Per-process reports on the following protocols are created:

- ▶ Internet Control Message Protocol (ICMP)
- ▶ Transmission Control Protocol (TCP)
- ▶ User Datagram Protocol (UDP)

NFS I/O

The **netpmon** command monitors read and write subroutines on client NFS files, Remote Procedure Calls (RPC) requests on NFS clients, and NFS server read and write requests.

Note: Only one **trace** can be run on a system at a time. If an attempt is made to run a second **trace**, this error message will be displayed:

```
0454-072 The trace daemon is currently active. Only one trace session
may be active at a time.
```

If network-intensive applications are being monitored, the **netpmon** command may not be able to capture all of the data. This occurs when the trace buffers are full. The following message is displayed:

```
Trace kernel buffer overflowed ....
```

The size of the trace buffer can be increased by using the **-T** flag. Using the offline mode is the most reliable way to limit buffer overflows. This is because **trace** is much more efficient in processing and logging than the trace-based utilities **filemon**, **netpmon**, and **tprof**.

In memory-constrained environments, the **-P** flag can be used to pin the text and data pages of the **netpmon** process in memory so they cannot be swapped out.

5.2 Examples for netpmon

In the test scenario, a file of approximately 100 MB was transferred between two servers. The /home file system of the one server is remotely mounted to the other server via NFS. This scenario has been set up to obtain trace results for the copy operation between the servers. The command in Example 5-1 on page 97 was used to obtain the **netpmon** information.

Example 5-1 The netpmon command used to monitor NFS transfers

```
# netpmon -o nmon1.out -O nfs
```

Enter the "trcstop" command to complete netpmon processing

Once the **netpmon** command is running, start the network activity to be monitored. Once the network activity that is being monitored is completed, run the **trcstop** command to stop the trace, as shown in Example 5-2.

Example 5-2 Stopping netpmon

```
# trcstop
[netpmon: Reporting started]
[netpmon: Reporting completed]
[netpmon: 162.629 secs in measured interval]
```

The output that was generated by the **netpmon** command in Example 5-1 can be seen in Example 5-3. This output will only display nfs statistics, as the **-O** option was used with **nfs**. The RPC statistics as well as the total calls are displayed for the server **wlmhost**.

Example 5-3 The netpmon command output data for NFS

```
Fri May 25 19:08:12 2001
System: AIX server1 Node: 5 Machine: 000BC6FD4C00
```

```
=====
```

NFS Client RPC Statistics (by Server):

```
-----
```

Server	Calls/s
--------	---------

wlmhost	31.02
---------	-------

Total (all servers)	31.02
---------------------	-------

```
=====
```

Detailed NFS Client RPC Statistics (by Server):

```
-----
```

SERVER: wlmhost

calls: 16594

call times (msec): avg 108.450 min 1.090 max 2730.069 sdev 102.420

COMBINED (All Servers)

calls: 16594

call times (msec): avg 108.450 min 1.090 max 2730.069 sdev 102.420

Example 5-4 shows the **netpmon** command providing a full compliment of report types. When the **-O** flag is *not* issued, the default of all is assumed.

Example 5-4 The netpmon command providing a full listing on all report types

```
server1> netpmon -o nmon2.out -v
```

Enter the "trcstop" command to complete netpmon processing

```
/usr/sbin/trace -a -T 256000 -o - -j
000,000,001,002,003,005,006,106,10C,139,134,135,100,200,102,103,101,104,465,
467,46A,00A,163,19C,256,255,262,26A,26B,32D,32E,2A7,2A8,351,352,320,321,30A,
30B,330,331,334,335,2C3,2C4,2A4,2A5,2E6,2E7,2DA,2DB,2EA,2EB,252,216,211,107,
212,215,213
```

Moving this process to the background.

The following script generates network traffic.

```
# ftp wlmhost
```

```
Connected to wlmhost.
```

```
220 wlmhost FTP server (Version 4.1 Sun Apr 8 07:45:00 CDT 2001) ready.
```

```
Name (wlmhost:root): root
```

```
331 Password required for root.
```

```
Password:
```

```
230 User root logged in.
```

```
ftp> cd /home/nmon
```

```
250 CWD command successful.
```

```
ftp> mput big*
```

```
mput big.? y
```

```
200 PORT command successful.
```

```
150 Opening data connection for big..
```

```
226 Transfer complete.
```

```
107479040 bytes sent in 68.91 seconds (1523 Kbytes/s)
```

```
local: big. remote: big.
```

```
ftp>
```

```
# trcstop
```

```
# [netpmon: Reporting started]
```

```
[netpmon: Reporting completed]
```

```
[netpmon: 1545.477 secs in measured interval]
```

The full listing for the **netpmon** command is shown for the duration of the **ftp** transfer operation in Example 5-4. It has been broken up into sections for clarity. The sections are broken up into process statistics, First Level Interrupt Handler (FLIH) and Second Level Interrupt Handler (SLIH) statistics, network device-driver statistics, TCP socket call statistics, and detailed statistics.

5.2.1 Process statistics

Example 5-5 below shows the process statistics for the `netpmon` command's full report.

Example 5-5 The netpmon command verbose output showing process information

```
Sun May 27 11:46:52 2001
System: AIX server1 Node: 5 Machine: 000BC6FD4C00

trace -a -T 256000 -o - -j
000,000,001,002,003,005,006,106,10C,139,134,135,100,200,102,
103,101,104,465,467,46A,00A,163,19C,256,255,262,26A,26B,32D,32E,2A7,2A8,351,
352,320,321,30A,30B,330,331,334,335,2C3,2C4,2A4,2A5,2E6,2E7,2DA,2DB,2EA,2EB,
252,216,211,107,212,215,213
TIME: 0.000000000 TRACE ON pid 7254 tid 0x82a9
channel 990982013
TIME: 120.467389060 TRACE OFF
...(lines omitted)...
Process CPU use Statistics:
-----
```

Process	PID	CPU Time	CPU %	Network CPU %
ypbind	10580	17.9523	3.726	0.000
ftp	19060	12.6495	2.625	1.146
netpmon	17180	2.5410	0.527	0.000
UNKNOWN	16138	0.5125	0.106	0.000
syncd	6468	0.2858	0.059	0.000
dtgreet	4684	0.2294	0.048	0.000
UNKNOWN	18600	0.1940	0.040	0.000
UNKNOWN	5462	0.1929	0.040	0.000
wlmsched	2580	0.1565	0.032	0.000
gil	2322	0.1057	0.022	0.022
aixterm	16050	0.0915	0.019	0.005
swapper	0	0.0468	0.010	0.000
X	5244	0.0428	0.009	0.000
lrud	1548	0.0404	0.008	0.000
trcstop	19062	0.0129	0.003	0.000
init	1	0.0112	0.002	0.000
ksh	18068	0.0080	0.002	0.000
rpc.lockd	11872	0.0070	0.001	0.000
nfsd	10326	0.0064	0.001	0.001
netpmon	14922	0.0034	0.001	0.000
netm	2064	0.0032	0.001	0.001
rmcd	15744	0.0028	0.001	0.000
IBM.FSrmcd	14714	0.0027	0.001	0.000
snmpd	4444	0.0023	0.000	0.000
trace	19058	0.0019	0.000	0.000
xmhc	1806	0.0015	0.000	0.000

sendmail	6236	0.0010	0.000	0.000
cron	9822	0.0009	0.000	0.000
hostmibd	8514	0.0007	0.000	0.000
IBM.AuditRMd	16516	0.0007	0.000	0.000
IBM.ERrmd	5080	0.0006	0.000	0.000
syslogd	6974	0.0005	0.000	0.000
PM	13932	0.0004	0.000	0.000
UNKNOWN	7254	0.0004	0.000	0.000
UNKNOWN	5460	0.0003	0.000	0.000
UNKNOWN	5464	0.0003	0.000	0.000
rtcmd	9032	0.0001	0.000	0.000
shdaemon	15480	0.0001	0.000	0.000

Total (all processes)		35.1103	7.286	1.175
Idle time		459.0657	95.268	

This example shows the **trace** command that produced the output. The command was asynchronous, as can be seen by the use of the **-a** flag. The buffer size was increased to 256 KB with the **-T** flag and, more important, the output was redirected to the standard output by using the **-o** flag. The list of trace hooks follows the **-j** flag. For more information about the trace command flags, refer to the **trace** command in 40.1, “trace” on page 760

Under the heading **Process CPU use Statistics**, the following headings can be seen:

Process	The name of the process that is being monitored
PID	The process identification number
CPU Time	The total CPU time used
CPU %	The CPU time as a percentage of total time
Network CPU %	The percentage of CPU time spent on executing network-related tasks

In Example 5-5 on page 99, the **-v** flag was used, so more than 20 processes are displayed. At the bottom of the **Process CPU use Statistics** output, the **Total CPU** and **total Idle** time is displayed. It can be seen from the process statistics that the **ftp** transfer used 12.6 seconds of CPU time. The total CPU time as seen from the bottom of the process statistics table is 494 seconds. This equates to 2.6 percent of the CPU total time spent executing this command.

5.2.2 FLIH and SLIH CPU statistics

Example 5-6 shows a report of the FLIH and SLIH CPU use statistics. The report is an extract from the full `netpmon` report.

Example 5-6 The full netpmon report showing FLIH and SLIH statistics

First Level Interrupt Handler CPU use Statistics:

```
-----  
FLIH                                CPU Time  CPU %  Network  
                                CPU %  CPU %  
-----  
PPC decremter                      1.8355  0.381  0.000  
external device                    0.9127  0.189  0.185  
data page fault                    0.0942  0.020  0.000  
queued interrupt                   0.0286  0.006  0.000  
instruction page fault              0.0061  0.001  0.000  
-----  
Total (all FLIHs)                  2.8770  0.597  0.186  
-----
```

Second Level Interrupt Handler CPU use Statistics:

```
-----  
SLIH                                CPU Time  CPU %  Network  
                                CPU %  CPU %  
-----  
cstokdd                          2.7421  0.569  0.569  
s_scsiddpin                        0.0045  0.001  0.000  
gxentdd                             0.0026  0.001  0.001  
unix                                0.0001  0.000  0.000  
-----  
Total (all SLIHs)                  2.7494  0.571  0.570  
-----
```

Additional information about first-level and second-level interrupt handlers is shown in the report. The statistics that are displayed under these headings are:

FLIH	The description of the first-level interrupt handler
SLIH	The description of the second level interrupt handler
CPU Time	The total amount of time used by the interrupt handler
CPU %	The CPU time used by this interrupt handler as a percentage of total CPU time.
Network CPU %	The percentage of total time that this interrupt handler executed for a network-related process.

At the bottom of the first-level and second-level interrupt handler reports, the total amount of CPU use for the specific level of interrupt handler is displayed. Note that in the SLIH column, the statistics for `cstokdd` are displayed. This is the time that the CPU spent handling interrupts from the token-ring adapter (which may have had traffic other than the **ftp** transfer data). Hence these CPU use statistics *cannot* be regarded as the statistics for the **ftp** transfer.

5.2.3 TCP socket call statistics

Example 5-7 is an extract from the full verbose output of the `netpmon` command. The extract shows the TCP socket call statistics.

Example 5-7 Extract from the full netpmon report showing socket call statistics

TCP Socket Call Statistics (by Process):

Process	PID	Read		Write	
		Calls/s	Bytes/s	Calls/s	Bytes/s
ftp	19060	0.30	1202	13.51	892186
aixterm	16050	0.81	26	2.27	142
Total (all processes)		1.10	1227	15.78	892328

A socket report is also provided under the heading Detailed TCP Socket Call Statistics (by Process). The details for the **ftp** transfer are shown in the first line of this report. Use the process identification (PID) to identify the correct **ftp** transfer. Note that over the same monitoring period, there could be more than one **ftp** transfer running. The following fields are displayed in this report:

Process	This is the name of the process
PID	This is the process identification number
Read Calls/s	This is the number of read, recv, and recvfrom subroutines made per second by this process on sockets of this type
Read Bytes/s	The number of bytes per second requested by the read, recv, and recvfrom subroutine calls
Write Calls/s	The number of write, send, and sendto subroutine calls per second made by this process on this socket type
Write Bytes/s	The number of bytes per second written to this process to sockets of this protocol type

5.2.4 Detailed statistics

Example 5-8 shows the detailed **netpmon** statistics, which are an extract from the **netpmon** full report.

Example 5-8 Extract from the netpmon full report showing detailed statistics

Detailed Second Level Interrupt Handler CPU use Statistics:

```
-----  
SLIH: cstokdd  
count:          43184  
  cpu time (msec):  avg 0.063  min 0.008  max 0.603  sdev 0.028  
  
SLIH: s_scsiddpin  
count:          221  
  cpu time (msec):  avg 0.020  min 0.009  max 0.044  sdev 0.009  
  
SLIH: gxentdd  
count:          122  
  cpu time (msec):  avg 0.021  min 0.011  max 0.024  sdev 0.002  
  
SLIH: unix  
count:          12  
  cpu time (msec):  avg 0.010  min 0.003  max 0.013  sdev 0.003  
  
COMBINED (All SLIHs)  
count:          43539  
  cpu time (msec):  avg 0.063  min 0.003  max 0.603  sdev 0.028  
  
-----
```

Detailed Network Device-Driver Statistics:

```
-----  
DEVICE: token ring 0  
recv packets:   37383  
  recv sizes (bytes):  avg 63.5  min 50  max 1514  sdev 44.1  
  recv times (msec):  avg 0.008  min 0.005  max 0.048  sdev 0.003  
  demux times (msec):  avg 0.046  min 0.005  max 0.569  sdev 0.024  
xmit packets:   74328  
  xmit sizes (bytes):  avg 1508.3  min 50  max 1514  sdev 89.0  
  xmit times (msec):  avg 35.348  min 0.130  max 7837.976  sdev 164.951  
  
-----
```

Detailed TCP Socket Call Statistics (by Process):

```
-----  
PROCESS: ftp  PID: 19060  
reads:          36  
  
-----
```

```

read sizes (bytes):  avg 4021.3  min 4000   max 4096   sdev 39.9
read times (msec):  avg 5.616   min 0.030  max 72.955 sdev 15.228

writes:
write sizes (bytes): avg 66019.2 min 6      max 66346  sdev 4637.1
write times (msec):  avg 38.122  min 0.115  max 542.537 sdev 14.785

PROCESS: aixterm  PID: 16050
reads:
read sizes (bytes):  avg 32.0    min 32     max 32     sdev 0.0
read times (msec):  avg 0.030   min 0.021  max 0.087  sdev 0.009
writes:
write sizes (bytes): avg 62.8    min 28     max 292    sdev 55.7
write times (msec):  avg 0.092   min 0.052  max 0.209  sdev 0.030

PROTOCOL: TCP (All Processes)
reads:
read sizes (bytes):  avg 1111.8  min 32     max 4096   sdev 1772.6
read times (msec):  avg 1.542   min 0.021  max 72.955 sdev 8.302
writes:
write sizes (bytes): avg 56547.3 min 6      max 66346  sdev 23525.1
write times (msec):  avg 32.661  min 0.052  max 542.537 sdev 19.107

```

Note that the values in the detailed report show the average, minimum, maximum, and standard deviation values for the process, FLIH and SLIH, network device driver, and TCP socket call statistics over the monitored period.

Performance Diagnostic Tool (PDT)

The Performance Diagnostic Tool (PDT) package attempts to identify performance problems automatically by collecting and integrating a wide range of performance, configuration, and availability data. The data is regularly evaluated to identify and anticipate common performance problems. PDT assesses the current state of a system and tracks changes in workload and performance.

PDT data collection and reporting are easily enabled, and no further administrator activity is required. While many common system performance problems are of a specific nature, PDT also attempts to apply some general concepts of well-performing systems to search for problems. Some of these concepts are:

- ▶ Balanced use of resources
- ▶ Operation within bounds
- ▶ Identified workload trends
- ▶ Error-free operation
- ▶ Changes investigated
- ▶ Appropriate setting of system parameters

The PDT programs reside in `/usr/sbin/perf/diag_tool` and are part of the `bos.perf.diag_tool` fileset, which is installable from the AIX base installation media.

6.1 PDT

To start the PDT configuration, enter:

```
/usr/sbin/perf/diag_tool/pdt_config
```

The **pdt_config** is a menu-driven program. Refer to 6.2, “Examples for PDT” on page 106 for its use.

To run the master script, enter:

```
/usr/sbin/perf/diag_tool/Driver_ <profile>
```

The master script, **Driver_**, only takes one parameter: the name of the collection profile for which activity is being initiated. This name is used to select which `_.sh` files to run. For example, if **Driver_** is executed with `$1=daily`, then only those `.sh` files listed with a daily frequency are run. Check the respective control files to see which `.sh` files are driven by which profile names.

<code>daily</code>	Collection routines for those <code>_.sh</code> files that belong to the <code>daily</code> profile. Normally this is only information gathering.
<code>daily2</code>	Collection routines for those <code>_.sh</code> files that belong to the <code>daily2</code> profile. Normally this is only reporting on previously collected information.
<code>offweekly</code>	Collection routines for those <code>_.sh</code> files that belong to the <code>offweekly</code> profile.

6.1.1 Information about measurement and sampling

The PDT package consists of a set of shell scripts that invoke AIX commands. When enabled, the collection and reporting scripts will run under the *adm* user.

The master script, **Driver_**, is started by the cron daemon entry `PDT:cron;Daemons:cron;cron;` Monday through Friday at 9:00 and 10:00 in the morning and every Sunday at 21:00 unless changed manually by editing the *crontab* entries. Each time the **Driver_** script is started it runs with different parameters.

6.2 Examples for PDT

To start PDT, run the following command and use the menu-driven configuration program to perform the basic setup:

```
/usr/sbin/perf/diag_tool/pdt_config
```


As `pdt_config` has a menu-driven interface, follow the menus. Example 6-1 shows the main menu.

Example 6-1 PDT customization menu

```
_____PDT customization menu_____
1) show current PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable PDT reporting
4) modify/enable PDT collection
5) disable PDT collection
6) de-install PDT
7) exit pdt_config
Please enter a number:
```

First check the current setting by selecting 1, as shown in Example 6-2.

Example 6-2 PDT current setting

```
current PDT report recipient and severity level
root 3
_____PDT customization menu_____
1) show current PDT report recipient and severity level
2) modify/enable PDT reporting
3) disable PDT reporting
4) modify/enable PDT collection
5) disable PDT collection
6) de-install PDT
7) exit pdt_config
Please enter a number:
```

Example 6-2 states level 3 reports are to be made and sent to the root user on the local system. To check whether root has a mail alias defined, run the following command:

```
grep ^root /etc/aliases
```

If nothing is returned, the mail should be delivered to the local node. If there is a return value, it is used to provide an alternate destination address. For example:

```
root:pdt@collector.itso.ibm.com,"|/usr/bin/cat >>/tmp/log"
```

This shows that mail for the root user is routed to another user on another host, in this case the user `pdt` on host `collector.itso.ibm.com®`, and the mail will also be appended to the `/tmp/log` file.

By default, the **Driver_** program reports are generated with severity level 1 with only the most serious problems identified. Severity levels 2 and 3 are more detailed. By default, the reports are mailed to the adm user, but can be changed to *root* or not sent at all.

The configuration program updates the adm user's crontab file. Check the changes made by using the **cronadm** command as in Example 6-3.

Example 6-3 Checking the PDT crontab entry

```
# cronadm cron -l adm|grep diag_tool
0 9 * * 1-5 /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5 /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6 /usr/sbin/perf/diag_tool/Driver_ offweekly
```

It could also be done by using **grep** on the crontab file as shown in Example 6-4.

Example 6-4 Another way of checking the PDT crontab entry

```
# grep diag_tool /var/spool/cron/crontabs/adm
0 9 * * 1-5 /usr/sbin/perf/diag_tool/Driver_ daily
0 10 * * 1-5 /usr/sbin/perf/diag_tool/Driver_ daily2
0 21 * * 6 /usr/sbin/perf/diag_tool/Driver_ offweekly
```

The **daily** parameter makes the **Driver_** program collect data and store it in the */var/perf/tmp* directory. The programs that do the actual collecting are specified in the */var/perf/cfg/diag_tool/.collection.control* file. These programs are also located in the */usr/sbin/perf/diag_tool* directory.

The **daily2** parameter makes the **Driver_** program create a report from the */var/perf/tmp* data files and e-mails it to the recipient specified in the */var/perf/cfg/diag_tool/.reporting.list* file. The **PDT_REPORT** is the formatted version, and the **.SM_RAW_REPORT** is the unformatted report file.

6.2.1 Editing the configuration files

Some configuration files for PDT should be edited to better reflect the needs of a specific system.

Finding PDT files and directories

PDT analyzes files and directories for systematic growth in size. It examines only those files and directories listed in the file */var/perf/cfg/diag_tool/.files*. The format of the *.files* file is one file or directory name per line. The default content of this file is as shown in Example 6-5 on page 109.

Example 6-5 .files file

```
/usr/adm/wtmp  
/var/spool/qdaemon/  
/var/adm/ras/  
/tmp/
```

You can use an editor or just append using the command **print filename >> .files** to modify this file to track files and directories that are important to your system.

Monitoring hosts

PDT tracks the average ECHO_REQUEST delay to hosts whose names are listed in the `/var/perf/cfg/diag_tool/.nodes` file. This file is not shipped with PDT (which means that no host analysis is performed by default), but may be created by the administrator. The file should contain a hostname or TCP/IP address for each host that is to be monitored (pinged). Each line in the `.nodes` file should only contain either a hostname or IP address. In the following example, we will monitor the connection to the Domain Name Server (DNS). Example 6-6 shows how to check which nameserver a DNS client is using by examining the `/etc/resolv.conf` file.

Example 6-6 ./etc/resolv.conf file

```
# awk '/nameserver/{print $2}' /etc/resolv.conf  
9.3.4.2
```

To monitor the nameserver shown in the example, the `.nodes` file could contain the IP address on a separate line, as in Example 6-7.

Example 6-7 .nodes file

```
# cat .nodes  
9.3.4.2
```

Changing thresholds

The file `/var/perf/cfg/diag_tool/.thresholds` contains the thresholds used in analysis and reporting. These thresholds have an effect on PDT report organization and content. Example 6-8 is the content of the default file.

Example 6-8 .thresholds default file

```
# grep -v ^# .thresholds  
DISK_STORAGE_BALANCE 800  
PAGING_SPACE_BALANCE 4  
NUMBER_OF_BALANCE 1  
MIN_UTIL 3  
FS_UTIL_LIMIT 90
```

MEMORY_FACTOR .9
TREND_THRESHOLD .01
EVENT_HORIZON 30

The settings in the example are the default values. The thresholds are:

DISK_STORAGE_BALANCE	The SCSI controllers having the largest and smallest disk storage are identified. This is a static size, not the amount allocated or free. The default value is 800. Any integer value between zero (0) and 10000 is valid.
PAGING_SPACE_BALANCE	The paging spaces having the largest and the smallest areas are identified. The default value is 4. Any integer value between zero (0) and 100 is accepted. This threshold is presently not used in analysis and reporting.
NUMBER_OF_BALANCE	The SCSI controllers having the greatest and fewest number of disks attached are identified. The default value is one (1). It can be set to any integer value from zero (0) to 10000.
MIN_UTIL	Applies to process utilization. Changes in the top three CPU consumers are only reported if the new process had a utilization in excess of MIN_UTIL. The default value is 3. Any integer value from zero (0) to 100 is valid.
FS_UTIL_LIMIT	Applies to journaled file system utilization. Any integer value between zero (0) and 100 is accepted.
MEMORY_FACTOR	The objective is to determine whether the total amount of memory is adequately backed up by paging space. The formula is based on experience and actually compares $\text{MEMORY_FACTOR} * \text{memory}$ with the average used paging space. The current default is .9. By decreasing this number, a warning is produced more frequently. Increasing this number eliminates the message altogether. It can be set anywhere between .001 and 100.
TREND_THRESHOLD	Used in all trending assessments. It is applied after a linear regression is performed on all available historical data. This technique basically draws the best line among the points. The slope of the fitted line must exceed the $\text{last_value} * \text{TREND_THRESHOLD}$. The objective is to try to ensure that a trend, however strong its statistical significance, has some practical

significance. The threshold can be set anywhere between 0.00001 and 100000.

EVENT_HORIZON

Also used in trending assessments. For example, in the case of file systems, if there is a significant (both statistical and practical) trend, the time until the file system is 100 percent full is estimated. The default value is 30, and it can be any integer value between zero (0) and 100000.

6.2.2 Using reports generated by PDT

Example 6-9 shows the default-configured level 3 report. It is an example of what will be delivered by e-mail every day.

Example 6-9 PDT sample e-mail report

Performance Diagnostic Facility 1.0

Report printed: Fri Apr 4 11:14:27 2003

Host name: lpar05
Range of analysis includes measurements
from: Hour 10 on Friday, April 4th, 2003
to: Hour 11 on Friday, April 4th, 2003

Notice: To disable/modify/enable collection or reporting
execute the pdt_config script as root

----- Alerts -----

I/O CONFIGURATION

- Note: volume hdisk1 has 14112 MB available for allocation while volume hdisk0 has 8032 MB available

PAGING CONFIGURATION

- Physical Volume hdisk1 (type: SCSI) has no paging space defined
- All paging spaces have been defined on one Physical volume (hdisk0) **I/O**

I/O BALANCE

- Phys. volume cd0 is not busy
volume cd0, mean util. = 0.00 %
- Phys. volume hdisk1 is not busy
volume hdisk1, mean util. = 0.00 %

PROCESSES

- First appearance of 15628 (ksh) on top-3 cpu list
(cpu % = 7.10)

- First appearance of 19998 (java) on top-3 cpu list (cpu % = 24.40)
- First appearance of 15264 (java) on top-3 cpu list (cpu % = 24.40)
- First appearance of 7958 (java) on top-3 cpu list

FILE SYSTEMS

- File system hd2 (/usr) is nearly full at 92 %

----- System Health -----

SYSTEM HEALTH

- Current process state breakdown:
 - 74.20 [99.5 %] : active
 - 0.40 [0.5 %] : zombie
 - 74.60 = TOTAL
- [based on 1 measurement consisting of 10 2-second samples]

----- Summary -----

This is a severity level 3 report
No further details available at severity levels > 3

The PDT_REPORT, at level 3, will have the following report sections:

- ▶ Alerts
- ▶ Upward Trends
- ▶ Downward Trends
- ▶ System Health
- ▶ Other
- ▶ Summary

And subsections such as the following:

- ▶ I/O CONFIGURATION
- ▶ PAGING CONFIGURATION
- ▶ I/O BALANCE
- ▶ PROCESSES
- ▶ FILE SYSTEMS
- ▶ VIRTUAL MEMORY

Example 6-10 shows the raw information from the .SM_RAW_REPORT file that is used for creating the PDT_REPORT file.

Example 6-10 .SM_RAW_REPORT file

```
H 1 | Performance Diagnostic Facility 1.0
H 1 |

H 1 | Report printed: Fri Apr  4 10:00:00 2003
H 1 |

H 1 | Host name: lpar05
```

```
H 1 | Range of analysis includes measurements
H 1 |   from: Hour 10 on Friday, April 4th, 2003
H 1 |   to: Hour 11 on Friday, April 4th, 2003
H 1 |
...(lines omitted)...
```

The script in Example 6-11 shows how to extract report subsections from the PDT_REPORT file. In this example it displays all subsections in turn.

Example 6-11 Script to extract subsections

```
#!/bin/ksh

set -A tab "I/O CONFIGURATION" "PAGING CONFIGURATION" "I/O BALANCE" \
          "PROCESSES" "FILE SYSTEMS" "VIRTUAL MEMORY"

for string in "${tab[@]};do
    grep -p "$string" /var/perf/tmp/PDT_*
done
```

Example 6-12 shows a sample output from the script in Example 6-11 using the same data as in Example 6-9 on page 111.

Example 6-12 Output from extract subsection script

```
I/O CONFIGURATION
- Note: volume hdisk1 has 14112 MB available for allocation
  while volume hdisk0 has 8032 MB available

PAGING CONFIGURATION
- Physical Volume hdisk1 (type: SCSI) has no paging space defined
- All paging spaces have been defined on one Physical volume (hdisk1)

I/O BALANCE
- Phys. volume cd0 is not busy
  volume cd0, mean util. = 0.00 %
- Phys. volume hdisk1 is not busy
  volume hdisk1, mean util. = 0.00 %

PROCESSES
- First appearance of 15628 (ksh) on top-3 cpu list
  (cpu % = 7.10)
- First appearance of 19998 (java) on top-3 cpu list
  (cpu % = 24.40)
- First appearance of 15264 (java) on top-3 cpu list
  (cpu % = 24.40)
```

- First appearance of 7958 (java) on top-3 cpu list (cpu % = 24.40)

FILE SYSTEMS

- File system hd2 (/usr) is nearly full at 92 %
-

6.2.3 Creating a PDT report manually

As an alternative to using the periodic report, any user can request a current report from the existing data by executing

```
/usr/sbin/perf/diag_tool/pdt_report #
```

where # is a severity number from one (1) to three (3). The report is produced with the given severity (if none is provided, it defaults to one) and is written to standard output. Generating a report in this way does not cause any change to the /var/perf/tmp/PDT_REPORT files.

Running PDT collection manually

In some cases, you might want to run the collection manually or by other means than using **cron**. You simply run the **Driver_** script with options as in the cronfile. The following example will perform the basic collection:

```
/usr/sbin/perf/diag_tool/Driver_ daily
```


The perfpmr command

perfpmr consists of a set of utilities that build a test case containing the necessary information to assist in analyzing performance issues. It is primarily designed to assist IBM software support, but is also useful as a documentation tool for your system.

As **perfpmr** is updated frequently, it is not distributed on AIX media. It can be downloaded from <ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr> – use the version that is appropriate for your AIX level. For our case, the file that we need is distributed in:

<ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/perf52/perf52.tar.Z>

7.1 perfpmr

The syntax of the **perfpmr** command is:

```
perfpmr.sh [-PDgfnpsc][-F file][-x file][-d sec] monitor_seconds
```

Flags

-P	Preview only - show scripts to run and disk space needed.
-D	Run perfpmr the original way without a perfpmr.cfg file.
-g	Do not collect gennames output.
-f	If gennames is run, specify gennames -f
-n	Used if no netstat or nfsstat is desired.
-p	Used if no pprof collection is desired while monitor.sh running.
-s	Used if no svmon is desired.
-c	Used if no configuration information is desired.
-F	File use file as the perfpmr cfg file - default is perfpmr.cfg
-x	File only execute file found in perfpmr installation directory.
-d	sec is time to wait before starting collection period, default is delay_seconds 0
-s	Used if svmon outout is not required.

Parameters

monitor_seconds Collection period in seconds. The minimum period is 60 seconds.

Use **perfpmr.sh 600** for a standard collection period of 600 seconds.

7.1.1 Information about measurement and sampling

Unless you run the shell scripts separately, **perfpmr.sh 600** executes the following shell scripts to obtain a test case. You can also run these scripts on their own. Refer to “Running perfpmr” on page 123 for details.

config.sh	Collects configuration information into a report called config.sum.
emstat.sh time	Builds a report called emstat.int on emulated instructions. The time parameter must be greater than or equal to 60.

filemon.sh time	Builds a report called filemon.sum on file I/O. The time parameter does not have any restrictions.
iostat.sh time	Builds two reports on I/O statistics: a summary report called iostat.sum and an interval report called iostat.int. The time parameter must be greater than or equal to 60.
iptrace.sh time	Builds a raw Internet Protocol (IP) trace report on network I/O called iptrace.raw. You can convert the iptrace.raw file to a readable ipreport file called iptrace.int using the iptrace.sh -r command. The time parameter does not have any restrictions.
monitor.sh time	Invokes system performance monitors and collects interval and summary reports:
lsps.after	Contains lsps -a and lsps -s output after monitor.sh was run. Used to report on paging space use.
lsps.before	Contains lsps -a and lsps -s output before monitor.sh was run. Used to report on paging space use.
nfsstat.int	Contains nfsstat -m and nfsstat -csnr output before and after monitor.sh was run. Used to report on Network File System use and configuration.
monitor.int	Contains samples by interval using ps -efk (showing active processes before and after monitor.sh was run). It also contains sadc , sar -A , iostat , vmstat , and emstat output.
monitor.sum	Contains samples by summary using ps -efk (showing changes in ps output for active processes before and after monitor.sh was run). It also contains sadc , sar -A , iostat , vmstat , and emstat outputs.
pprof.trace.raw	Contains the raw trace for pprof .
psb.elfk	Contains a modified ps -e1k output before monitor.sh was run.
svmon.after	Contains svmon -G and svmon -Pns output and top segments use by process with the svmon -S command

after monitor.sh was run. Used to report on memory use.

svmon.before Contains **svmon -G** and **svmon -Pns** output and top segment use by process with the **svmon -S** command before monitor.sh was run. Used to report on memory use.

vmstati.after Contains **vmstat -i** output after monitor.sh was run. Used to report on I/O device interrupts.

vmstati.before Contains **vmstat -i** output before monitor.sh was run. Used to report on I/O device interrupts.

netstat.sh [-r] time

Builds a report on network configuration and use called netstat.int containing **tokstat -d** of the token-ring interfaces, **entstat -d** of the Ethernet interfaces, **netstat -in**, **netstat -m**, **netstat -rn**, **netstat -rs**, **netstat -s**, **netstat -D**, and **netstat -an** before and after monitor.sh was run. You can reset the Ethernet and token-ring statistics and re-run this report by running **netstat.sh -r 60**. The time parameter must be greater than or equal to 60.

nfsstat.sh time

Builds a report on NFS configuration and use called netstat.int containing **nfsstat -m**, and **nfsstat -csnr** before and after nfsstat.sh was run. The time parameter must be greater than or equal to 60.

pprof.sh time

Builds a file called pprof.trace.raw that can be formatted with the **pprof.sh -r** command. Refer to 19.3.2, “Examples for pprof” on page 311 for more details. The time parameter does not have any restrictions.

ps.sh time

Builds reports on process status (ps). ps.sh creates the following files:

psa.elfk A **ps -elfk** listing after ps.sh was run.

psb.elfk A **ps -elfk** listing before ps.sh was run.

ps.int Active processes before and after ps.sh was run.

ps.sum A summary report of the changes between when ps.sh started and finished. This is useful

for determining what processes are consuming resources.

The time parameter must be greater than or equal to 60.

sar.sh time

Builds reports on sar. sar.sh creates the following files:

sar.int Output of commands **sadc 10 7** and **sar -A**
sar.sum A **sar** summary over the period sar.sh was run

The time parameter must be greater than or equal to 60.

tcpdump.sh int.time

The int. parameter is the name of the interface; for example, tr0 is token-ring. Creates a raw **trace** file of a TCP/IP dump called tcpdump.raw. To produce a readable tcpdump.int file, use the **tcpdump.sh -r** command. The time parameter does not have any restrictions.

tprof.sh time

Creates a **tprof** summary report called tprof.sum. Used for analyzing memory use of processes and threads. You can also specify a program to profile by specifying the **tprof.sh -p program 60** command, which enables you to profile the executable-called program for 60 seconds. The time parameter does not have any restrictions.

trace.sh time

Creates the raw trace files (trace*) from which an ASCII trace report can be generated using the **trcrpt** command or by running **trace.sh -r**. This command creates a file called trace.int that contains the readable trace. Used for analyzing performance problems. The time parameter does not have any restrictions.

vmstat.sh time

Builds reports on **vmstat**: a **vmstat** interval report called vmstat.int and a **vmstat** summary report called vmstat.sum. The **time** parameter must be greater than or equal to 60.

Due to the volume of data **trace** collects, the **trace** will only run for five seconds (by default), so it is possible that it will not be running when the performance problems occur on your system, especially if performance problems occur for short periods. In this case, it would be advisable to run the **trace** by itself for a period of 15 seconds when the problem is present. The command **trace.sh 15** runs a trace for 15 seconds.

An RS/6000 SP can produce a test case of 135 MB, with 100 MB just for the traces. This size can vary considerably depending on system load. If you run the trace on the same system with the same workload for 15 seconds, then you could expect the trace files to be approximately 300 MB in size.

One raw trace file per CPU is produced. The files are called trace.raw-0, trace.raw-1, and so forth for each CPU. An additional raw trace file called trace.raw is also generated. This is a master file that has information that ties in the other CPU-specific traces. To merge the trace files together to form one raw trace file, run the following commands:

```
trcrpt -C all -r trace.raw > trace.r  
rm trace.raw*
```

7.1.2 Building and submitting a test case

You may be asked by IBM to supply a test case for a performance problem or you may wish to run **perfpmr.sh** for your own requirements (for example, to produce a base line for detecting future performance problems). In either case, **perfpmr.sh** is the tool to collect performance data. Even if your performance problem is attributed to one component of your system, such as the network, **perfpmr.sh** is still the way to send a test case because it contains other information that is required for problem determination. Additional information for problem determination may be requested by IBM software support.

Note: IBM releases Maintenance Levels for AIX. These are a collection of Program Temporary Fixes (PTFs) used to upgrade the operating system to the latest level, but remaining within your current release. Often these, along with the current version of micro-code for the disks and adapters, have performance enhancement fixes. You may therefore wish to load these.

There are five stages to building and sending a test case. These steps must be completed when you are logged in as root. The steps are listed as follows:

- ▶ Prepare to download **perfpmr**
- ▶ Download **perfpmr**
- ▶ Install **perfpmr**
- ▶ Run **perfpmr**
- ▶ Upload the test case

Preparing for perfpmr

These filesets should be installed before running **perfpmr.sh**:

- ▶ bos.acct
- ▶ bos.sysmgt.trace
- ▶ perfagent.tools
- ▶ bos.net.tcp.server
- ▶ bos.adt.include
- ▶ bos.adt.samples

Downloading perfpmr

The **perfpmr** is downloadable from:

<ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr>

Using a browser, download the version that is applicable to your version of AIX. The file size should be under 1 MB.

Important: Always download a new copy of **perfpmr** in case of changes. Do not use an existing pre-downloaded copy.

If you have downloaded **perfpmr** to a PC, transfer it to the system in binary mode using **ftp**, placing it in an empty directory.

Installing perfpmr

Uncompress and extract the file with the **tar** command. The directory contains:

- ▶ Install
- ▶ PROBLEM.INFO
- ▶ README
- ▶ config.sh
- ▶ emstat.sh
- ▶ filemon.sh
- ▶ getdate
- ▶ getevars
- ▶ iostat.sh
- ▶ iptrace.sh
- ▶ lsc
- ▶ memfill
- ▶ monitor.sh
- ▶ netstat.sh
- ▶ nfsstat.sh
- ▶ perfpmr.cfg
- ▶ perfpmr.sh
- ▶ pprof.sh
- ▶ ps.sh
- ▶ pstat.sh
- ▶ sar.sh
- ▶ setpri
- ▶ setsched
- ▶ svmon
- ▶ tcpdump.sh
- ▶ tprof.sh
- ▶ trace.sh
- ▶ vmstat.sh

In the directory you will notice files ending in `.sh`. These are shell scripts that may be run separately. Normally these shell scripts are run automatically by running `perfpmr.sh`. Read the README file to find any additional steps that may be applicable to your system.

Install `perfpmr` by running `./Install`. This will replace the following files in the `/usr/bin` directory with symbolic links to the files in the directory where you installed `perfpmr`:

- ▶ `config.sh`
- ▶ `curt`
- ▶ `emstat.sh`
- ▶ `filemon.sh`
- ▶ `getevars`
- ▶ `hd_pbuf_cnt.sh`
- ▶ `iostat.sh`
- ▶ `iptrace.sh`
- ▶ `lsc`
- ▶ `monitor.sh`
- ▶ `netstat.sh`
- ▶ `nfsstat.sh`
- ▶ `perfpmr.sh`
- ▶ `pprof.sh`
- ▶ `ps.sh`
- ▶ `sar.sh`
- ▶ `setpri`
- ▶ `tcpdump.sh`
- ▶ `tprof.sh`
- ▶ `trace.sh`
- ▶ `utld`
- ▶ `vmstat.sh`

The output of the installation procedure will be similar to Example 7-1.

Example 7-1 perfpmr installation screen

```
# ./Install
```

```
(C) COPYRIGHT International Business Machines Corp., 2000
```

```
PERFPMR Installation started...
```

```
PERFPMR Installation completed.
```

Running perfpmr

There are two scenarios to consider when running `perfpmr`.

- ▶ If your system is performing poorly for long periods of time and you can predict when it runs slow, then you can run `./perfpmr.sh 600`.
- ▶ In some situations, a system may perform normally but will run slow at various times of the day. If you run `perfpmr.sh 600` then there is a chance that `perfpmr` might not have captured the performance slowdown. In this case you could run the scripts manually when the system is slow and use a longer time-out period: for example, a `trace.sh 15` will perform a trace for 15 seconds instead of the default five seconds. We would still need a `perfpmr.sh 600` to be initially run before running individual scripts. This will ensure that all of the data and configuration have been captured.

Attention: If you are using HACMP, then you may want to extend the Dead Man Switch (DMS) time-out or shut down HACMP prior to collecting `perfpmr` data to avoid accidental failovers.

After executing `perfpmr.sh`, it creates the files in Table 7-1.

Table 7-1 Files created by `perfpmr`

config.sum	crontab_l	devtree.out
errpt_a	etc_security_limits	filemon.sum
genkex.out	genkld.out	gennames.out
getevars.out	iptrace.raw	lsps.after
lsps.before	lsrset.out	monitor.int
monitor.sum	netstat.int	nfsstat.int
perfpmr.int	pprof.trace.raw	psa.elfk
psb.elfk	psemo.after	psemo.before
svmon.after	svmon.before	tcpdump.raw
tprof.csyms	tprof.ctrc	tprof.out
tprof.sum	trace.crash.inode	trace.fmt
trace.inode	trace.j2.inode	trace.maj_min2lv
trace.nm	trace.raw	trace.raw-0
trace.raw-1	trace.raw-10	trace.raw-11
trace.raw-12	trace.raw-13	trace.raw-14

trace.raw-15	trace.raw-2	trace.raw-3
trace.raw-4	trace.raw-5	trace.raw-6
trace.raw-7	trace.raw-8	trace.raw-9
trace.syms	tunables_lastboot	tunables_lastboot.log
tunables_nextboot	vfs.kdb	vmstat_v.after
vmstat_v.before	vmstati.after	vmstati.before
vnode.kdb	w.int	

Tip: After you have installed **perfpmr** you can run it at any time to make sure that all of the files described above are captured. By doing this, you can be confident that you will get a full test case.

Uploading the test case

The directory also contains a file called **PROBLEM.INFO** that must be completed. Bundle the files together using the **tar** command and upload the file to IBM as documented in the **README** files.

7.2 Examples for perfpmr

Example 7-2 is an example of running **perfpmr.sh 600**.

Example 7-2 Running perfpmr.sh

```
# perfpmr.sh 600
C) COPYRIGHT International Business Machines Corp., 2000

PERFPMR: perfpmr.sh Version 520 2003/02/24
PERFPMR: Parameters passed to perfpmr.sh: 600
PERFPMR: Data collection started in foreground (renice -n -20)

TRACE.SH: Starting trace for 5 seconds
TRACE.SH: Data collection started
TRACE.SH: Data collection stopped
TRACE.SH: Trace stopped
TRACE.SH: Trcnm data is in file trace.nm
TRACE.SH: /etc/trcfmt saved in file trace.fmt
TRACE.SH: Binary trace data is in file trace.raw

MONITOR: Capturing initial lsps and vmstat data
MONITOR: Starting system monitors for 600 seconds.
MONITOR: Waiting for measurement period to end....
```

```
MONITOR: Capturing final lsps and vmstat data
MONITOR: Generating reports....
MONITOR: Network reports are in netstat.int and nfsstat.int
MONITOR: Monitor reports are in monitor.int and monitor.sum

IPTRACE: Starting iptrace for 10 seconds....
0513-059 The iptrace Subsystem has been started. Subsystem PID is 28956.
0513-044 The iptrace Subsystem was requested to stop.
IPTRACE: iptrace collected....
IPTRACE: Binary iptrace data is in file iptrace.raw

FILEMON: Starting filesystem monitor for 60 seconds....
FILEMON: tracing started
FILEMON: tracing stopped
FILEMON: Generating report....

TPROF: Starting tprof for 60 seconds....
TPROF: Sample data collected....
TPROF: Generating reports in background (renice -n 20)
TPROF: Tprof report is in tprof.sum

CONFIG.SH: Generating SW/HW configuration
WLM is running
CONFIG.SH: Report is in file config.sum

PERFPMR: Data collection complete.
```

Tip: It is useful to run **perfpmr** when your system is under load and performing normally. This gives you a baseline to determine future performance problems.

You should run **perfpmr** again when:

- ▶ Your system is experiencing performance problems.
- ▶ You make hardware changes to the system.
- ▶ You make any changes to your network configuration.
- ▶ You make changes to the AIX Operating System, such as when you install upgrades or tune AIX.
- ▶ You make changes to your application.

Archived

The ps command

The **ps** (Process Status) command produces a list of processes on the system that can be used to determine how long a process has been running, how much CPU resource the processes are using, and whether processes are being penalized by the system. It also shows how much memory processes are using, how much I/O a process is performing, the priority and nice values for the process, and who created the process.

The **ps** executable resides in `/usr/bin` and is part of the `bos.rte.commands` fileset, which is installed by default from the AIX base installation media.

8.1 ps

The syntax of the **ps** command depends on the standard being used:

► X/Open standard

```
ps [-ARNaedfklm] [-n namelist] [-F Format] [-o specifier[=header],...][-p  
proclist][-G|-g grouplist] [-t termlist] [-U|-u userlist] [-c classlist]
```

► Berkeley standard

```
ps [ a ] [ c ] [ e ] [ ew ] [ eww ] [ g ] [ n ] [ U ] [ w ] [ x ] [ l | s | u |  
v ] [ t Tty ] [ ProcessNumber ]
```

The following flags are all preceded by a - (minus sign):

- A** Writes information about all processes to standard output.
- a** Writes information about all processes except the session leaders and processes not associated with a terminal to standard output.
- c Clist** Displays only information about processes assigned to the Workload Manager (WLM) classes listed in the Clist variable. The Clist variable is either a comma separated list of class names or a list of class names is enclosed in double quotation marks (" ") that are separated from one another by a comma, by one or more spaces, or both.
- d** Writes information to standard output about all processes except the session leaders.
- e** Writes information to standard output about all processes except the kernel processes.
- F Format** Equivalent to the -o Format flag.
- f** Generates a full listing.
- G Glist** Writes information to standard output only about processes that are in the process groups listed for the Glist variable. The Glist variable is either a comma-separated list of process group identifiers or a list of process group identifiers enclosed in double quotation marks (" ") and separated from one another by a comma or by one or more spaces.
- g Glist** Equivalent to the -G Glist flag.
- k** Lists kernel processes.
- l** Generates a long listing.
- m** Lists kernel threads as well as processes. Output lines for processes are followed by an additional output line for each kernel thread. This flag does not display thread-specific fields (bnd,

scount, sched, thcount, and tid) unless the appropriate -o Format flag is specified.

- N** Gathers no thread statistics. With this flag, **ps** simply reports those statistics that can be obtained by not traversing through the threads chain for the process.
- n NameList** Specifies an alternative system name-list file in place of the default. This flag is not used by AIX.
- o Format** Displays information in the format specified by the Format variable. Multiple field specifiers can be specified for the Format variable. The Format variable is either a comma-separated list of field specifiers or a list of field specifiers enclosed within a set of " " (double-quotation marks) and separated from one another by a comma, one or more spaces, or both. Each field specifier has a default header. The default header can be overridden by appending an = (equal sign) followed by the user-defined text for the header. The fields are written in the order specified on the command line in column format. The field widths are specified by the system to be at least as wide as the default or user-defined header text. If the header text is null (such as if -o user= is specified), the field width is at least as wide as the default header text. If all header fields are null, no header line is written.
- p Plist** Displays only information about processes with the process numbers specified for the Plist variable. The Plist variable is either a comma-separated list of Process ID (PID) numbers or a list of process ID numbers enclosed in double quotation marks (" ") and separated from one another by a comma, one or more spaces, or both.
- t Tlist** Displays only information about processes associated with the workstations listed in the Tlist variable. The Tlist variable is either a comma separated list of workstation identifiers or a list of workstation identifiers enclosed in double quotation marks (" ") and separated from one another by a comma, one or more spaces, or both.
- U Ulist** Displays only information about processes with the user ID numbers or login names specified in the Ulist variable. The Ulist variable is either a comma-separated list of user IDs or a list of user IDs enclosed in double quotation marks (" ") and separated from one another by a comma and one or more spaces. In the listing, the **ps** command displays the numerical user ID unless the -f flag is used, in which case the command displays the login name. See also the u flag.
- u Ulist** Equivalent to the -U Ulist flag.

The following options are not preceded by a - (minus sign):

- a** Displays information about all processes with terminals (ordinarily only the user's own processes are displayed).
- c** Displays the command name, as stored internally in the system for purposes of accounting, rather than the command parameters, which are kept in the process address space.
- e** Displays the environment as well as the parameters to the command, up to a limit of 80 characters.
- ew** Wraps display from the e flag one extra line.
- eww** Wraps display from the e flag as many times as necessary.
- g** Displays all processes.
- l** Displays a long listing of the F, S, UID, PID, PPID, C, PRI, NI, ADDR, SZ, PSS, WCHAN, TTY, TIME, and CMD fields.
- n** Displays numerical output. In a long listing, the WCHAN field is printed numerically rather than symbolically. In a user listing, the USER field is replaced by a UID field.
- s** Displays the size (SSIZ) of the kernel stack of each process (for use by system maintainers) in the basic output format. This value is always 0 (zero) for a multi-threaded process.
- t tty** Displays processes whose controlling tty is the value of the tty variable, which should be specified as printed by the **ps** command; that is, 0 for terminal /dev/tty0, lft0 for /dev/lft0, and pts/2 for /dev/pts/2.
- u** Displays user-oriented output. This includes the USER, PID, %CPU, %MEM, SZ, RSS, TTY, STAT, STIME, TIME, and COMMAND fields.
- v** Displays the PGIN, SIZE, RSS, LIM, TSIZ, TRS, %CPU, and %MEM fields.
- w** Specifies a wide-column format for output (132 columns rather than 80). If repeated (for example, ww), uses arbitrarily wide output. This information is used to decide how much of long commands to print.
- x** Displays processes with no terminal.

8.1.1 Information about measurement and sampling

The **ps** command is useful for determining:

- ▶ How long a process has been running on the system
- ▶ How much CPU resource a process is using
- ▶ If processes are being penalized by the system
- ▶ How much memory a process is using

- ▶ How much I/O a process is performing
- ▶ The priority and nice values for the process
- ▶ Who created the process

8.2 Examples for ps

The following examples can be used to analyze performance problems using **ps**.

8.2.1 Displaying the top 10 CPU-consuming processes

The commands in Example 8-1 are useful for determining the top 10 processes that are consuming the most CPU. The aux flags of the **ps** command display USER, PID, %CPU, %MEM, SZ, RSS, TTY, STAT, STIME, TIME, and COMMAND fields. The **sort -rn +2** is a reverse-order numeric sort of the third column, which is %CPU. The **head -10** displays only the first 10 processes.

Example 8-1 Displaying the top 10 CPU-consuming processes

```
# ps aux | head -1; ps aux | sort -rn +2 | head -10
```

USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	1290	24.7	0.0	12	12	-	A	Apr 01 2659:37		wait
root	1032	24.6	0.0	12	12	-	A	Apr 01 2649:17		wait
root	516	24.6	0.0	12	12	-	A	Apr 01 2650:22		wait
root	774	24.5	0.0	12	12	-	A	Apr 01 2634:07		wait
root	43268	0.1	0.0	1524	1544	pts/7	A	09:25:36	1:16	topas
root	37522	0.1	0.0	224	252	pts/4	A	17:14:37	4:57	/usr/WebSphere/Ap
root	5676	0.1	0.0	68	68	-	A	Apr 01 12:04		gil
root	44426	0.0	0.0	416	440	-	A	09:25:04	0:00	telnetd -a
root	44230	0.0	0.0	416	440	-	A	09:16:55	0:00	telnetd -a
root	43930	0.0	0.0	416	440	-	A	09:16:09	0:00	telnetd -a

The **wait** processes listed in the report show that this system is mainly idle. There are four **wait** processes, one for each CPU. You can determine how many processors your system has by running the **lsdev -Cc processor** command.

Example 8-2 Displaying number of processors in the system

```
#lsdev -Cc processor
```

proc18	Available	00-18	Processor
proc19	Available	00-19	Processor
proc22	Available	00-22	Processor
proc23	Available	00-23	Processor

In Example 8-3, a test program called `cpu` was started and, as can be observed, processes 31758, 14328, and 33194 used more CPU than wait. The report displays the `%CPU` column sorted in reverse numerical order. `%CPU` represents the percentage of time the process was actually consuming CPU resource in relation to the life of the process.

Example 8-3 Displaying the top 10 CPU-consuming processes

#	ps aux		head -1	;	ps aux		sort -rn +2		head	
USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	31758	24.7	2.0	4156	4152	pts/8	A	13:58:33	4:53	cpu 5
root	14328	24.5	2.0	4156	4152	pts/8	A	13:58:33	4:50	cpu 5
root	33194	24.3	2.0	4156	4152	pts/8	A	13:58:33	4:47	cpu 5
root	516	24.2	5.0	8	11536	-	A	May 11	9573:27	wait
root	1290	24.1	5.0	8	11536	-	A	May 11	9528:52	wait
root	774	24.1	5.0	8	11536	-	A	May 11	9521:18	wait
root	1032	24.0	5.0	8	11536	-	A	May 11	9494:31	wait
root	31256	11.2	2.0	4156	4152	pts/8	A	13:58:33	2:13	cpu 5
root	25924	11.2	2.0	4208	4204	pts/8	A	13:58:33	2:13	cpu 5
root	31602	1.6	0.0	1172	944	pts/10	A	10:37:21	13:29	xmwl1m

8.2.2 Displaying the top 10 memory-consuming processes

The following command line is useful for determining the percentage of real memory (size of working segment and the code-segment combined together) used by the process. The report shown in Example 8-4 displays the `%MEM` column sorted in reverse numerical order.

Example 8-4 Displaying the top 10 memory-consuming processes using RSS

#	ps aux		head -1	;	ps aux		sort -rn +3		head	
USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	32788	0.0	3.0	16	24160	-	A	Mar 28	0:00	mmkproc
root	32308	0.0	3.0	16	24160	-	A	Mar 28	0:00	mmkproc
root	27616	0.0	3.0	76	24220	-	A	Mar 28	3:58	vsd kp
root	24534	0.0	3.0	16	24160	-	A	Mar 28	0:00	mmkproc
root	22714	0.0	3.0	16	24160	-	A	Mar 28	0:07	nfsWatchKproc
root	18600	0.0	3.0	16	24160	-	A	Mar 28	0:00	cash
root	17546	0.0	3.0	20	24164	-	A	Mar 28	0:05	aump
root	12918	0.0	3.0	16	24160	-	A	Mar 28	0:13	jfsz
root	11928	0.0	3.0	16	24160	-	A	Mar 28	0:00	PM
root	7526	0.0	3.0	20	24164	-	A	Mar 28	0:00	rtcnd

Another way to determine memory use is to use the command line in Example 8-5 on page 133. The `SZ` represents the virtual size in kilobytes of the data section of the process. (This is sometimes displayed as `SIZE` by other flags). This number is equal to the number of working-segment pages of the process

that have been touched (that is, the number of paging-space slots that have been allocated) times four. File pages are excluded. If some working-segment pages are currently paged out, this number is larger than the amount of real memory being used. The report displays the SZ column sorted in reverse numerical order.

Example 8-5 Displaying the top 10 memory-consuming processes using SZ

```
# ps -ealf | head -1 ; ps -ealf | sort -rn +9 | head
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	STIME	TTY	TIME	CMD
240001	A	root	4712	5944	0	181	20	f19e	6836	30b50f10	May 20	-	4:58	/usr/lpp/X11/bin/X -Wjfp7a
240001	A	root	27146	3418	0	181	20	a4d7	5296	*	13:10:57	-	0:05	/usr/sbin/rsct/bin/IBM.FSrm
200001	A	root	33744	24018	0	181	20	c739	3856		May 22	pts/5	17:02	xmperf
240001	A	root	17042	3418	0	181	20	53ca	3032		May 20	-	3:01	/usr/opt/ifor/bin/i41lmd -b -n
200001	A	root	19712	26494	5	183	24	412a	2880		May 21	pts/9	27:32	xmperf
40001	A	root	17548	17042	0	181	20	7bcf	2644	309ceed8	May 20	-	0:00	/usr/opt/ifor/bin/i41lmd -b -n
240401	A	root	28202	4238	0	181	20	418a	2452		May 21	-	0:09	dtwm
240001	A	root	16048	3418	0	181	20	4baa	2356	*	May 22	-	0:03	/usr/sbin/rsct/bin/IBM.HostRMd
240001	A	root	4238	6196	0	181	20	9172	2288		May 21	-	0:10	/usr/dt/bin/dtsession
240001	A	root	17296	3418	0	181	20	fbdf	2160	*	May 20	-	0:00	/usr/sbin/rsct/bin/IBM.ERmd

8.2.3 Displaying the processes in order of being penalized

The following command line is useful for determining which processes are being penalized by the Virtual Memory Manager. See 1.2.2, “Processes and threads” on page 6 for details about penalizing processes. The maximum value for the C column is 120. The report in Example 8-6 displays the C column sorted in reverse numerical order.

Example 8-6 Displaying the processes in order of being penalized

```
# ps -eakl | head -1 ; ps -eakl | sort -rn +5
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
303	A	0	1290	0	120	255	--	b016	8		-	8570:28	wait
303	A	0	1032	0	120	255	--	a815	8		-	8540:22	wait
303	A	0	774	0	120	255	--	a014	8		-	8568:09	wait
303	A	0	516	0	120	255	--	9813	8		-	8590:49	wait
303	A	0	0	0	120	16	--	9012	12		-	3:53	swapper
240001	A	0	25828	1	34	187	24	2040	1172	30bf6fd8	-	27:25	xmwl
200001	A	0	36434	25250	4	181	20	da3e	460		pts/4	0:00	ps
240001	A	0	25250	29830	2	181	20	59ef	1020		pts/4	0:01	ksh
200001	A	0	36682	25250	2	181	20	69c9	300	30b4a6fc	pts/4	0:00	sort
200001	A	0	34898	25250	2	181	20	4b6a	236	3098fce0	pts/4	0:00	head

...(lines omitted)...

Ignoring the wait processes, which will always show 120, the xmwl process is being penalized by the CPU. When this occurs, the process is awarded less CPU time, thereby stopping xmwl from monopolizing the CPU and giving more time to the other processes.

8.2.4 Displaying the processes in order of priority

The command line in Example 8-7 is useful for listing processes by order of the CPU priority. The report displays the PRI column sorted in numerical order. Refer to Chapter 20, “The nice and renice commands” on page 349 for details on priority.

Example 8-7 Displaying the processes in order of priority

```
# ps -eakl | sort -n +6 | head
  F S UID  PID  PPID  C  PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
 303 A  0    0    0 120 16 -- 9012 12          -  3:54 swapper
 303 A  0 1548    0  16 -- d81b 12          - 21:11 lrud
 303 A  0 2580    0  16 -- b036 16 849970 -  4:23 wlmshed
40201 A  0 5420    1  17 20 8130 40          * -  0:00 dog
 303 A  0 2064    0  36 -- 9833 16          -  0:10 netm
 303 A  0 2322    0  37 -- a034 64          * -  1:37 gil
40303 A  0 9306    0  38 -- f27e 152         * -  0:00 j2pg
40303 A  0 7244    0  50 -- 2284 16          -  0:00 jfsz
 303 A  0 1806    0  60 -- 502a 16 35028158 -  0:04 xmgc
```

The report shows that swapper, lrud, and wlmshed have the highest priority.

8.2.5 Displaying the processes in order of nice value

The command line in Example 8-8 is useful for determining processes by order of nice value. The report displays the NI column sorted in numerical order. Refer to Chapter 20, “The nice and renice commands” on page 349 for details on priority. The report displays the NI column sorted in reverse numerical order.

Example 8-8 Displaying the processes in order of nice value

```
# ps -eakl | sort -n +7
  F S UID  PID  PPID  C  PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
 303 A  0    0    0 120 16 -- 9012 12          -  0:28 swapper
 303 A  0  516    0 120 255 -- 9813  8          - 1462:08 wait
 303 A  0  774    0 120 255 -- a014  8          - 1352:04 wait
 303 A  0 1032    0 120 255 -- a815  8          - 1403:23 wait
 303 A  0 1290    0 120 255 -- b016  8          - 1377:28 wait
 303 A  0 1548    0  16 -- d81b 12          -  1:50 lrud
 303 A  0 1806    0  60 -- 502a 16 30066198 -  0:00 xmgc
...(lines omitted)...
40303 A  0 5972    0  38 -- fa7f 152          * -  0:00 j2pg
40001 A  0 3918  4930  0  60 20 91b2  944         -  0:00 dtlogin
...(lines omitted)...
40001 A  0 4930    1  60 20 a995  424         -  0:00 dtlogin
10005 Z  0 29762 27922  1  68 24          0:00 <defunct>
200001 A  0 20804 19502  1  68 24 4b2b  804 30b35fd8 pts/2  2:39 xmtrend
200001 A  0 22226 26070 86 116 24 b6b4  572          pts/10  3:01 dc
```

```

200001 A 0 27922 25812 85 115 24 782e 572 pts/10 4:40 dc
200001 A 0 28904 23776 2 69 24 46ca 268 pts/8 3:14 seen+done
200001 A 0 30446 23776 2 69 24 7ecd 268 pts/8 3:09 seen+done
200001 A 0 30964 23776 3 68 24 66ce 268 pts/8 3:12 seen+done
200001 A 0 31218 23776 3 69 24 96d0 268 pts/8 2:58 seen+done
...(lines omitted)...

```

In the report, the NI values are sometimes displayed as --. This is because the processes do not have a nice value, as they are running at a fixed priority.

Displaying the processes in order of time

The command in Example 8-9 is useful for determining processes by order of CPU time. This is the total accumulated CPU time for the life of the process. The report displays the TIME column sorted in reverse numerical order.

Example 8-9 Displaying the processes in order of time

```

# ps vx | head -1 ; ps vx | grep -v PID | sort -rn +3 | head -10
  PID TTY STAT  TIME PGIN  SIZE  RSS  LIM TSIZ  TRS %CPU %MEM COMMAND
   516 - A   9417:11  0    8 11780  xx   0 11772 24.3  2.0 wait
  1290 - A   9374:49  0    8 11780  xx   0 11772 24.2  2.0 wait
   774 - A   9367:13  0    8 11780  xx   0 11772 24.2  2.0 wait
  1032 - A   9342:08  0    8 11780  xx   0 11772 24.1  2.0 wait
 10836 - A   115:40  106  16 11788 32768 0 11772  0.3  2.0 kbiod
26640 - A   67:26 25972 32 11796 32768 0 11772  1.1  2.0 nfsd
  1548 - A   21:11  0   12 11784  xx   0 11772  0.1  2.0 lrud
  6476 - A   16:18 2870 316  184  xx   2    4  0.0  0.0 /usr/sbin
16262 - A    6:24 4074 1112 1320 32768 1922  724  0.0  0.0 /usr/opt/
  2580 - A    4:33  0   16 11780  xx   0 11772  0.0  2.0 wlmshed

```

The report shows that wait has accumulated the most CPU time. If we were to run our test program called CPU as in Example 8-3 on page 132 which creates a CPU bottleneck, then the wait process would still feature at the top of the report because the test system is normally idle and the wait processes would therefore have accumulated the most time.

8.2.6 Displaying the processes in order of real memory use

Example 8-10 shows the command for determining processes by order of RSS value. (The RSS value is the size of working segment and the code segment combined together in memory in 1 KB units). The report displays the RSS column sorted in reverse numerical order.

Example 8-10 Displaying the processes in order of RSS

```

# ps vx | head -1 ; ps vx | grep -v PID | sort -rn +6 | head -10
  PID  TTY STAT TIME PGIN  SIZE  RSS  LIM TSIZ  TRS %CPU %MEM COMMAND

```

34958	pts/6	A	1:29	20	87976	88004	32768	21	28	0.6	17.0	java	wlmp
9306	-	A	0:00	174	152	11916	32768	0	11772	0.0	2.0	j2pg	
2322	-	A	1:43	0	64	11832	xx	0	11772	0.0	2.0	gil	
26640	-	A	67:26	25972	32	11796	32768	0	11772	1.1	2.0	nfsd	
10580	-	A	0:00	8	20	11792	32768	0	11772	0.0	2.0	rtcmd	
24564	-	A	0:06	1	32	11788	32768	0	11772	0.0	2.0	rpc.lockd	
13418	-	A	0:01	0	16	11788	32768	0	11772	0.0	2.0	PM	
10836	-	A	115:40	106	16	11788	32768	0	11772	0.3	2.0	kbiod	
2064	-	A	0:11	120	16	11788	xx	0	11772	0.0	2.0	netm	
1806	-	A	0:04	12	16	11788	xx	0	11772	0.0	2.0	xmgc	

The report shows that the process `java wlmp` is using the most memory.

Important: Because the values in the `RSS` column contain shared working memory, you cannot add the entries in the `RSS` column for all processes to ascertain the amount of memory used on your system. For example, the `ksh` process can consume about 1 KB of memory and each user can be running at least one `ksh`, but this does not mean that for 300 users logged in, all `ksh` processes will be using a minimum of 300 KB of memory. This is because `ksh` uses share memory, enabling all `ksh` processes to access the same memory. Refer to Chapter 22, “The `ipcs` command” on page 365 for details about memory use.

8.2.7 Displaying the processes in order of I/O

The command in Example 8-11 is useful for determining processes by order of `PGIN` value. `PGIN` represents the number of page ins caused by page faults. Because all AIX I/O is classified as page faults, this value represents the measure of all I/O volume.

The report displays the `PGIN` column sorted in reverse numerical order.

Example 8-11 Displaying the processes in order of PGIN

```
# ps vx | head -1 ; ps vx | grep -v PID | sort -rn +4 | head -10
  PID TTY STAT TIME  PGIN  SIZE  RSS  LIM TSIZ  TRS  %CPU %MEM COMMAND
26640 - A   67:26 25972   32 11796 32768   0 11772  1.1  2.0 nfsd
16262 - A    6:25 4074  1112 1320 32768 1922  724  0.0  0.0 /usr/opt/
 6476 - A   16:19 2870   316  184   xx    2    4  0.0  0.0 /usr/sbin
 5176 - A    3:20 1970  3448   788   xx 2406  196  0.0  0.0 /usr/lpp/
12202 - A    1:00 1394  2152   640 32768  492   44  0.0  0.0 dtwm
15506 - A    0:23 1025 16260  5200 32768   58   48  0.0  1.0 /usr/sbin
 6208 - A    0:40  910  2408   532 32768   99   12  0.0  0.0 /usr/dt/b
 5954 - A    0:05  789  2844   324 32768  179    0  0.0  0.0 /usr/sbin
```

```
16778 - A 0:00 546 724 648 32768 1922 340 0.0 0.0 /usr/opt/  
8290 - A 0:04 420 740 592 32768 75 76 0.0 0.0 /usr/sbin
```

The report shows that the `nfsd` process is producing the most I/O.

8.2.8 Displaying WLM classes

Example 8-12 shows how Workload Manager (WLM) classes can be displayed. In WLM, you can categorize processes into classes. When you run the `ps` command with the `-o class` option, you will see the class displayed.

Example 8-12 Displaying WLM classes

```
# ps -a -o pid,user,class,pcpu,pmem,args  
PID USER CLASS %CPU %MEM COMMAND  
...(lines omitted)...  
20026 root System 0.0 0.0 ps -a -o pid,user,class,pcpu,pmem,arg  
21078 root System 0.0 0.0 wlmstat 1 100  
...(lines omitted)...
```

8.2.9 Viewing threads

The `ps` command enables you to access information relating to the threads running for a particular process. For example, if we wanted to ascertain that particular threads are bound to a CPU, we could use the command in Example 8-13. Threads are bound using the `bindprocessor` command. Refer to 18.2, “`bindprocessor`” on page 292 for more details.

Example 8-14 on page 138 demonstrates how to use `ps` to see if threads are bound to a CPU. As each processor has a wait process that is bound to each active CPU on the system, we will use the wait process as an example.

To check how many CPUs are installed on our system we can use the following command.

Example 8-13 Determining the number of installed processors

```
# lsdev -Cc processor  
proc0 Available 00-00 Processor  
proc1 Available 00-01 Processor  
proc2 Available 00-02 Processor  
proc3 Available 00-03 Processor
```

From the output, we know that there will be four wait processes (assuming all CPUs are enabled). We can determine the Process IDs (PID) of the wait processes using the following command.

Example 8-14 Determining the PID of wait processes

```
# ps vg | head -1 ; ps vg | grep -w wait
```

PID	TTY	STAT	TIME	PGIN	SIZE	RSS	LIM	TSIZ	TRS	%CPU	%MEM	COMMAND
516	-	A	1397:04	0	8 12548	xx	0	12540	21.2	3.0	wait	
774	-	A	1393:52	0	8 12548	xx	0	12540	21.2	3.0	wait	
1032	-	A	1392:39	0	8 12548	xx	0	12540	21.1	3.0	wait	
1290	-	A	1395:14	0	8 12548	xx	0	12540	21.2	3.0	wait	

The output tells us that wait processes PIDs are 516, 774, 1032, and 1290. We can therefore determine whether the threads are actually bound as we would expect by using the command line in Example 8-15.

Example 8-15 Wait processes bound to CPUs

```
# ps -mo THREAD -p 516,774,1032,1290
```

USER	PID	PPID	TID	ST	CP	PRI	SC	WCHAN	F	TT	BND	COMMAN
root	516	0	-	A	120	255	1	-	303	-	0	wait
-	-	-	517	R	120	255	1	-	3000	-	0	-
root	774	0	-	A	120	255	1	-	303	-	1	wait
-	-	-	775	R	120	255	1	-	3000	-	1	-
root	1032	0	-	A	120	255	1	-	303	-	2	wait
-	-	-	1033	R	120	255	1	-	3000	-	2	-
root	1290	0	-	A	120	255	1	-	303	-	3	wait
-	-	-	1291	R	120	255	1	-	3000	-	3	-

The example shows that the wait processes are indeed bound to CPUs. Each of the wait processes has an associated thread. In AIX (starting from version 4), with the exception of init, Process IDs (PIDs) have even numbers, and Threads IDs (TIDs) have odd numbers.

The sar command

The **sar** command is used to gather statistical information about your system — cpu, queuing, paging, file access, and more — that can help determine system performance. The **sar** command can have an impact on system performance.

The **sar** command can be used for:

- ▶ Collecting real-time information
- ▶ Displaying previously captured data
- ▶ Collecting data using **cron**

sar resides in `/usr/sbin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

9.1 sar

The syntax of the **sar** command is:

```
sar [ { -A | [ -a ] [ -b ] [ -c ] [ -d ] [ -k ] [ -m ] [ -q ] [ -r ] [ -u ]  
[ -V ] [ -v ] [ -w ] [ -y ] } ] [ -P ProcessorIdentifier, ... | ALL ]  
[ -ehh [ :mm [ :ss ] ] ] [ -fFile ] [ -iSeconds ] [ -oFile ]  
[ -shh [ :mm [ :ss ] ] ] [ Interval [ Number ] ]
```

Flags

- A** Without the **-P** flag, using the **-A** flag is equivalent to specifying **-abcdkmqruvwy**. When used with the **-P** flag, the **-A** is equivalent to specifying **-acmuw**.
- a** Reports use of file access routines specifying how many times per second several of the file access routines have been called. When used with the **-P** flag, the information is provided for each specified processor. Otherwise it is provided only systemwide.
- b** Reports buffer activity for transfers, accesses, and cache (kernel block buffer cache) hit ratios per second. Access to most files bypasses kernel block buffering and therefore does not generate these statistics. However, if a program opens a block device or a raw character device for I/O, traditional access mechanisms are used, making the generated statistics meaningful.
- c** Reports system calls. When used with the **-P** flag, the information is provided for each specified processor; otherwise, it is provided only systemwide.
- d** Reports activity for each block device.
- e hh[:mm[:ss]]** Sets the ending time of the report. The default ending time is 18:00.
- f File** Extracts records from File (created by **-o File** flag). The default value of the File parameter is the current daily data file, `/var/adm/sa/sadd`.
- i Seconds** Selects data records at intervals as close as possible to the number specified by the Seconds parameter. Otherwise, the **sar** command reports all seconds found in the data file.
- k** Reports kernel process activity.
- m** Reports message (sending and receiving) and semaphore (creating, using, or destroying) activities per second. When

used with the -P flag, the information is provided for each specified processor. Otherwise it is provided only systemwide.

- o File** Saves the readings in the file in binary form. Each reading is in a separate record, and each record contains a tag identifying the time of the reading.
- P ProcessorIdentifier, ... | ALL** Reports per-processor statistics for the specified processor or processors. Specifying the ALL keyword reports statistics for each individual processor, and globally for all processors of the flags that specify the statistics to be reported; only the -a, -c, -m, -u, and -w flags are meaningful with the -P flag.
- q** Reports queue statistics.
- r** Reports paging statistics.
- s hh[:mm[:ss]]** Sets the starting time of the data, causing the **sar** command to extract records time-tagged at, or following, the time specified. The default starting time is 08:00.
- u** Reports per-processor or systemwide statistics. When used with the -P flag, the information is provided for each specified processor; otherwise, it is provided only systemwide. Because the -u flag information is expressed as percentages, the systemwide information is simply the average of each individual processor's statistics. Also, the I/O wait state is defined systemwide and not per processor.
- V** Reads the files created by **sar** on other operating system versions. This flag can only be used with the -f flag.
- v** Reports status of the process, kernel-thread, inode, and file tables.
- w** Reports system switching activity. When used with the -P flag, the information is provided for each specified processor; otherwise, it is provided only systemwide.
- y** Reports tty device activity per second.

9.1.1 Information about measurement and sampling

The **sar** command only formats input generated by the **sadc** command (sar data collector). The **sadc** command acquires statistics mainly from the Perfstat kernel extension (kex) (see 41.1, “Perfstat API” on page 786). The operating system contains a number of counters that are incremented as various system actions occur. The various system counters include:

- ▶ System unit utilization counters
- ▶ Buffer use counters
- ▶ Disk and tape I/O activity counters
- ▶ tty device activity counters
- ▶ Switching and subroutine counters
- ▶ File access counters
- ▶ Queue activity counters
- ▶ Interprocess communication counters

The **sadc** command samples system data a specified number of times at a specified interval measured in seconds. It writes in binary format to the specified output file or to stdout. When neither the measuring interval nor the interval number are specified, a dummy record, which is used at system startup to mark the time when the counter restarts from zero (0), will be written.

9.2 Examples for sar

When starting to look for a potential performance bottleneck, we need to find out more about how the system uses CPU, memory, and I/O. For these resource areas we can use the **sar** command.

9.2.1 Monitoring one CPU at a time

Example 9-1 shows the use of the **sar** command with the **-P** flag.

Example 9-1 Individual CPUs can be monitored separately

```
# sar -P 3 10 3
```

```
AIX 1par05 2 5 00040B1F4C00    04/07/03
```

17:47:52	cpu	%usr	%sys	%wio	%idle
17:48:02	3	16	31	6	47
17:48:12	3	10	19	7	65
17:48:22	3	32	57	4	7
Average	3	20	35	5	40

In this output we ran the **sar** command to display utilization information for the fourth CPU (number 3), three intervals 10 seconds apart. When using the **-P** flag you must specify the CPU number, starting with 0,1,2,3 etc. for CPU 1,2,3,4, respectively.

You can monitor multiple CPUs by specifying the CPU numbers separated by a comma (,), and then followed by the interval and count values as shown in Example 9-2.

Example 9-2 Individual CPUs can be monitored together

```
# sar -P 0,1,2,3 10 2
```

```
AIX lpar05 2 5 00040B1F4C00    04/07/03
```

17:46:33	cpu	%usr	%sys	%wio	%idle
17:46:43	0	29	71	0	0
	1	39	61	0	0
	2	35	65	0	0
	3	36	64	0	0
17:46:53	0	19	51	1	29
	1	45	47	0	8
	2	15	53	1	31
	3	27	40	1	32
17:47:03	0	18	46	1	36
	1	22	43	0	34
	2	20	49	1	30
	3	41	42	1	16
Average	0	22	56	1	22
	1	35	50	0	14
	2	23	56	1	20
	3	35	48	1	16

In the output above you see that the load was fairly evenly spread among the four CPUs. For more information about the **sar** output columns in the example above, see 9.2.11, “Monitoring the processor utilization” on page 158.

When using the **-P** you can also display CPU information for all of the CPUs by using the **ALL** string, as in Example 9-3, where you would normally specify the CPU number(s) to be monitored.

Example 9-3 CPU utilization per CPU or systemwide statistics

```
# sar -P ALL 10 3
```

```
AIX lpar05 2 5 00040B1F4C00    04/07/03
```

```
17:48:33 cpu    %usr    %sys    %wio    %idle
```

17:48:43	0	24	75	0	1
	1	34	64	1	2
	2	37	60	0	2
	3	31	68	0	2
	-	32	66	0	2
17:48:55	0	37	51	1	12
	1	27	39	1	33
	2	19	46	1	35
	3	25	43	1	31
	-	27	45	1	28
17:49:06	0	27	73	0	0
	1	36	64	0	0
	2	33	67	0	0
	3	40	60	0	0
	-	34	66	0	0
Average	0	30	65	0	5
	1	32	55	1	12
	2	29	57	0	13
	3	32	56	0	12
	-	31	58	0	11

As can be seen in Example 9-3 on page 143, the ALL argument was used to display usage information for all of the CPUs, with three intervals of 10 seconds. The last line of each time stamp shows the average CPU usage for all of the CPUs for that time stamp; it is denoted by a dash (-). The last stanza of the output gives the average utilization for each CPU for the duration of the monitoring.

When using the -A flag, `sar` enables most of the report flags to be combined. The -A flag without -P flag is equivalent to using the -abcdkmqruvwy flags. Using the -A and -P flags together is the same as the -acmuw flags. Example 9-4 shows the `sar` command with the abcdkmqruvwy option.

Example 9-4 Using sar -abcdkmqruvwy

```
# sar -abcdkmqruvwy 1
AIX lpar05 2 5 00040B1F4C00 04/07/03
18:24:13 iget/s lookupn/s dirblk/s
bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
ksched/s kproc-ov kexit/s
msg/s sema/s
runq-sz %runocc swpq-sz %swpocc
slots cycle/s fault/s odio/s
%usr %sys %wio %idle
proc-sz inod-sz file-sz thrd-sz
cswch/s
```

```

rawch/s  canch/s  outch/s  rcvin/s  xmtin/s  mdmin/s
18:24:14      8      1585      0
              0      0      0      0      0      0      0
43575      1352      545      313.67      313.67      1753704      95040
              0      0      0
0.00      0.00
1.0      100
117421      0.00      15483.93      8.63
              1      25      0      74
39/262144      279/358278      188/853      47/524288
1943
              0      0      816      0      0      0

```

As can be seen from the example, when using too many flags the output will become more difficult to read. Example 9-5 shows the `sar -A` report, which is similar to the output above but includes the block device I/O report.

Example 9-5 Using sar -A

```

# sar -A 1

AIX lpar05 2 5 00040B1F4C00 04/07/03

18:25:20  iget/s  lookupp/s  dirblk/s
bread/s  lread/s  %rcache  bwrit/s  lwrit/s  %wcache  pread/s  pwrit/s
scall/s  sread/s  swrit/s  fork/s  exec/s  rchar/s  wchar/s
device   %busy    avque    r+w/s    blks/s    await    avserv
ksched/s  kproc-ov  kexit/s
msg/s     sema/s
runq-sz  %runocc  swpq-sz  %swpocc
slots   cycle/s  fault/s  odio/s
%usr    %sys    %wio    %idle
proc-sz  inod-sz  file-sz  thrd-sz
cswch/s
rawch/s  canch/s  outch/s  rcvin/s  xmtin/s  mdmin/s

18:25:21  345      10067      2327
              0      0      0      0      0      0      0
80720      15088      965      402.77      395.38      10975227      612103
hdisk0      0      0.0      1      9      0.0      0.0
hdisk1      0      0.0      0      0      0.0      0.0
hdisk12     5      0.0      65      295      0.0      0.0
hdisk3      5      0.0      64      292      0.0      0.0
hdisk2      5      0.0      65      296      0.0      0.0
hdisk9      0      0.0      0      0      0.0      0.0
hdisk16     5      0.0      64      293      0.0      0.0
hdisk15     5      0.0      65      296      0.0      0.0
hdisk7      5      0.0      64      293      0.0      0.0

```

hdisk8	0	0.0	7	39	0.0	0.0
hdisk4	5	0.0	64	293	0.0	0.0
hdisk17	1	0.0	24	98	0.0	0.0
hdisk11	0	0.0	1	28	0.0	0.0
hdisk6	2	0.0	33	133	0.0	0.0
hdisk14	5	0.0	64	292	0.0	0.0
hdisk5	5	0.0	64	293	0.0	0.0
hdisk13	5	0.0	65	297	0.0	0.0
hdisk10	0	0.0	0	1	0.0	0.0
<hr/>						
2	0	2				
0.00	0.00					
55.0	100					
117353	0.00	21257.27	1545.50			
10	48	2	39			
177/262144	525/358278	517/853	185/524288			
2552						
0	0	920	0	0	0	

9.2.2 Collecting statistics by using cron

You can collect statistical information by editing the *crontab* entries of *sar*. The user *adm*'s *crontab* stores entries for *sar*. The lines that contain scripts *sa1* and *sa2* must be uncommented by deleting the *#* character at the beginning of the line as shown in Example 9-6. Remember to use the ***crontab -e*** command to edit the *crontab* file, as this automatically updates the cron.

Example 9-6 System default crontab entries for the adm user

```
# cronadm cron -l adm
...(lines omitted)...
#-----
#      SYSTEM ACTIVITY REPORTS
# 8am-5pm activity reports every 20 mins during weekdays.
# activity reports every an hour on Saturday and Sunday.
# 6pm-7am activity reports every an hour during weekdays.
# Daily summary prepared at 18:05.
#-----
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1 &
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwyqvm &
```

- ▶ The first line runs the *sa1* command between 8 a.m. and 5 p.m. (17), Monday (1) through Friday (5), for 20 minutes (1200 seconds), three times an hour (3).
- ▶ The second line also runs the *sa1* command, but only on Saturdays (6) and Sundays (0) and then only once every hour.

- ▶ The third line runs the **sa1** command every hour between 6 p.m. (18) and 7 a.m, from Monday (1) through Friday (5).
- ▶ The fourth line runs the **sa2** command every Monday (1) through Friday (5), at five (5) minutes past six (18) p.m .

The **sa1** commands create binary files in the `/var/adm/sa` directory, and the **sa2** command creates an ASCII report in the same directory. The files are named `saDD`, where `DD` stands for day of month, so on the 21st the file name will be `sa21`.

In addition to commenting out the lines in the crontab file for the `adm` user as shown in the previous example, a dummy record must be inserted into the standard system activity daily data file in the `/var/adm/sa` directory at the time of system start by uncommenting the corresponding **sadc** lines in the `/etc/rc` file:

```
/usr/bin/su - adm -c /usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`
```

For 24x7 operations, it is better to just collect the statistical information in binary format, and when needed use **sar** to create reports from the binary files. The following command enables only the statistical collection:

```
0 * * * * /usr/lib/sa/sa1 1200 3 &
```

To create reports from the files created in the `/var/adm/sa` directory, run the **sar** command with the `-f` flag, as shown in Example 9-7.

Example 9-7 Using sar with the -f flag

```
# sar -f /var/adm/sa/sa23

AIX wlmhost 2 5 000BC6AD4C00    04/07/03

00:00:01   %usr   %sys   %wio   %idle
00:20:01     2     1     0     97
00:40:01     2     1     0     97
01:00:01     2     1     0     97
01:20:01     2     1     0     98

Average     2     1     0     97
```

By using the `-s` and `-e` flags with the **sar** command the starting time (`-s`) and ending time (`-e`) can be specified and the report will show the recorded statistics between the starting and ending time only, as shown in Example 9-8.

Example 9-8 Using sar with the -f, -s, and -e flags

```
# sar -f /var/adm/sa/sa23 -s00:00 -e01:00

AIX lpar05 2 5 000BC6AD4C00    04/07/03

00:00:01   %usr   %sys   %wio   %idle
```

00:20:01	2	1	0	97
00:40:01	2	1	0	97
Average	2	1	0	97

The output only reports statistics between 00:00 and 01:00 from the file created on the 23rd of the month.

Note: if collection and analysis of the workload should be performed for more than a month, you need to save the binary statistical collection files from the /var/adm/sa directory elsewhere and rename them with the year and month in addition to the day. The **sa2** command will remove files older than seven days when it is run. The **sa1** command will overwrite existing files with the same day number in the /var/adm/sa directory.

To use a customized **sa1** script that names the binary statistical collection files with year and month instead of only by day, create a script such as the one in Example 9-9 and run it with **cron** instead of the **sa1** command (here called **sa1.custom**).

Example 9-9 The sa1.custom script

```
# expand -4 sa1.custom|nl
1 DATE=`date +%d`
2 NEWDATE=`date +%Y%m%d`
3 ENDIR=/usr/lib/sa
4 DFILE=/var/adm/sa/sa$DATE
5 NEWDFILE=/var/adm/sa/sa$NEWDATE
6 cd $ENDIR
7 if [ $# = 0 ]; then
8     $ENDIR/sadc 1 1 $NEWDFILE
9 else
10    $ENDIR/sadc $* $NEWDFILE
11 fi
12 ln -s $NEWDFILE $DFILE >/dev/null 2>&1
```

The **sa1.custom** script creates files named saYYYYMMDD instead of only saDD. It also creates a symbolic link from the saYYYYMMDD file to a file named saDD. By doing this, other commands that expect to find a saDD file in the /var/adm/sa directory will still do so. These files are also easy to save to a backup server because they can be retrieved by using their filename and thus are unique, and you will not risk losing them if, for example, the backup “class”¹ for these files does not permit enough versions to save the required number of saDD files.

¹ Class in this context refers to a collection of rules and file specifications that specify what, when, and how to back up files.

9.2.3 Displaying access time system routines

Example 9-10 shows the use of the `sar` command with the `-a` flag to display file access system routines.

Example 9-10 Using sar -a

```
# sar -a 10 3

AIX lpar05 2 5 00040B1F4C00 04/07/03

18:06:31  iget/s lookupn/s dirblk/s
18:06:41    1441      22212    6534
18:06:51     412       6415    2902
18:07:01    1353      20375    5268

Average    1072      16377    4913
```

The output shows 1072 calls per second for inode lookup routines, 16377 lookups per second to find a file entry using a pathname (low level file system routine), and 4913 512-byte directory reads per second to find a file name (2.4 MBs read).

The `sar -a` report has the following format:

dirblk/s	Number of 512-byte blocks read per second by the directory search routine to locate a directory entry for a specific file.
iget/s	Calls per second to any of several inode ² lookup routines that support multiple file system types. The <i>iget</i> routines return a pointer to the inode structure of a file or device.
lookupn/s	Calls per second to the directory search routine that finds the address of a vnode ³ given a path name.

Example 9-11 shows how the different CPUs use the file access system routines.

Example 9-11 Using sar -a

```
# sar -aP ALL 10 3

AIX lpar05 2 5 00040B1F4C00 04/07/03

18:07:36 cpu  iget/s lookupn/s dirblk/s
18:07:46  0    29    693    81
          1    56    569   181
```

² An inode is an index node reference number (inode number), which is the file system internal representation of a file. The inode number identifies the file, not the file name.

³ A vnode is either created or used again for every reference made to a file by path name. When a user attempts to open or create a file, if the VFS containing the file already has a vnode representing that file, a use count in the vnode is incremented, and the existing vnode is used. Otherwise, a new vnode is created.

	2	34	667	162
	3	40	604	118
	-	162	2564	554
18:07:56	0	124	1882	556
	1	141	2163	682
	2	137	2257	614
	3	117	2046	546
	-	520	8367	2400
18:08:06	0	77	1090	276
	1	118	1525	383
	2	67	1041	275
	3	68	1229	312
	-	327	4855	1238
Average	0	77	1222	305
	1	105	1419	415
	2	79	1322	350
	3	75	1293	325
	-	336	5261	1397

The last line of each time stamp and the average part of the report show the average for all CPUs. They are denoted by a dash (-).

9.2.4 Monitoring buffer activity for transfers, access, and caching

Example 9-12 shows the use of the `sar` command with the `-b` flag to find out more about buffer activity and utilization.

Example 9-12 Using sar -b

```
# sar -b 10 3

AIX lpar05 2 5 000BC6AD4C00 04/07/03

17:13:18 bread/s 1read/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
17:13:28      1    284    100      0      0      0      0      0
17:13:38      1    283    100      0      0      0      0      0
17:13:48      1    283    100      0      0      0      0      0
Average      1    283    100      0      0      0      0      0
```

In the example, the read cache efficiency is $100 * (283 - 1) / 283$ or 99.64 (approximately 100 percent as shown).

The `sar -b` report has the following format:

bread/s, bwrit/s Reports the number of block I/O operations per second. These I/Os are generally performed by the kernel to

manage the block buffer cache area, as discussed in the description of the `lread/s` and `lwrit/s` values.

`lread/s, lwrit/s` Reports the number of logical I/O requests per second. When a logical read or write to a block device is performed, a logical transfer size of less than a full block size may be requested. The system accesses the physical device units of complete blocks, and buffers these blocks in the kernel buffers that have been set aside for this purpose (the block I/O cache area). This cache area is managed by the kernel, so that multiple logical reads and writes to the block device can access previously buffered data from the cache and require no real I/O to the device. Application read and write requests to the block device are reported statistically as logical reads and writes. The block I/O performed by the kernel to the block device in management of the cache area is reported as block reads and block writes.

`pread/s, pwrit/s` Reports the number of I/O operations on raw devices per second. Requested I/O to raw character devices is not buffered as it is for block devices. The I/O is performed to the device directly.

`%rcache, %wcache` Reports caching effectiveness (cache hit percentage). This percentage is calculated as:
$$100 * (lreads - breads) / lreads$$

9.2.5 Monitoring system calls

Example 9-13 shows the `sar` command being used with the `-c` flag to display system call statistics.

Example 9-13 Using `sar -c`

```
# sar -c 10 3
```

AIX lpar05	1	5	00040B1F4C00	04/07/03															
18:04:30	sca11/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s												
18:04:40	30776	9775	841	95.42	95.22	2626011	1319494												
18:04:50	52742	14190	1667	168.81	168.33	4208049	2781644												
18:05:00	83248	25785	2334	266.34	265.57	6251254	3632468												
Average	55592	16584	1614	176.87	176.39	4362015	2578121												

The output shows that the system did an average of about 177 fork system calls to create new processes. The system also performed 10 times as many read system calls per second than write system calls, but only read 1.7 times more data. This is calculated by dividing the rchar/s by the wchar/s, $4362015 / 2578121 = 1.69$. However the average transfer size for the read system calls was about 260 bytes ($4362015 / 16584 = 263.02$) and the average transfer size for the write system calls was approximately 1600 bytes ($2578121 / 1614 = 1597.34$).

The `sar -c` report has the following format:

exec/s	Reports the total number of exec system calls
fork/s	Reports the total number of fork system calls
sread/s	Reports the total number of read system calls
swrit/s	Reports the total number of write system calls
rchar/s	Reports the total number of characters transferred by read system calls
wchar/s	Reports the total number of characters transferred by write system calls
scall/s	Reports the total number of system calls

To display system call information per CPU, use the `-P` flag with the `-c` flag as in Example 9-14.

Example 9-14 Using sar -c

```
# sar -cPALL 10 3

AIX lpar05 2 5 00040B1F4C00 04/07/03

18:05:05 cpu scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
18:05:15 0 18276 6313 399 53.66 66.50 706060 459381
          1 17527 3898 474 64.65 58.89 2714377 827040
          2 16942 3211 578 63.17 56.56 1196307 1042826
          3 18786 4589 507 66.83 66.13 1161274 870102
          - 71420 17914 1965 248.22 247.64 5856025 3217526
18:05:25 0 23951 11926 601 42.08 56.63 1933828 796038
          1 18926 4965 658 62.67 61.39 1341808 1133376
          2 19277 6462 545 67.51 61.37 1094918 833909
          3 21008 6788 640 75.40 68.80 1164796 951693
          - 83076 30068 2440 248.44 248.24 5515219 3709782
18:05:35 0 20795 7932 426 55.35 67.92 805022 500341
          1 21442 4615 718 62.14 67.39 2425419 1292411
          2 17528 3939 504 77.17 66.21 1122280 873226
          3 18983 3310 648 78.04 71.46 1375725 1176041
          - 78535 19718 2286 273.23 272.84 5692685 3823642
```

Average	0	21006	8722	475	50.36	63.69	1148012	585170
	1	19299	4493	617	63.15	62.56	2159898	1084462
	2	17917	4539	542	69.28	61.38	1137801	916592
	3	19591	4894	598	73.43	68.80	1234002	999354
	-	77676	22566	2230	256.63	256.24	5688003	3583571

As before, the last line of each time stamp displays the average, and the last stanza displays the average of all of the data collected per CPU.

9.2.6 Monitoring activity for each block device

Example 9-15 shows the use of the `sar` command with the `-d` flag. This will display information about each block device with the exception of tape drives.

Example 9-15 Using sar -d

```
# sar -d 10 3

AIX bolshoi 1 5 00040B1F4C00 05/20/01

17:58:08 device %busy avque r+w/s blks/s await avserv
17:58:18 hdisk0 0 0.0 1 8 0.0 0.0
hdisk1 0 0.0 0 0 0.0 0.0
hdisk12 3 0.0 39 176 0.0 0.0
hdisk3 3 0.0 38 174 0.0 0.0
hdisk2 3 0.0 39 177 0.0 0.0
hdisk9 0 0.0 0 0 0.0 0.0
hdisk16 3 0.0 38 175 0.0 0.0
hdisk15 3 0.0 38 176 0.0 0.0
hdisk7 3 0.0 38 175 0.0 0.0
hdisk8 0 0.0 4 23 0.0 0.0
hdisk4 3 0.0 38 174 0.0 0.0
hdisk17 1 0.0 14 58 0.0 0.0
hdisk11 0 0.0 1 16 0.0 0.0
hdisk6 1 0.0 19 79 0.0 0.0
hdisk14 3 0.0 38 174 0.0 0.0
hdisk5 3 0.0 38 175 0.0 0.0
hdisk13 3 0.0 39 177 0.0 0.0
hdisk10 0 0.0 0 0 0.0 0.0

...(lines omitted)...

Average hdisk0 0 0.0 1 10 0.0 0.0
hdisk1 0 0.0 0 0 0.0 0.0
hdisk12 15 0.0 171 764 0.0 0.0
hdisk3 15 0.0 160 721 0.0 0.0
hdisk2 15 0.0 168 750 0.0 0.0
```

hdisk9	0	0.0	0	0	0.0	0.0
hdisk16	16	0.0	169	759	0.0	0.0
hdisk15	15	0.0	170	760	0.0	0.0
hdisk7	16	0.0	168	753	0.0	0.0
hdisk8	0	0.0	10	62	0.0	0.0
hdisk4	15	0.0	167	753	0.0	0.0
hdisk17	5	0.0	61	247	0.0	0.0
hdisk11	0	0.0	4	74	0.0	0.0
hdisk6	7	0.0	89	358	0.0	0.0
hdisk14	15	0.0	168	752	0.0	0.0
hdisk5	15	0.0	167	750	0.0	0.0
hdisk13	15	0.0	171	767	0.0	0.0
hdisk10	0	0.0	0	1	0.0	0.0

Notice that the output displays block device activity for every specified time interval. The last column, as before, displays the averages.

The **sar -d** report has the following format:

%busy	The portion of time the device was busy servicing a transfer request
avque	The average number of requests in the queue
r+w/s	Number of read and write requests per second
blks/s	Number of bytes transferred in 512-byte blocks per second
await	The average time each request waits in the queue before it is serviced
avserv	The average time taken for servicing a request

9.2.7 Monitoring kernel process activity

Example 9-16 shows the use of the **sar** command with the **-k** flag to find out more about kernel process activity.

Example 9-16 Using sar -k

```
# sar -k 10 3

AIX 1par05 2 5 000BC6AD4C00 04/07/03

22:57:45 ksched/s kproc-ov kexit/s
22:57:55 0 0 0
22:58:05 0 0 0
22:58:15 0 0 0

Average 0 0 0
```

The **sar -k** report has the following format:

kexit/s	Reports the number of kernel processes terminating per second.
kproc-ov/s	Reports the number of times kernel processes could not be created because of enforcement of process threshold limit per second.
ksched/s	Reports the number of kernel processes assigned to tasks per second.

A kernel process (kproc) exists only in the kernel protection domain. It is created using the creatp and initp kernel services.

9.2.8 Monitoring the message and semaphore activities

Example 9-17 uses the **sar** command with the **-m** flag to display message and semaphore utilization.

Example 9-17 Using sar -m

```
# sar -m 10 3

AIX lpar05 2 5 000BC6AD4C00 04/07/03

17:03:45 msg/s sema/s
17:03:50 0.00 2744.71
17:03:55 0.00 2748.94
17:04:00 0.00 2749.15

Average 0.00 2747.60
```

Message queues and semaphores are used by processes to communicate with each other. See 22.1, “ipcs” on page 366 for more information about working with and managing Inter Process Communication (IPC) resources using the **ipcs** command.

The **sar -m** report has the following format:

msg/s	The number of IPC message primitives per second.
sema/s	The number of IPC semaphore primitives per second.

Using the **-P** flag with the **sar -m** flag displays message queue and semaphore information per CPU.

Example 9-18 Using sar -m

```
# sar -mPALL 10 3

AIX lpar05 2 5 000BC6AD4C00 04/07/03
```

17:04:49	cpu	msg/s	sema/s
17:04:54	0	0.00	638.17
	1	0.00	706.14
	2	0.00	712.38
	3	0.00	694.84
	-	0.00	2746.03
17:04:59	0	0.00	639.11
	1	0.00	708.95
	2	0.00	712.35
	3	0.00	699.20
	-	0.00	2754.35
17:05:04	0	0.00	640.93
	1	0.00	704.97
	2	0.00	710.16
	3	0.00	689.15
	-	0.00	2739.90
<hr/>			
Average	0	0.00	639.40
	1	0.00	706.68
	2	0.00	711.63
	3	0.00	694.39
	-	0.00	2746.76

As before, the last line of each interval displays the average for that interval, denoted with a dash (-), and the last column displays the average for the collection period.

9.2.9 Monitoring the kernel scheduling queue statistics

Example 9-19 shows the use of the `sar` command with the `-q` flag to find out more about kernel scheduling queues:

Example 9-19 Using sar -q

```
# sar -q 10 3
AIX lpar05 2 5 00040B1F4C00 04/07/03
18:02:02 runq-sz %runocc swpq-sz %swpocc
18:02:12 23.8 100 2.9 70
18:02:22 35.0 100 8.0 100
18:02:32 13.0 100 3.0 30
Average 23.9 97 5.5 65
```

The output tells us that the run queue had approximately 24 threads ready to run (`runq-sz`), on average, and was occupied 97 percent of the time (`% runocc`).

If the system is idle the output would appear as in Example 9-20.

Example 9-20 Using sar -q

```
# sar -q 2 4

AIX lpar05 2 5 00040B1F4C00 04/07/03

16:44:35 runq-sz %runocc swpq-sz %swpocc
16:44:37
16:44:39
16:44:41
16:44:43

Average
```

A blank value in any column indicates that the associated queue is empty.

The **sar -q** report has the following format:

runq-sz	The average number of kernel threads in the run queue (the r column reported by vmstat is the actual value)
%runocc	The percentage of the time that the run queue is occupied
swpq-sz	The average number of kernel threads waiting for resources or I/O (the b column reported by vmstat is the actual value)
%swpocc	The percentage of the time the swap queue is occupied

9.2.10 Monitoring the paging statistics

Example 9-21 uses the the **sar** command with the -r flag will display paging statistics.

Example 9-21 Using sar -r

```
# sar -r 10 3

AIX lpar05 2 5 00040B1F4C00 04/07/03

17:57:16 slots cycle/s fault/s odio/s
17:57:26 117419 0.00 15898.29 2087.03
17:57:36 117419 0.00 6051.20 1858.52
17:57:46 117419 0.00 13186.44 1220.44

Average 117419 0 11718 1722
```

The output shows that there was approximately 460 MB of free space on the paging spaces in the system ($117419 * 4096 / 1024 / 1024 = 458$) during our measurement interval.

The `sar -r` report has the following format:

cycle/s	Reports the number of page replacement cycles per second (equivalent to the <code>cy</code> column reported by <code>vmstat</code>).
fault/s	Reports the number of page faults per second. This is not a count of page faults that generate I/O because some page faults can be resolved without I/O.
slots	Reports the number of free 4096-byte pages on the paging spaces.
odio/s	Reports the number of non-paging disk I/Os per second.

9.2.11 Monitoring the processor utilization

Example 9-22 uses the `sar` command with the `-u` to display processor utilization.

Example 9-22 Using sar -u

```
# sar -u 10 3
```

AIX lpar05 2 5 00040B1F4C00 04/07/03

17:54:58	%usr	%sys	%wio	%idle
17:55:08	30	57	1	12
17:55:18	29	57	1	12
17:55:28	26	43	1	29
Average	29	53	1	18

The output shows that the system spent 29% in user mode, 53% in system mode, and 1% waiting for IO requests, and was idle 18% of the time.

The output in Example 9-23 displays per-CPU utilization using the `-u` and `-P` flags.

Example 9-23 Using sar -u

```
# sar -uPALL 10 3
```

AIX lpar05 2 5 00040B1F4C00 04/07/03

17:55:49	cpu	%usr	%sys	%wio	%idle
17:55:59	0	38	51	1	9
	1	27	43	1	29
	2	24	45	1	31
	3	25	46	0	29

	-	28	46	1	25
17:56:09	0	26	74	0	0
	1	40	60	0	0
	2	33	67	0	0
	3	40	60	0	0
	-	35	65	0	0
17:56:19	0	12	38	1	50
	1	18	37	1	44
	2	56	33	1	10
	3	13	37	1	48
	-	26	36	1	37
Average	0	25	54	1	20
	1	28	47	1	24
	2	38	48	0	14
	3	26	48	0	26
	-	30	49	1	21

The last line of each time stamp and the average part of the report show the average for all CPUs; they are denoted by a dash (-). The output above shows that the system load was fairly evenly distributed among the CPUs.

The `sar -u` report has the following format:

<code>%idle</code>	Reports the percentage of time the CPU(s) were idle with no outstanding disk I/O requests (equivalent to the <code>id</code> column reported by <code>vmstat</code>).
<code>%sys</code>	Reports the percentage of time the CPU(s) spent in execution at the system (or kernel) level (equivalent to the <code>sy</code> column reported by <code>vmstat</code>).
<code>%usr</code>	Reports the percentage of time the CPU(s) spent in execution at the user (or application) level (equivalent to the <code>us</code> column reported by <code>vmstat</code>).
<code>%wio</code>	Reports the percentage of time the CPU(s) were idle during which the system had outstanding disk/NFS I/O request(s). Equivalent to the <code>wa</code> column reported by <code>vmstat</code> .

9.2.12 Monitoring tty device activity

Using the **sar** command with the **-y** flag displays information about tty device utilization, as shown in Example 9-24.

Example 9-24 Using sar -y

```
# sar -y 10 3

AIX lpar05 2 5 000BC6AD4C00    04/07/03

23:01:17 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
23:01:27      2      0     51      0      0      0
23:01:37      1      0    446      0      0      0
23:01:47      1      0    360      0      0      0

Average      2      0    286      0      0      0
```

The output above shows that this system only wrote, on average, 286 characters to terminal devices during our measurement interval. Terminal devices can be directly attached through the tty devices (`/dev/tty`) or through PTY device drivers (`/dev/pty` or `/dev/pts`) for network connections with terminal emulation.

The **sar -y** report has the following format:

canch/s	Reports tty canonical input queue characters per second. This field is always zero (0) for AIX Version 4 and later.
mdmin/s	Reports tty modem interrupts per second.
outch/s	Reports tty output queue characters per second (similar to the <code>tout</code> column, but per second, reported by iostat).
rawch/s	Reports tty input queue characters per second (similar to the <code>tin</code> column, but per second, reported by iostat).
rcvin/s	Reports tty receive interrupts per second.
xmtin/s	Reports tty transmit interrupts per second.

9.2.13 Monitoring kernel tables

Using the **sar** command with the **-v** flag displays kernel table utilization, as shown in Example 9-25.

Example 9-25 Using sar -v

```
# sar -v 10 3

AIX lpar05 2 5 00040B1F4C00    04/07/03

17:52:58  proc-sz    inod-sz    file-sz    thrd-sz
```

17:53:08	248/262144	641/358248	709/853	256/524288
17:53:18	227/262144	632/358248	642/853	235/524288
17:53:28	42/262144	282/358248	192/853	50/524288

The **sar -v** report has the following format:

file-sz	Reports the number of entries in the kernel file table. The column is divided into two parts:
file-size	The number of open files in the system (the currently used size of the file entry table). Note that a file may be open multiple times (multiple file opens for one inode).
file-size-max	The maximum number of files that have been open since IPL (high watermark).
	The file entry table is allocated dynamically, so the file-size-max value signifies a file entry table with file-size-max entries available, and only file-size entries used.
inod-sz	Reports the number of entries in the kernel inode table. The column is divided into two parts:
inode-size	The current number of active (open) inodes.
inode-size-max	The maximum number of inodes allowed. This value is calculated at system boot time based on the amount of memory in the system
proc-sz	Reports the number of entries in the kernel process table. The column is divided into two parts:
proc-size	The current number of processes running on the system.
proc-size-max	The maximum number of processes allowed. Maximum value depends on whether it is a 32-bit or 64-bit system (NPROC).
thrd-sz	Reports the number of entries in the kernel thread table. The column is divided into two parts:
thread-size	The current number of active threads.
thread-size-max	The maximum number of threads allowed. Maximum value depends

on whether it is a 32-bit or 64-bit system (NTHREAD).

The current limits for some of the kernel tables (per process) can be found using the shell built in function `ulimit`, as shown in Example 9-26.

Example 9-26 Using ulimit

```
# ulimit -a
time(seconds)      unlimited
file(blocks)       2097151
data(kbytes)       131072
stack(kbytes)      32768
memory(kbytes)     32768
coredump(blocks)   2097151
nofiles(descriptors) 2000
```

9.2.14 Monitoring system context switching activity

Using the `sar` command with the `-w` flag displays information about context switching between threads. Context switching happens when a multi-process operating system stops running one process or thread and starts another. By default, CPU time is allocated to threads in 10 ms chunks.

Example 9-27 Using sar -w

```
# sar -w 10 3

AIX 1par05 2 5 000BC6AD4C00 04/07/03

23:00:46 cswch/s
23:00:56      516
23:01:06      599
23:01:16      307

Average      474
```

The output shows that there were 474 context switches per second on average during the measurement interval.

The `sar -w` report has the following format:

<code>cswch/s</code>	Reports the number of context switches per second (equivalent to the <code>cs</code> column reported by <code>vmstat</code>).
----------------------	--

Using the -P flag with the -w flag displays the number of context switches per second for the different CPUs as shown in Example 9-28.

Example 9-28 Using sar -w

```
# sar -wP ALL 10 3

AIX lpar05 2 5 000BC6AD4C00 04/07/03

23:04:18 cpu cswch/s
23:04:28 0 212
          1 140
          2 152
          3 125
          - 625
23:04:38 0 186
          1 119
          2 111
          3 82
          - 494
23:04:48 0 66
          1 60
          2 52
          3 30
          - 210

Average 0 154
          1 106
          2 106
          3 79
          - 443
```

The last line of each time stamp is denoted by a dash (-) and the last stanza displays the averages.

The schedo and schedtune commands

The **schedtune** program is being phased out and will not be supported in future releases. It is being replaced by the **schedo** command. In AIX 5.2, a compatibility script calling **schedtune** is provided to help the transition.

The **schedo** and **schedtune** commands can only be executed by root to manage CPU scheduler tunable parameters.

These commands set or display current or next boot values for all scheduler tuning parameters. The **schedo** command can also make permanent changes or defer changes until the next reboot. Whether the command sets or displays a parameter is determined by the accompanying flag. The **-o** flag performs both actions. It can either display the value of a parameter or set a new value for a parameter.

The **schedo** resides in `/usr/bin/schedo` and is part of the `bos.perf.tune` fileset. The **schedtune** resides in `/usr/samples/kernel` and is part of the `bos.adt.samples` fileset. Both are installable from the AIX base installation media.

Attention: Incorrect changes of scheduling parameters can cause performance degradation or operating-system failure. Refer to *AIX 5L Version 5.2 Performance Management Guide* before using these tools.

10.1 schedo

The schedo command can be invoked with the following syntaxes:

```
schedo [ -p | -r ] { -o tunable[=Newvalue] | -d tunable | -D | -a }  
schedo {-? | -h [tunable]}
```

Flags

- h tunable** Displays help about the tunable parameter.
- d tunable** Resets tunable to its default value. If a tunable is currently not set to its default value, and -r is not used in combination, it will not be changed but a warning is displayed.
- o tunable** Displays the value or sets tunable to a new value.
- a** Displays the current, reboot (when used in conjunction with -r) or permanent (when used in conjunction with -p) value for all tunable parameters, one per line in pairs Tunable=Value. For the permanent option, a value is only displayed for a parameter if its reboot and current values are equal. Otherwise NONE displays as the value.
- D** Resets all tunables to their default value.
- p** Makes changes apply to both current and reboot values, when used in combination with -o, -d, or -D; that is, turns on the updating of the /etc/tunables/nextboot file in addition to the updating of the current value.
- r** Makes changes apply to reboot values when used in combination with -o, -d, or -D; that is, turns on the updating of the /etc/tunables/nextboot file.
- L [tunable]** Lists the characteristics of one or all tunables, one per line. It specifies the current value, default value, next reboot value, minimum and maximum values, and its unit and type. The current set of parameters managed by **schedo** only includes Dynamic types.
- x [tunable]** Generates tunable characteristics in a comma-separated value for loading into a spreadsheet.

AIX uses different scheduler policies:

SCHED_OTHER This is the default AIX scheduling policy. This scheduler policy only applies to threads with a non-fixed priority. A threads priority is recalculated after each clock interrupt.

SCHED_RR This policy only applies to threads running with a fixed priority. Threads are time sliced. Once the time slice

expires, the thread is moved to the back of the queue of threads of this same priority.

SCHED_FIFO	This scheduler policy applies to fixed priority threads owned by the root user only. A thread runs until completion unless blocked or unless it gives up the CPU voluntarily.
SCHED_FIFO2	This scheduler policy enables a thread that sleeps for a short period of time to resume at the head of the queue rather than the tail of the queue. The length of time the thread sleeps is determined by the <code>schedo -o affinity_lim</code> .
SCHED_FIFO3	With this scheduler policy, whenever a thread becomes runnable, it moves to the head of its run queue.

10.1.1 Recommendations and precautions

The following section provides suggestions and precautions when using the `schedo` command.

Important: The `schedo` command should be used with caution. The use of inappropriate values can seriously impair the performance of the system. Always keep a record of the current value settings before making changes.

- ▶ Setting the CPU decay factor `sched_D` to a low value will force the current effective priority value of the process down. A CPU-intensive process therefore will achieve more CPU time at the expense of the interactive process types. When the `sched_D` is set high, then CPU-intensive processes are less favored because the priority value will decay less the longer it runs. The interactive type processes will be favored in this case. It is therefore important to understand the nature of the processes that are running on the system before adjusting this value.
- ▶ When the value of `sched_R` is set high, the nice value, as set by the nice command, has less effect on the process, which means that CPU-intensive processes that have been running for some time will have a lower priority than interactive processes.
- ▶ The smaller the value of `v_repage_hi`, the closer to thrashing the system gets before process suspension starts. Conversely, if the value is set too high, processes may become suspended needlessly.
- ▶ It is not recommended that the `v_min_process` value is set lower than 2 (two). Even though this is permitted, the result is that only one user processes will be permitted when suspension starts.

- ▶ Setting the value of the `v_sec_wait` high results in unnecessarily poor response times from suspended processes. The system's processors could be idle while the suspended processes wait for the delay set by the `v_sec_wait`. Ultimately, this will result in poor performance.

10.2 Examples for schedo

This section presents a collection of sample usages of the `schedo` command.

10.2.1 Displaying current settings

The `schedo -L` command displays the current, default, and reboot settings as shown in Example 10-1.

Example 10-1 Using schedo to display the current, default, and reboot values

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
# schedo -L							
DEPENDENCIES							
v_repage_hi	0	0	0	0	2047M		D
v_repage_proc	4	4	4	0	2047M		D
v_sec_wait	1	1	1	0	2047M	seconds	D
v_min_process	2	2	2	0	2047M	processes	D
v_exempt_secs	2	2	2	0	2047M	seconds	D
pacefork	10	10	10	10	2047M	clock ticks	D
sched_D	16	16	16	0	32		D
sched_R	16	16	16	0	32		D
timeslice	1	1	1	0	2047M	clock ticks	D
maxspin	16K	16K	16K	1	4095M	spins	D
%usDelta	100	100	100	0	100		D
affinity_lim	7	7	7	0	100	dispatches	D
idle_migration_barrier	4	4	4	0	100	sixteenth	D
fixed_pri_global	0	0	0	0	1	boolean	D

n/a means parameter not supported by the current platform or kernel

Parameter types:

S = Static: cannot be changed
D = Dynamic: can be freely changed
B = Bosboot: can only be changed using bosboot and reboot
R = Reboot: can only be changed during reboot
C = Connect: changes are only effective for future socket connections
M = Mount: changes are only effective for future mountings
I = Incremental: can only be incremented

Value conventions:

K = Kilo: 2^{10} G = Giga: 2^{30} P = Peta: 2^{50}
M = Mega: 2^{20} T = Tera: 2^{40} E = Exa: 2^{60}

10.2.2 Tuning CPU parameters

This example deals with tuning the **schedo** parameters that affect the CPU. In Example 10-2, the **schedo** parameters that have an effect on the CPU are shown in bold.

Example 10-2 The schedo CPU flags

```
#schedo -a
    v_repage_hi = 0
    v_repage_proc = 4
    v_sec_wait = 1
    v_min_process = 2
    v_exempt_secs = 2
    pacefork = 10
    sched_D = 16
    sched_R = 16
    timeslice = 1
    maxspin = 16384
    %usDelta = 100
    affinity_lim = 7
idle_migration_barrier = 4
    fixed_pri_global = 0
```

In order to correctly tune the **schedo** parameters, it is necessary to understand the nature of the workload that is running on the system, such as whether the processes are CPU-intensive or interactive.

Thread prioritizing and aging

The priority of most user processes varies with the amount of CPU time the process has used recently. The CPU scheduler's priority calculations are based on two parameters that are set with **schedo**: `sched_R` and `sched_D`. The `sched_R` and `sched_D` values are in 1/32 seconds. Both `r` (`sched_R` parameter) and `d` (`sched_D` parameter) have default values of 16.

See 1.2.2, "Processes and threads" on page 6 for a detailed usage of these parameters in calculating priority and aging.

Time slice

The default time slice is one clock tick. One clock tick equates to 10 ms. The time slice value can be changed using **schedo -o timeslice=...**. Context switching sometimes decreases by setting `timeslice` to a higher value.

Fixed priority threads

The **schedo** command can be used to force all fixed priority threads to be placed on the global run queue. The global run queue is examined for runnable threads before the individual processor's run queues are examined. A thread that is on the global run queue will be dispatched to a CPU prior to threads on the CPU's run queue when that CPU becomes available if that thread has a better priority than the threads on the CPU's local run queue. The syntax for this is **schedo -o fixed_pri_global=1**.

Fork retries

The **schedo -o pacefork** command displays the number of clock ticks to delay before retrying a failed `fork()` call. If a `fork()` subroutine fails due to a lack of paging space, then the system will wait until the specified number of clock ticks have elapsed before retrying. The default value is 10. Because the duration of one clock tick is 10 ms, the system will wait 100 ms by default.

Lock tuning

When a thread needs to acquire a lock, if that lock is held by another thread on another CPU, then the thread will spin on the lock for a length of time before it goes to sleep and puts itself on an event run queue waiting for the lock to be released. The value of the `maxspin` from **schedo -o maxspin** command determines how many iterations the thread will check the lock word to see if the lock is available. On SMP systems, this value is defaulted to 16384 as in the next example. In the case of an upgrade to a faster system, it should be realized that the duration for spinning on a lock will be less than on a slower system for the same `maxspin` value. The spin on a lock parameter can be changed using the **schedo -o maxspin=...** command.

10.2.3 Tuning memory parameters

This section deals with the **schedo** command values that affect memory, which are highlighted in Example 10-3.

Example 10-3 The schedo command's memory-related parameters

```
# schedo -a
    v_repage_hi = 0
    v_repage_proc = 4
      v_sec_wait = 1
    v_min_process = 2
    v_exempt_secs = 2
      pacefork = 10
      sched_D = 16
      sched_R = 16
      timeslice = 1
      maxspin = 16384
      %usDelta = 100
      affinity_lim = 7
idle_migration_barrier = 4
fixed_pri_global = 0
```

The load control mechanism is used to suspend processes when the available memory is overcommitted. Pages are stolen as they are needed from Least Recently Used (LRU) pages. Pages from the suspended processes are the most likely to be stolen. The intention of memory load control is to smooth out infrequent peaks in memory demand to minimize the chance of thrashing taking place. It is *not* intended as a mechanism to cure systems that have inadequate memory. Certain processes are exempt from being suspended, such as kernel processes and processes with a fixed priority below 60.

Thrashing

Using the output of the **vmstat** command as referenced in Chapter 13, “The **vmstat** command” on page 211, the system is said to be thrashing when:

```
po/fr > 1/h
```

where:

po	Number of page writes
fr	Number of page steals
h	The schedo -o v_repage_hi value

Note: On systems with a memory size greater than 128 MB, the size of the **schedo -o v_repage_hi** value by default is 0 (zero). On systems where the memory is less than 128 MB, the default value is set to 6 (six). When **v_repage_hi** is set to 0 (zero), then the load control mechanism is disabled.

On a server with 128 MB of memory or less with the default settings, the system is thrashing when the ratio of page writes to page steals is greater than one to six. The value of **h** in the equation above, which can be changed by the **schedo -o v_repage_hi=...**, therefore has the function of determining at which point the system is said to be thrashing.

If the algorithm detects that memory is overcommitted, then the values associated with the **v_min_process**, **v_repage_proc**, **v_sec_wait**, and **v_exempt_secs** are used. If the load control mechanism is disabled, then these values are ignored.

v_min_process This value sets the minimum number of active processes that are exempt from suspension. Active processes are defined as those that are runnable and waiting for page I/O. Suspended processes and processes waiting for events are not considered active processes. The default value is 2 (two). To increase this value defeats the object of the load control mechanism's ability to suspend processes. To decrease this value means that fewer processes are active when the mechanism starts suspending processes. In large systems, setting this value above the default may result in better performance.

v_repage_proc This value sets the per-process criterion used to determine which processes to suspend depending on the rate of thrashing of each individual process. The default value is set to four and implies that the process can be suspended when the ratio of repages to page faults is greater than four.

v_sec_wait This value sets the time delay until the process can become active again after the system is no longer thrashing. The default value is 1 (one) second. Setting this value high will result in an unnecessarily poor response time from suspended processes.

v_exempt_secs This value is used to exempt a recently suspended process from being suspended again for a period of time. The default value is 2 (two) seconds.

When the CPU penalty factor **sched_R** is large, the nice value assigned to a thread has less effect. When the CPU penalty factor is small, the nice value assigned to the thread has more effect. This is shown in the following example. In Example 10-4 on page 173, the **sched_R** value is set to 4 (four). The nice value has a low impact on the value of the current effective priority, as can be seen in Table 10-1 on page 173.

Example 10-4 sets CPU penalty factor to 4 using the **schedo** command.

Example 10-4 CPU penalty factor of four using the schedo command

```
# schedo -o sched_R=4
Setting sched_R to 4
# schedo -a
    v_repage_hi = 0
    v_repage_proc = 4
    v_sec_wait = 1
    v_min_process = 2
    v_exempt_secs = 2
    pacefork = 10
    sched_D = 16
    sched_R = 4
    timeslice = 1
    maxspin = 16384
    %usDelta = 100
    affinity_lim = 7
idle_migration_barrier = 4
fixed_pri_global = 0
```

The result of changing the sched_R value to 4 (four) is tabulated in Table 10-1. Values are obtained from this calculation:

$$\begin{aligned} cp &= bp + nv + (C * r/32) \\ &= 40 + 20 + (100 * 4/32) \\ &= 72 \end{aligned}$$

Table 10-1 Current effective priority calculated where sched_R is four

Time	Current effective priority	sched_R	Clock ticks consumed (count)
0 (initial value)	60	4	0
10 ms	60	4	1
20 ms	60	4	2
30 ms	60	4	3
40 ms	60	4	4
1000 ms	72	4	100

In Example 10-5, the sched_R is set to 16; the nice value has less effect on the current effective priority of the thread as can be seen in Table 10-2. The CPU penalty factor is set to 16 using the **schedo** command.

Example 10-5 CPU penalty factor of 16 using the schedo command

```
# schedo -o sched_R=16
Setting sched_R to 16
# schedo -a
    v_repage_hi = 0
    v_repage_proc = 4
    v_sec_wait = 1
    v_min_process = 2
    v_exempt_secs = 2
    pacefork = 10
    sched_D = 16
    sched_R = 16
    timeslice = 1
    maxspin = 1638
    %usDelta = 100
    affinity_lim = 7
idle_migration_barrier = 4
fixed_pri_global = 0
```

With the default value of 16, the current effective priority will be as in Table 10-2. Values are obtained from this calculation:

$$\begin{aligned}
 cp &= bp + nv + (C * r/32) \\
 &= 40 + 20 + (100 * 16/32) \\
 &= 110
 \end{aligned}$$

Table 10-2 Current effective priority calculated where sched_R is 16

Time	Current effective priority	sched_R	Clock ticks consumed (count)
0	60	16	0
10 ms	60	16	1
20 ms	61	16	2
30 ms	61	16	3
40 ms	62	16	4
1000 ms	110	16	100

Even though the calculation allows for the priority value to exceed 126, the kernel will cap it at this value.

In the next example, the effect of the CPU decay factor can be seen. In Table 10-3, the swapper wakes up at 1000 ms and sets the value of CPU use count to 50. The current effective priority is significantly affected by the CPU decay factor.

$$\begin{aligned}
 C_{new} &= C * d/32 \\
 &= 100 * 16/32 \\
 &= 50
 \end{aligned}$$

Table 10-3 The CPU decay factor using the default sched_D value of 16

Time	Current effective priority	sched_R	Clock ticks consumed (count)	sched_D
990 ms	72	4	99	16
1000 ms	72	4	100	16
1010 ms	66	4	50	16
1020 ms	67	4	60	16

When the sched_D value is set to 31 as in Table 10-4, then the impact of the CPU decay factor has less effect on the current effective priority value. With the decay factor set in this way, interactive-type threads are favored over CPU-intensive threads.

$$\begin{aligned}
 C_{new} &= C * d/32 \\
 &= 100 * 31/32 \\
 &= 97
 \end{aligned}$$

Table 10-4 The CPU decay factor using a sched_D value of 31

Time	Current effective priority	sched_R	Clock ticks consumed (count)	sched_D
990 ms	72	4	99	31
1000 ms	72	4	100	31
1010 ms	72	4	96	31
1020 ms	72	4	97	31

The changes made using the **schedo** command will be lost on a reboot, unless the -p or -r flag is used to preserved the values for reboot.

Example 10-7 uses the **schedo -o maxspin=n** command to improve system performance where there is lock contention. If there is inode lock contention on, for example, database files within a logical volume, this can be reduced by an increase in the maxspin value, provided that CPU use is not too high. Faster CPUs spin on a lock for a shorter period of time than slower CPUs because of maxspin is used up more quickly.

As can be seen in Example 10-6, the default value for spin on a lock is 16384 on SMP systems. This value is usually too low, and it should be set about four times the default value. Run the command in the example below to increase the value. Example 10-7 shows the **schedo** output after the change. Example 10-6 shows maxspin's default value before setting.

Example 10-6 Default maxspin value displayed by schedo

```
# schedo -a
  v_repage_hi = 0
  v_repage_proc = 4
    v_sec_wait = 1
  v_min_process = 2
  v_exempt_secs = 2
    pacefork = 10
      sched_D = 16
      sched_R = 16
    timeslice = 1
      maxspin = 16384
    %usDelta = 100
  affinity_lim = 7
idle_migration_barrier = 4
  fixed_pri_global = 0
idle_migration_barrier = 4
  fixed_pri_global = 0
```

Example 10-7 shows how to change maxspin with **schedo** command.

Example 10-7 The new maxspin value set by schedo -o maxspin=n command

```
# schedo -o maxspin=65536
Setting maxspin to 65536
# schedo -a
  v_repage_hi = 0
  v_repage_proc = 4
    v_sec_wait = 1
  v_min_process = 2
  v_exempt_secs = 2
    pacefork = 10
      sched_D = 16
      sched_R = 16
    timeslice = 1
```

```
maxspin = 65536
%usDelta = 100
affinity_lim = 7
idle_migration_barrier = 4
fixed_pri_global = 0
```

10.3 schedtune

Note: This command in AIX 5L Version 5.2 is just a sample compatibility script that calls the **schedo** command.

The syntax of the **schedtune** command is:

```
schedtune [-D] | [-h n][-p n][-w n][-m n][-e n][-f n][-r n][-d n][-t n]
[-s n][-c n][-a n][-b n][-F n]
```

If no flags are specified, **schedo -a** is called to display the current values. Otherwise the following flags apply:

- D** Restores the default values.
- h n** Calls **schedo -o v_repage_hi=n** to change the systemwide criterion used to determine when process suspension begins and ends.
- p n** Calls **schedo -o v_repage_proc=n** to change the per-process criterion used to determine which processes to suspend.
- w n** Calls **schedo -o v_sec_wait=n** to set the number of seconds to wait after thrashing ends before adding processes back into the mix.
- m n** Calls **schedo -o v_min_process=n** to set the minimum multiprogramming level.
- e n** Calls **schedo -o v_exempt_seconds=n** to set the time until a recently suspended and resumed process is eligible for re-suspension.
- f n** Calls **schedo -o pacefork=n** to set the number of clock ticks to delay before retrying a failed fork call.
- r n** Calls **schedo -o sched_R=n** to set the rate at which to accumulate CPU usage.

- d n** Calls `schedo -o sched_D=n` to set the factor used to decay CPU usage.
- t n** Calls `schedo -o timeslice=n` to set the number of 10ms time slices.
- s n** Calls `schedo -o maxspin=n` to set the number of times to spin on a lock before sleeping.
- c n** Calls `schedo -o %usDelta=n` to control the adjustment of the clock drift.
- a n** Calls `schedo -o affinity_lim=n` to set the number of context.
- b n** Calls `schedo -o idle_migration_barrier=n` to set the idle migration barrier.
- F n** Calls `schedo -o fixed_pri_global=n` to keep fixed priority threads in the global run queue.
- ?** Displays a brief description of the command and its parameters.

The topas command

The **topas** command is a performance monitoring tool that is ideal for broad spectrum performance analysis. The command is capable of reporting on local system statistics such as CPU use, CPU events and queues, memory and paging use, disk performance, network performance, and NFS statistics. It can report on the top hot processes of the system as well as on Workload Manager (WLM) hot classes. The WLM class information is only displayed when WLM is active. The **topas** command defines hot processes as those processes that use a large amount of CPU time. The **topas** command does not have an option for logging information. All information is real time.

Note: In order to obtain a meaningful output from the **topas** command, the screen or graphics window must support a minimum of 80 characters by 24 lines. If the display is smaller than this, then parts of the output become illegible.

The **topas** command requires the peragent.tools filesset to be installed on the system. The **topas** command resides in /usr/bin and is part of the bos.perf.tools filesset that is obtained from the AIX base installable media.

11.1 topas

The syntax of the **topas** command is as follows.

```
topas    [ -d number_of_monitored_hot_disks ]  
         [ -h show help information ]  
         [ -i monitoring_interval_in_seconds ]  
         [ -n number_of_monitored_hot_network_interfaces ]  
         [ -p number_of_monitored_hot_processes ]  
         [ -w number_of_monitored_hot_WLM_classes ]  
         [ -c number_of_monitored_hot_CPUs ]  
         [ -P show full-screen process display ]  
         [ -W show full-screen WLM display ]
```

Flags

- d** Specifies the number of disks to be displayed and monitored. The default value of two is used by the command if this value is omitted from the command line. In order that no disk information is displayed, the value of zero must be used. If the number of disks selected by this flag exceeds the number of physical disks in the system, then only the physically present disks will be displayed. Because of the limited space available, only the number of disks that fit into the display window are shown. The disks by default are listed in descending order of kilobytes read and written per second KBPS. This can be changed by moving the cursor to an alternate disk heading (for example, Busy%).
- h** Used to display the topas help.
- i** Sets the data collection interval and is given in seconds. The default value is two.
- n** Used to set the number of network interfaces to be monitored. The default is two. The number of interfaces that can be displayed is determined by the available display area. No network interface information will be displayed if the value is set to zero.
- p** Used to display the top hot processes on the system. The default value of 20 is used if the flag is omitted from the command line. To omit top process information from the displayed output, the value of this flag must be set to zero. If there is no requirement to determine the top hot processes on the system, then this flag should be set to zero as this function is the main contributor of the total overhead of the topas command on the system.
- w** Specifies the number of WLM classes to be monitored. The default value of two is assumed if this value is omitted. The classes are displayed as display space permits. If this value is set to zero, then no information about WLM classes will be displayed. If the WLM daemons

are not active on the system, then this flag may be omitted. Setting this flag to a value greater than the number of available WLM classes results in only the available classes being displayed.

- P** Used to display the top hot processes on the system in greater detail than is displayed with the **-p** flag. Any of the columns can be used to determine the order of the list of processes. To change the order, simply move the cursor to the appropriate heading.
- W** Splits the full screen display. The top half of the display shows the top hot WLM classes in detail, and the lower half of the screen displays the top hot processes of the top hot WLM class.

11.1.1 Information about measurement and sampling

The **topas** command makes use of the System Performance Measurement Interface (SPMI) Application Program Interface (API) for obtaining its information. By using the SPMI API, the system overhead is kept to a minimum. The **topas** command uses the perfstat library call to access the perfstat kernel extensions.

In instances where the **topas** command determines values for system calls, CPU clicks, and context switches, the appropriate counter is incremented by the kernel and the mean value is determined over the interval period set by the **-i** flag. Other values such as free memory are merely snapshots at the interval time.

The sample interval can be selected by the user by using the **-i** flag option. If this flag is omitted in the command line, then the default of two seconds is used.

11.2 Examples for topas

In this section, we discuss some usage examples of the **topas** command.

11.2.1 Common uses of the topas command

Example 11-1 shows the standard **topas** command and its output. The system host name is displayed on the left hand side on the top line of the screen. The line below shows the time and date as well as the sample interval used for measurement.

Example 11-1 The default topas display

```
# topas
Topas Monitor for host:  lpar05          EVENTS/QUEUES  FILE/TTY
Mon Apr  7 14:01:03 2003  Interval:  2    Cswitch    236  Readch      0
Kernel  1.2  |          | Syscall    343  Writech    102
              |          | Reads      0    Rawin      0
```

```

User      25.1 |#####| Writes      1 Ttyout      0
Wait      0.0 |      | Forks       0 Igets       0
Idle      73.6 |#####| Execs       0 Namei      31
          |      | Runqueue   3.0 Dirblk     0
Network  KBPS  I-Pack  O-Pack  KB-In  KB-Out  Waitqueue  0.0
en0       0.5    3.5    1.5    0.4    0.1
lo0       0.0    0.0    0.0    0.0    0.0
          |      | PAGING     MEMORY
          |      | Faults    0 Real,MB   8191
          |      | Steals    0 % Comp    9.1
Disk      Busy%   KBPS     TPS  KB-Read  KB-Writ  Pgspln    0 % Noncomp  2.3
hdisk0    0.0    0.0    0.0    0.0    0.0 Pgspln    0 % Client   0.5
hdisk1    0.0    0.0    0.0    0.0    0.0 PageIn     0
          |      | PageOut   0 PAGING SPACE
WLM-Class (Active)  CPU%   Mem%  Disk-I/O%  Sios     0 Size,MB   2048
http      25     0     0          |      | % Used    9.1
java      0     2     0          |      | NFS (calls/sec) % Free   90.8
          |      | ServerV2   0
Name      PID  CPU%  PgSp  Class  ClientV2  0 Press:
sh        23728 24.9  0.3  http  ServerV3  0 "h" for help
java     15712 0.3   0.2  java  ClientV3  0 "q" to quit
topas    25764 0.1   1.4  System
rmcd     19870 0.1   2.1  System

```

CPU utilization statistics

CPU utilization is graphically and numerically displayed below the date and time and is split up into a percentage of idle, wait, user, and kernel time:

- Idle time** The percentage of time when the processor is performing no tasks.
- Wait time** The percentage of time when the CPU is waiting for the response of an input output device such as a disk or network adapter.
- User time** The percentage of time when the CPU is executing a program in user mode.
- Kernel time** The percentage of time when the CPU is running in kernel mode.

Network interface statistics

The following network statistics are available over the monitoring period:

- Network** The name of the interface adapter.
- KBPS** Reports total throughput of the interface in kilobytes per second.
- I-Pack** Reports the number of packets received per second.
- O-Pack** Reports the number of packets sent per second.
- KB-In** Reports the number of kilobytes received per second.
- KB-Out** Reports the number of kilobytes sent per second.

Disk drive statistics

The following disk drive statistics are available:

Disk	The name of the disk drive.
Busy%	Reports the percentage of time that the disk drive was active.
KBPS	Reports the total throughput of the disk in kilobytes per second. This value is the sum of KB-Read and KB-Writ.
TPS	Reports the number of transfers per second or I/O requests to a disk drive.
KB-Read	Reports the number of kilobytes read per second.
KB-Writ	Reports the number of kilobytes written per second.

WLM statistics

The following WLM statistics are available:

WLM-Class	The name of the WLM class.
CPU%	The average CPU utilization of the WLM class over the monitoring interval
Mem%	The average memory utilization of the WLM class over the monitoring interval
Disk-I/O%	The average percent of disk I/O of the WLM class over the monitoring interval

Process statistics

The top hot processes are displayed with the following headings:

Name	The name of the process. Where the number of characters in the process name exceeds nine, the name will be truncated. No pathname details for the process are displayed.
PID	Shows the process identification number for the process. This is useful when a process needs to be stopped.
CPU%	Reports on the CPU time utilized by this process.
PgSp	Reports on the paging space allocated to this process.
Owner	Displays the owner of the process.

Event and queue statistics

This part of the report is on the top right-hand side of the **topas** display screen. It reports on select system global events and queues over the sampling interval:

Cswit ch	Reports the number of context switches per second.
----------	--

Syscall	Reports the total number of system calls per second.
Reads	Reports the number of read system calls per second.
Writes	Reports the number of write system calls per second.
Forks	Reports the number of fork system calls per second.
Exec	Reports the number of exec system calls per second.
Runqueue	Reports the average number of threads that were ready to run, but were waiting for a processor to become available.
Waitqueue	Reports the average number of threads waiting for paging to complete.

File and tty statistics

The file and tty part of the **topas** screen is located on the extreme right-hand side at the top. The reported items are listed below.

Readch	Reports the number of bytes read through the read system call per second.
Writch	Reports the number of bytes written through the write system call per second.
Rawin	Reports the number of bytes read in from a tty device per second.
Ttyout	Reports the number of bytes written to a tty device per second.
Igets	Reports the number of calls per second to the inode lookup routines.
Namei	Reports the number of calls per second to the path lookup routine.
Dirblk	Reports on the number of directory blocks scanned per second by the directory search routine.

Paging statistics

There are two parts of the paging statistics reported by **topas**. The first part is total paging statistics. This simply reports the total amount of paging available on the system and the percentages free and used. The second part provides a breakdown of the paging activity. The reported items and their meanings are listed below.

Faults	Reports the number of faults.
Steals	Reports the number of 4 KB pages of memory stolen by the Virtual Memory Manager per second.
PgspIn	Reports the number of 4 KB pages read in from the paging space per second.
PgspOut	Reports the number of 4 KB pages written to the paging space per second.

PageIn	Reports the number of 4 KB pages read per second.
PageOut	Reports the number of 4 KB pages written per second.
Sios	Reports the number of input/output requests per second issued by the Virtual Memory Manager.

Memory statistics

The memory statistics are listed below.

Real	Shows the actual physical memory of the system in megabytes.
%Comp	Reports real memory allocated to computational pages.
%Noncomp	Reports real memory allocated to non-computational pages.
%Client	Reports on the amount of memory that is currently used to cache remotely mounted files.

NFS statistics

Statistics for client and server calls per second are displayed.

11.2.2 Using subcommands

When the **topas** screen is displayed, these subcommands and their functions are available:

- a Always reverts to the default **topas** screen (Example 11-1 on page 181).
- c Toggles CPU display between off, cumulative, and busiest CPU.
- d Toggles disk display between off, total disk activity, and busiest disks.
- f When the cursor is over a WLM class name, this option shows the top processes of this class in the WLM window.
- h Provides online help.
- n Toggles network display between off, cumulative, and busiest interface.
- p Toggles the top hot process list on and off.
- P Toggles between the full top process screen, which is the same as the -P option from the **topas** command line. The top 20 processes are displayed showing the following information:

USER	User name
PID	Process identification
PPID	Parent process identification
PRI	Priority given to the process

NI Nice value for the process
 TIME Accumulative CPU time
 CPU% Percentage of time that the CPU has been busy with this process during the sample period
 COMMAND The name of the process

The full processor screen is shown in Example 11-2.

Example 11-2 The full process topas screen

```
Topas Monitor for host:  lpar05      Interval:  2   Mon Apr  7 15:21:13 2003
```

USER	PID	PPID	PRI	NI	RES	TEXT	PAGE	TIME	CPU%	I/O	OTH	COMMAND
root	12082	34750	255	24	32	7	32	5:25	57.5	0	0	java
root	15480	34750	255	24	32	7	32	5:41	57.0	0	0	java
root	4902	0	16	41	4	0	5	0:57	0.0	0	0	lrud
root	5160	0	60	41	4	0	4	0:08	0.0	0	0	xmgc
root	5418	0	36	41	4	0	4	0:02	0.0	0	0	netm
root	5676	0	37	41	14	0	17	7:31	0.0	0	0	gil
root	5934	0	16	41	2	0	4	0:02	0.0	0	0	wlmsched
root	6502	0	50	41	2	0	4	0:00	0.0	0	0	jfsz
root	6678	11198	60	20	58	66	81	0:00	0.0	0	0	snmpmibd
root	7132	1	60	20	8	2	8	0:00	0.0	0	0	uprintfd
root	7266	0	60	20	2	0	4	0:00	0.0	0	0	kbiod
root	7498	0	60	20	2	0	4	0:00	0.0	0	0	lvmbb
root	7748	0	60	20	5	0	5	0:00	0.0	0	0	rtcnd
nobody	8346	16434	60	20	91	82	749	0:00	0.0	0	0	httpd
db2as	8516	1	60	20	413	52	419	0:21	0.0	0	0	db2dasrrm
root	8798	23410	60	20	90	56	103	0:00	0.0	0	0	ksh
root	9118	1	60	20	2	0	4	0:00	0.0	0	0	random
root	9268	11198	60	20	164	10	297	0:00	0.0	0	0	portmap
root	9916	11198	60	20	5	0	16	0:00	0.0	0	0	srcd
root	10178	36810	60	20	91	18	91	0:00	0.0	0	0	telnetd

- q** This option is used to exit the **topas** performance tool.
- r** This option is used to refresh the screen.
- w** This option toggles the WLM section of the display on and off.
- W** This option toggles the full WLM display on and off; see Example 11-3.

Example 11-3 Typical display from using the W subcommand

```
Topas Monitor for host:  lpar05      Interval:  2   Mon Apr  7 15:23:38 2003
```

WLM-Class (Active)	CPU%	Mem%	Disk-I/O%
java	46	26	0
http	0	0	0
System	0	4	0

Shared	0	0	0
Default	0	0	0
Unmanaged	0	3	0
Unclassified	0	1	0

```

=====
USER      PID    PPID  PRI  NI   DATA  TEXT  PAGE      TIME CPU%  I/O  OTH  COMMAND
root     15480  34750 207 24    32     7    32     7:53 99.8   0    0  java
root     12082  34750 225 24    32     7    32     7:36 99.8   0    0  java
root     21914  14920  58 41   465    12   465     0:01 0.5    0    0  topas
root      5160     0  60 41     4     0    4     0:08 0.0    0    0  xmgc
root      5418     0  36 41     4     0    4     0:02 0.0    0    0  netm
root      5676     0  37 41    14     0   17     7:31 0.0    0    0  gil
root      5934     0  16 41     2     0    4     0:02 0.0    0    0  wlmsched
root      6502     0  50 41     2     0    4     0:00 0.0    0    0  jfsz
root      6678  11198  60 20    58    66   81     0:00 0.0    0    0  snmpmibd
root      7132     1  60 20     8     2    8     0:00 0.0    0    0  uprintfd
root      7266     0  60 20     2     0    4     0:00 0.0    0    0  kbiod
=====

```

11.2.3 Monitoring CPU usage

Some common uses of the **topas** command are given in Example 11-4.

Example 11-4 Excessive CPU % user use indicated by topas

```

=====
Topas Monitor for host:  lpar05          EVENTS/QUEUES  FILE/TTY
Mon Apr  7 15:29:36 2003  Interval:  2    Cswitch      235  Readch      5952
                               Syscall      477  Writech      794
Kernel  0.1 |                               Reads         7  Rawin        0
User    99.8 | #####                               Writes         1  Ttyout       0
Wait    0.0 |                               Forks          0  Igets        0
Idle    0.0 |                               Execs          0  Namei        51
                               Runqueue      5.0  Dirblk       0
Network KBPS  I-Pack  O-Pack  KB-In  KB-Out  Waitqueue  0.0
en0      1.2    6.9    0.9    0.4    0.8
lo0      0.0    0.0    0.0    0.0    0.0
Disk  Busy%  KBPS    TPS  KB-Read  KB-Writ  PAGING  MEMORY
hdisk0  0.0    0.0    0.0    0.0    0.0    Faults   26  Real,MB    8191
hdisk1  0.0    0.0    0.0    0.0    0.0    Steals   0  % Comp     31.9
                               PgspIn   0  % Noncomp  2.3
                               PgspOut  0  % Client   0.5
                               PageIn   0
WLM-Class (Active)  CPU%  Mem%  Disk-I/O%  PageOut  0  PAGING SPACE
java                0     0     0          Sios     0  Size,MB    2048
http                0     0     0          % Used   9.2
                               NFS (calls/sec) % Free   90.7
Name                PID CPU% PgSp Class  ServerV2  0
sh                   27920 25.0 0.3 http  ClientV2  0  Press:
sh                   23770 25.0 0.4 Default  ServerV3  0  "h" for help
=====

```

```

java      12082 25.0 0.1 java      ClientV3      0  "q" to quit
java      15480 24.9 0.1 java

```

In Example 11-4 on page 187, it can be seen that the CPU percentage use is excessively high. This typically indicates that one or more processes are hogging CPU time. The next step to analyzing the problem would be to press the **P** subcommand key for a full list of top hot processes. Example 11-5 shows this output.

Example 11-5 Full process display screen shows processes hogging CPU time

```

Topas Monitor for host:  lpar05      Interval:  2  Mon Apr  7 15:36:11 2003

```

USER	PID	PPID	PRI	NI	DATA RES	TEXT RES	PAGE SPACE	TIME	CPU%	I/O	OTH	COMMAND
root	12082	34750	240	24	32	7	32	19:10	99.7	0	0	java
root	15480	34750	231	24	32	7	32	19:27	99.7	0	0	java
nobody	23770	21454	71	24	24	56	101	7:06	99.7	0	0	sh
nobody	27922	21454	71	24	24	56	111	0:40	99.7	0	0	sh
root	5418	0	36	41	4	0	4	0:02	0.0	0	0	netm
root	5676	0	37	41	14	0	17	7:31	0.0	0	0	gil
root	5934	0	16	41	2	0	4	0:02	0.0	0	0	wlmsched
root	6502	0	50	41	2	0	4	0:00	0.0	0	0	jfsz
root	6678	11198	60	20	58	66	81	0:00	0.0	0	0	snmpmibd
root	7132	1	60	20	8	2	8	0:00	0.0	0	0	uprintfd
root	7266	0	60	20	2	0	4	0:00	0.0	0	0	kbiod
root	7498	0	60	20	2	0	4	0:00	0.0	0	0	lvmbb
root	7748	0	60	20	5	0	5	0:00	0.0	0	0	rtcmd
nobody	8346	16434	60	20	91	82	749	0:00	0.0	0	0	httpd
db2as	8516	1	60	20	413	52	419	0:21	0.0	0	0	db2dasrrm
root	8798	23410	60	20	90	56	103	0:00	0.0	0	0	ksh
root	9118	1	60	20	2	0	4	0:00	0.0	0	0	random
root	9268	11198	60	20	164	10	297	0:00	0.0	0	0	portmap
root	9916	11198	60	20	5	0	16	0:00	0.0	0	0	srcd
root	10178	36810	60	20	91	18	91	0:00	0.0	0	0	telnetd

The first four processes are responsible for maximum CPU use. These four processes could also be seen on the default **topas** display.

Example 11-6 on page 189 shows **topas** CPU statistics obtained on a server with two CPUs and 68 GB of real memory. As can be seen, the CPU wait time is consistently high. This indicates that the CPU is spending a large amount of time waiting for an I/O operation to complete. This could indicate such problems as insufficient available real memory space resulting in excessive paging, or even a hardware problem on a disk. Further investigation is required to determine exactly where the problem is.

Example 11-6 *topas* used to initially diagnose the source of a bottleneck

```

Kernel  12.2  |###
User    9.3   |###
Wait   30.3 |#####
Idle    48.0  |##### 41

```

The **topas** command can be regarded as the starting point to resolving most performance problems. As an example, it might be useful to check the amount of paging activity on the system. The **topas** command also provides hard disk and network adapter statistics that can be useful for finding I/O bottlenecks. These **topas** statistics should be examined to determine whether a single disk or adapter is responsible for the abnormally high CPU wait time.

11.2.4 Monitoring disk problem

In Example 11-7, **topas** is used to monitor a system. The CPU percentage wait is more than 23 percent and has consistently been at this level or higher. `hdisk1` is close to 100 percent busy and has a high transfer rate. The other disks on the system are not at all busy. If this condition persisted, this scenario might suggest that a better distribution of data across the disks is required. It is recommended, however, to investigate further using a tool such as **filemon**, which is covered in Chapter 25, “The `filemon` command” on page 457.

Example 11-7 *Monitoring disk problems with topas*

```

Topas Monitor for host:  lpar05          EVENTS/QUEUES  FILE/TTY
Mon Apr  7 16:37:57 2003  Interval:  2    Cswitch      353  Readch  3498.0K
                               Syscall      2154  Writech 3498.1K
Kernel  1.2  |          | Reads        874  Rawin    0
User    0.1  |          | Writes       875  Ttyout   0
Wait    23.8 |##### | Forks         0  Igets    0
Idle    74.7 |##### | Execs         0  Namei    37
                               Runqueue     0.0  Dirblk   0
                               Waitqueue    1.0
Network KBPS  I-Pack  O-Pack  KB-In  KB-Out
en0      0.4    6.0    0.5    0.3    0.1
lo0      0.0    0.0    0.0    0.0    0.0
Disk     Busy%   KBPS    TPS   KB-Read  KB-Writ
hdisk1   99.7   6991.9  224.1 3502.0  3489.9
hdisk0   0.0    0.0    0.0   0.0    0.0
PAGING
Faults   111  Real,MB  8191
Steals   0   % Comp   9.5
PgspIn   0   % Noncomp 5.7
PgspOut  0   % Client  0.5
PageIn   875
PageOut  874  PAGING SPACE
WLM-Class (Active)  CPU%   Mem%  Disk-I/O%  Sios      383  Size,MB  2048
System              1     8     32          % Used   9.1
java                 0     2     0          % Free  90.8
                               NFS (calls/sec)
Name                PID CPU% PgSp Class  ServerV2  0
rw                  25846 1.1 0.0 System ClientV2  0  Press:

```

topas	21978	0.0	1.4	System	ServerV3	0	"h" for help
java	15712	0.0	0.2	java	ClientV3	0	"q" to quit
java	32414	0.0	0.4	java			

The truss command

The **truss** command tracks a process's system calls, received signals, and incurred machine faults. The application to be examined is either specified on the command line of the **truss** command, or **truss** can be attached to one or more already running processes.

The executable for **truss** resides in `/usr/bin` and is part of the `bos.sysmgt.serv_aid` fileset, which is installable from the AIX base installation media.

12.1 truss

The syntax of the **truss** command is:

```
truss [ -f ] [ -c ] [ -a ] [ -e ] [ -i ] [ { -t | -x } [!] Syscall [...] ]
      [ -s [!] Signal [...] ] [ -m [!] Fault [...] ]
      [ { -r | -w } [!] file descriptor [...] ]
      [ -o Outfile ] { Command | -p pid [ . . . ] }
truss [ -f ] [ -c ] [ -a ] [ -i ] [ -d ] [ -D ] [ -e ] [ -i ]
      [ { -t | -x } [!] Syscall [...] ] [ -s [!] Signal [...] ]
      [ { -m } [!] Fault [...] ] [ { -r | -w } [!] FileDescriptor [...] ]
      [ { -u } [!] LibraryName [...] :: [!] FunctionName [ ... ] ]
      [ -o Outfile ] { Command | -p pid [ . . . ] }
```

Flags

- a** Displays the parameter strings that are passed in each executed system call.
- c** Counts tracked system calls, faults, and signals rather than displaying the results line by line. A summary report is produced after the tracked command terminates or when **truss** is interrupted. If the **-f** flag is also used, the counts include all tracked Syscalls, Faults, and Signals for child processes.
- d** A time stamp will be included with each line of output. Time displayed is in seconds relative to the beginning of the trace. The first line of the trace output will show the base time from which the individual time stamps are measured. By default time stamps are not displayed.
- D** Delta time, displayed on each line of output, represents the time elapsed since the last reported event for the LWP.
- e** Displays the environment strings that are passed in each executed system call.
- f** Follows all children created by the **fork** system call and includes their signals, faults, and system calls in the output. Normally only the first-level command or process is tracked. When the **-f** flag is specified, the process ID is included with each line of output to show which process executed the system call or received the signal.
- i** Keeps interruptible sleeping system calls from being displayed. Certain system calls on terminal devices or pipes, such as **open** and **kread**, can sleep for indefinite periods and are interruptible. Normally, **truss** reports such sleeping system calls if they remain asleep for more

than one second. The system call is then reported a second time when it completes. The `-i` flag causes such system calls to be reported only upon completion.

- l** Displays the thread ID of the responsible LWP process along with `truss` output. By default LWP ID is not displayed in the output.
- m [!] Fault** Machine faults to track or exclude. Listed machine faults must be separated from each other by a comma. Faults may be specified by name or number (see the `sys/procfs.h` header file or Table 12-1 on page 194). If the list begins with the `"!"` symbol, the specified faults are excluded from being displayed with the output. The default is `-mall`.
- o Outfile** Designates the file to be used for the output. By default, the output goes to standard error.
- p** Interprets the parameters to `truss` as a list of process IDs (PIDs) of existing processes rather than as a command to be executed. `truss` takes control of each process and begins tracing it, provided that the user ID and group ID of the process match those of the user, or that the user is a privileged user.
- r [!] file descriptor** Displays the full contents of the I/O buffer for each read on any of the specified file descriptors. The output is formatted 32 bytes per line, and shows each byte either as an ASCII character (preceded by one blank) or as a two-character C language escape sequence for control characters, such as horizontal tab (`\t`) and newline (`\n`). If ASCII interpretation is not possible, the byte is shown in two-character hexadecimal representation. The first 16 bytes of the I/O buffer for each tracked read are shown, even in the absence of the `-r` flag. The default is `-r!all`.
- s [!] Signal** Permits listing Signals to examine or exclude. Those signals specified in a list (separated by a comma) are tracked. The output reports the receipt of each specified signal even if the signal is being ignored, but not blocked, by the process. Blocked signals are not received until the process releases them. Signals may be specified by name or number (see `sys/signal.h` or Table 12-2 on page 195). If the list begins with the `"!"` symbol, the listed signals are excluded from being displayed with the output. The default is `-s all`.

- t [!] Syscall** Includes or excludes system calls from the tracked process. System calls to be tracked must be specified in a list and separated by commas. If the list begins with an "!" symbol, the specified system calls are excluded from the output. The default is -tall.
- u** Traces dynamically loaded user level function calls from user libraries. The LibraryName and FunctionName are comma-separated lists that can include name-matching metacharacters *, ?, [] with the same meanings as interpreted by the shell but as applied to the library or function name spaces, and not to files.
- w [!] file descriptor** Displays the contents of the I/O buffer for each write on any of the listed file descriptors (see -r for more details). The default is -w!all.
- x [!] Syscall** Displays data from the specified parameters of tracked system calls in raw format, usually hexadecimal rather than symbolically. The default is -x!all.

The -m flag enables tracking of machine faults. Machine fault numbers are analogous to signal numbers. These correspond to hardware faults. Table 12-1 describes the numbers or names to use with the -m flag to specify machine faults. This information was be extracted from the /usr/include/sys/procfs.h (default location) file.

Table 12-1 Machine faults

Symbolic fault name	Fault ID	Fault description
FLTILL	1	Illegal instruction
FLTPRIV	2	Privileged instruction
FLTBPT	3	Breakpoint instruction
FLTTRACE	4	Trace trap (single-step)
FLTACCESS	5	Memory access (for example alignment)
FLTBOUNDS	6	Memory bounds (invalid address)
FLTIOVF	7	Integer overflow
FLTIZDIV	8	Integer zero divide
FLTFPE	9	Floating-point exception
FLTSTACK	10	Unrecoverable stack fault

Symbolic fault name	Fault ID	Fault description
FLTPAGE	11	Recoverable page fault (no signal)

Table 12-2 describes the numbers or names to use with the `-s` flag to specify signals. This list can also be accessed in the `/usr/include/sys/signal.h` file.

Table 12-2 *Signals*

Symbolic signal name	Signal ID	Signal description
SIGHUP	1	Hangup, generated when terminal disconnects
SIGINT	2	Interrupt, generated from terminal special char
SIGQUIT	3	Quit, generated from terminal special char
SIGILL	4	Illegal instruction (not reset when caught)
SIGTRAP	5	Trace trap
SIGABRT	6	Abort process
SIGEMT	7	EMT instruction
SIGFPE	8	Floating point exception
SIGKILL	9	Kill
SIGBUS	10	Bus error (specification exception)
SIGSEGV	11	Segmentation violation
SIGSYS	12	Bad argument to system call
SIGPIPE	13	Write on a pipe with no one to read it
SIGALRM	14	Alarm clock timeout
SIGTERM	15	Software termination signal
SIGURG	16	Urgent condition on I/O channel
SIGSTOP	17	Stop
SIGTSTP	18	Interactive stop
SIGCONT	19	Continue
SIGCHLD	20	Sent to parent on child stop or exit
SIGTTIN	21	Background read attempted from control terminal

Symbolic signal name	Signal ID	Signal description
SIGTTOU	22	Background write attempted to control terminal
SIGIO	23	I/O possible, or completed
SIGXCPU	24	CPU time limit exceeded
SIGXFSZ	25	File size limit exceeded
SIGMSG	27	Input data is in the ring buffer
SIGWINCH	28	Window size changed
SIGPWR	29	Power-fail restart
SIGUSR1	30	User defined signal 1
SIGUSR2	31	User defined signal 2
SIGPROF	32	Profiling time alarm
SIGDANGER	33	System crash imminent; free up some page space
SIGVTALRM	34	Virtual time alarm
SIGMIGRATE	35	Migrate process
SIGPRE	36	Programming exception
SIGVIRT	37	AIX virtual time alarm
SIGALRM1	38	m:n condition variables
SIGWAITING	39	m:n scheduling
SIGCPUFAIL	59	Predictive de-configuration of processors
SIGKAP	60	Keep alive poll from native keyboard
SIGGRANT	SIGKAP	Monitor mode granted
SIGRETRACT	61	Monitor mode should be relinquished
SIGSOUND	62	Sound control has completed
SIGSAK	63	Secure attention key
SIGIOINT	SIGURG	Printer to backend error signal
SIGAIO	SIGIO	Base LAN I/O
SIGPTY	SIGIO	PTY I/O

Symbolic signal name	Signal ID	Signal description
SIGIOT	SIGABRT	Abort (terminate) process
SIGCLD	SIGCHLD	Old death of child signal
SIGLOST	SIGIOT	Old BSD signal
SIGPOLL	SIGIO	Another I/O event

12.1.1 Information about measurement and sampling

The **truss** command executes a specified command, or attaches to listed process IDs, and produces a report of the system calls, received signals, and machine faults a process incurs. Each line of the output report is either the *Fault* or *Signal* name, or the *Syscall* name with parameters and return values.

The subroutines defined in system libraries are not necessarily the exact system calls made to the kernel. The **truss** command does not report these subroutines but, rather, the underlying system calls they make. When possible, system call parameters are displayed symbolically using definitions from relevant system header files. For path name pointer parameters, **truss** displays the string being pointed to. By default, undefined system calls are displayed with their name, all eight possible arguments, and the return value in hexadecimal format.

The command **truss** retrieves a lot of the information about processes from the `/proc` filesystem. The `/proc` filesystem is a pseudo device that will return information from the kernel structures depending on the structure of the files that are read. For more information see 1.7, “The `/proc` file system” on page 46 and Chapter 16, “Process-related commands” on page 267.

12.2 Examples for truss

The **truss** command can generate large amounts of output, so you should reduce the number of system calls you are tracing or attach **truss** to a running process only for a limited amount of time.

12.2.1 Using truss

One way to use **truss** is to start by checking the general application flow, then use a summary output as provided with the `-c` flag. To pinpoint the most important system calls in the application flow, indicate these specifically with the `-t` flag. Example 12-1 on page 198 shows the flow of using the **date** command.

Example 12-1 Using truss with the date command

```
# truss date
execve("/usr/bin/date", 0x2FF22BF8, 0x2FF22C00)  argc: 1
kiocntl(1, 22528, 0x00000000, 0x00000000)      = 0
Tue Apr  8 11:42:45 CDT 2003
kwrite(1, 0xF01B5168, 29)                       = 29
kfcntl(1, F_GETFL, 0x00000000)                 = 2
kfcntl(2, F_GETFL, 0xF01B5168)                 = 2
_exit(0)
```

We can see that after the program has been loaded and the initial setup has been performed, the **date** program's use of subroutines gets translated into `kiocntl` for the collection of the current time, and the display of the date uses a `kwrite` system call.

12.2.2 Using the summary output

In the following example we ran `dd` and used `truss` to do a summary report about what `dd` is doing when it reads and writes. This is especially interesting because `dd` splits itself with the `fork` system call and has a child process. First we use the `-c` flag only as is shown in Example 12-2.

Example 12-2 Using truss with the dd command

```
# truss -c dd if=/dev/zero of=/dev/null bs=512 count=1024
1024+0 records in
1024+0 records out
signals -----
SIGCHLD      1
total:      1

syscall      seconds  calls  errors
kfork        .00      1
execve       .00      1
_exit        .00      1
kwaitpid     .00      1
_sigaction   .00     10
close        .00      6
kwrite       .00    1034
kread        .02    2050
kiocntl      .00      2      2
open         .00      2
statx        .00      3
shmctl       .00      6      6
shmdt       .00      3
shmat        .00      3
shmget       .00      3
```

_pause	.00	1	1
pipe	.00	3	
kfcntl	.00	2	
	----	---	---
sys totals:	.04	3132	9
usr time:	.00		
elapsed:	.04		

As the example shows, **dd** performs a fork, and the number of system calls during its execution is 3132. However, including the child processes (-f) in the calculation gives a different result from the same run, as shown in Example 12-3.

Example 12-3 Using truss with the dd command including child processes

```
# truss -fc dd if=/dev/zero of=/dev/null bs=512 count=1024
```

```
1024+0 records in
1024+0 records out
signals -----
SIGCHLD      1
total:       1
syscall      seconds  calls  errors
kfork        .00      1
execve       .00      1
_exit        .00      2
kwaitpid     .00      1
_sigaction   .00     13
close        .00     12
kwrite       .00   3089
kread        .04   3076
kiocntl      .00      2      2
open         .00      2
statx        .00      3
shmctl       .00      9      6
shmdt        .00      6
shmat        .00      6
shmget       .00      3
_pause       .00      1      1
pipe         .00      3
fcntl        .00      4
-----
sys totals:  .04   6234    9
usr time:    .00
elapsed:    .04
```

The example shows that the total number of system calls made on behalf of the **dd** program was in fact 6234 because we included all processes that were

necessary for it to perform its task in the statistical output. Because these two samples were run on a AIX system with other loads at the same time, you can disregard the reported time statistics as they are not important here.

12.2.3 Monitoring running processes

In Example 12-4, we track a running process . The process is known and it performs random seeks on one file and random seeks on the other file, then it reads a block from one file and writes it to the other, changing block sizes and the file to read from and write to randomly.

Example 12-4 Extract of sample read_write.c program

```
# expand -4 read_write.c|nl
...(lines omitted)...
90     while (1) {
91         bindex = (random()%12);
92         j = random()%2;
93         if (lseek(fd[j],(random()%FILE_SIZE), SEEK_SET) < 0) {
94             perror("lseek 1");
95             exit(-1);
96         }
97         if (lseek(fd[j==0?1:0],(random()%FILE_SIZE), SEEK_SET) < 0) {
98             perror("lseek 2");
99             exit(-1);
100        }
101        if (read(fd[j],buf,bsize[bindex]) <= 0) {
102            perror("read");
103            exit(-1);
104        }
105        if (write(fd[j==0?1:0],buf,bsize[bindex]) <= 0) {
106            perror("write");
107            exit(-1);
108        }
...(line omitted)...
```

When using **truss** to track the running process, we can see the seeks, reads, and writes as in the extracted output in Example 12-5. The running process name is `read_write`.

Example 12-5 Using truss on a running process¹

```
# ps -Fpid,args|grep read_write|awk '!/grep/{print $1}'
19534
# truss -t lseek,kread,kwrite -p 19534|nl
1  lseek(3, 919890044, 0)           = 919890044
```

¹ Instead of two lines to run the command we could use one: `truss -t lseek,kread,kwrite -p $(ps -Fpid,args | grep read_write | awk '!/grep/{print $1}') | nl`

```

2 lseek(4, 757796945, 0) = 757796945
3 kread(3, "\0\0\0\0\0\0\0\0\0\0"..., 64) = 64
4 kwrite(4, "\0\0\0\0\0\0\0\0\0\0"..., 64) = 64
5 lseek(4, 906212625, 0) = 906212625
6 lseek(3, 332914556, 0) = 332914556
7 kread(4, "\0\0\0\0\0\0\0\0\0\0"..., 128) = 128
8 kwrite(3, "\0\0\0\0\0\0\0\0\0\0"..., 128) = 128
9 lseek(4, 241598273, 0) = 241598273
10 lseek(3, 848068334, 0) = 848068334
11 kread(4, "\0\0\0\0\0\0\0\0\0\0"..., 131072) = 131072
12 kwrite(3, "\0\0\0\0\0\0\0\0\0\0"..., 131072) = 131072
13 lseek(3, 717721518, 0) = 717721518
14 lseek(4, 314891145, 0) = 314891145
15 kread(3, "\0\0\0\0\0\0\0\0\0\0"..., 131072) = 131072
16 kwrite(4, "\0\0\0\0\0\0\0\0\0\0"..., 131072) = 131072
17 lseek(3, 1016755287, 0) = 1016755287
18 lseek(4, 922527047, 0) = 922527047
19 kread(3, "\0\0\0\0\0\0\0\0\0\0"..., 512) = 512
20 kwrite(4, "\0\0\0\0\0\0\0\0\0\0"..., 512) = 512
21 lseek(4, 476810507, 0) = 476810507
22 lseek(3, 117563634, 0) = 117563634
23 kread(4, "\0\0\0\0\0\0\0\0\0\0"..., 512) = 512
24 kwrite(3, "\0\0\0\0\0\0\0\0\0\0"..., 512) = 512
25 lseek(4, 624368317, 0) = 624368317
26 lseek(3, 980376023, 0) = 980376023
27 kread(4, "\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
28 kwrite(3, "\0\0\0\0\0\0\0\0\0\0"..., 1024) = 1024
...(lines omitted)...

```

In lines 1 and 2 in the **truss** output, you see the `lseek` subroutine with the first parameter being the file descriptor used in the program; the second parameter, the byte offset in the file; and the third, the seek operation. This corresponds to the source lines 93 and 97 that call the `lseek` system call. On line 3 the `kread` is tracked with the first parameter as the file descriptor, the second parameter the read buffer sent to the program (in this case all hex 0), and the third parameter being the buffer size (block size), in this case 64 bytes. This corresponds with the read system call on line 101 in the source program. Line 4 shows the path for the `kwrite`, which translates into line 105 in the source program. The first parameter is the file descriptor, the second parameter is the write buffer and the third is the buffer size to write (block size), which is 64 bytes, as for the read system call.

Note that the `lseek` system calls position the file pointers at different offsets in the two files before the read and write commence. Buffer sizes (block sizes) will vary; in the output shown they vary between 64, 128, 131072, 512, and 1024 bytes.

Depending on which system calls **truss** tracks and how the program is written, the output format can vary. The code in Example 12-6 on page 202 and **truss**

output in Example 12-7 show a possible result of using `fprintf` to write output from a program.

Example 12-6 Sample program for fprintf

```
1 #include <stdio.h>
2 main()
3 {
4     fprintf(stderr,"this is from %s, %s %s %s\n","fprintf","yes","it","is");
5 }
```

To track the program with **truss**:

```
truss -o truss.out -tkwrite fprintftest
```

truss will give an output similar to the one in Example 12-7.

Example 12-7 truss output for fprintf

```
# expand truss.out|nl
1 kwrite(2, " t h i s   i s   f r o m".., 13)      = 13
2 kwrite(2, " f p r i n t f", 7)                 = 7
3 kwrite(2, " ,   ", 2)                          = 2
4 kwrite(2, " y e s", 3)                         = 3
5 kwrite(2, "   ", 1)                            = 1
6 kwrite(2, " i t", 2)                           = 2
7 kwrite(2, "   ", 1)                            = 1
8 kwrite(2, " i s", 2)                           = 2
9 kwrite(2, "\n", 1)                             = 1
```

12.2.4 Analyzing file descriptor I/O

With **truss** you can also track what a program is reading and writing; that is, you can actually track the content of the read and write buffers. Instead of including debug statements in a program that shows input and output buffers (read and write), you can use **truss** instead.

Read file descriptors

The small program in Example 12-8 reads 24 bytes from the process file descriptor 0 (standard input) on line 4.

Example 12-8 Sample read program (readit)

```
1 main ()
2 {
3     char buf[24];
4     read(0,buf,sizeof(buf));
5 }
```

The **truss** output (formatted with the **expand** and **n1** commands) will look similar to the output shown in Example 12-9.

Example 12-9 truss output from the sample read program (readit)

```
# echo "hello world\c"|truss -r0 readit 2>&1|expand|n1
1  execve("./readit", 0x2FF22B9C, 0x2FF22BA4)      argc: 1
2  kread(0, 0x2FF22B30, 24)                       = 12
3  h e l l o   w o r l d
4  kfcntl(1, F_GETFL, 0xF06C2968)                = 1
5  kfcntl(2, F_GETFL, 0xF06C2968)                = 1
6  _exit(0)
```

The command line writes the sentence `hello world` to standard input (*stdin*) of the `truss/readit` pipe. **truss** will track file descriptor 0 (*stdin*) with the `-r` flag and we direct the output from **truss** (from *stderr* or file descriptor 2) to *stdin* for the next pipe to the **expand** and **n1** commands (for formatting of the output only). On line 2 of the **truss** output you see the `kread` system call that is created by the read on line 4 in Example 12-6 on page 202. The first parameter to `kread` is file descriptor 0, the second is the read buffer address, and the third is the number of bytes to read. On the end of the line is the return code from the `kread` system call, which is 11. (This is the actual number of bytes read.) On line 3 you see the content of the read buffer containing our `hello world` string².

Write file descriptors

The following small program writes a string of bytes (the number of bytes to write is determined by the length of the string in this case) to process file descriptor 1 (standard output) on line 4 in Example 12-10.

Example 12-10 Sample write program

```
1  main ()
2  {
3      char *buf = "abcdefghijklmnopqrstuvxyz0123456789\0";
4      write(1,buf,strlen(buf));
5  }
```

The **truss** output (formatted with the **expand** and **n1** commands) will look similar to the output shown in Example 12-11.

Example 12-11 truss output from the sample write program

```
# truss -w1 writeit 2>&1 >/dev/null|expand|n1
1  execve("./writeit", 0x2FF22BF0, 0x2FF22BF8)    argc: 1
2  kwrite(1, 0x20000488, 35)                     = 35
3  a b c d e f g h i j k l m n o p q r s t u v x y z 0 1 2 3 4 5 6
```

² The **echo** command would normally add a newline (`\n`) to the end of a string, but since we added `\c` at the end of the string, it did not.

```
4 7 8 9
5 kfcntl(1, F_GETFL, 0x00000000) = 67108865
6 kfcntl(2, F_GETFL, 0x00000000) = 1
7 _exit(0)
```

truss will track file descriptor 1 (stdout) with the **-w** flag, and we direct the output from **truss** (from stderr or file descriptor 2) to stdin for the next pipe to the **expand** and **n1** commands (for formatting of the output only). Note that we discard the output from the **writeit** program itself (**>/dev/null**). On line 2 of the **truss** output, you see the **kwrite** system call that is created by the **read** on line 4 in Example 12-6 on page 202. The first parameter to **kwrite** is file descriptor 1, the second is the write buffer address (0x20000488), and the third parameter is the number of bytes to write (35). On the end of the line is the return code from the **kwrite** system call, which is 35; this is the actual number of bytes written. On line 3 and 4 you see the content of the write buffer containing our string that was declared on line 3 in the source program in the Example 12-6 on page 202³.

Combining different flags

Example 12-12 shows how to use **truss** by combining different flags to track our sample write program. We use the **-t** flag to only track the **kwrite** system call, the **-w** flag will show detailed output from the write buffers to all file descriptors (**all**), and the **-x** flag will show us the raw data of the options to the **kwrite** system call (in hex).

Example 12-12 truss output using combined flags for the writeit sample program

```
# truss -xkwrite -tkwrite -wall writeit 2>&1 >/dev/null|expand|n1
1 kwrite(0x00000001, 0x20000488, 0x00000023) = 0x00000023
2 a b c d e f g h i j k l m n o p q r s t u v x y z 0 1 2 3 4 5 6
3 7 8 9
```

On line 1 of the **truss** output you see the **kwrite** system call that is created by the **read** on line 4 in the Example 12-6 on page 202. The first parameter to **kwrite** is file descriptor 1 (in hex 0x00000001), the second is the write buffer address (in hex 0x20000488), and the third parameter is the number of bytes to write (in hex 0x00000023). On the end of the line is the return code from the **kwrite** system call, which is 35 (in hex 0x00000023); this is the actual number of bytes written. On lines 2 and 3 you see the content of the write buffer containing our string that was declared on line 3 in the source program in the Example 12-6 on page 202.

³ The **\0** in the bufferstring is just to make sure that the end of the string ends with binary zero, which indicates the end of a byte string in the C programming language.

12.2.5 Checking program parameters

To check the parameters passed to the program when it was started, you can use the `-a` flag with `truss`. This can be done if you start a program and track it with `truss`, but you can do it on a running process as well. In Example 12-13 we use `truss` to track the system calls loaded by the `/etc/init`.

Example 12-13 Using truss to track system calls

```
# truss -a -p 1
psargs: /etc/init
_pause()                (sleeping...)
_pause()                Err#4  EINTR
    Received signal #20, SIGCHLD [caught]
kwaitpid(0x2FF229D0, -1, 5, 0x00000000, 0x00000000) = 348298
open("/etc/security/monitord_pipe", O_RDWR|O_NONBLOCK) Err#2  ENOENT
kwaitpid(0x2FF229D0, -1, 5, 0x00000000, 0x00000000) = 0
ksetcontext_sigreturn(0x2FF22A70, 0x00000000, 0x20029C8C, 0x0000D0B2,
0x00000000
, 0x00000000, 0x00000000, 0x00000000)
incinterval(0, 0x2FF22DC8, 0x2FF22DD8)          = 0
statx("/etc/inittab", 0x200295E8, 76, 0)         = 0
sigprocmask(0, 0x2FF22A00, 0x00000000)         = 0
lseek(0, 0, 0)                                  = 0
kread(0, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 648) = 648
lseek(0, 0, 1)                                  = 648
kread(0, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 648) = 648
lseek(0, 0, 1)                                  = 1296
kread(0, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 648) = 648
lseek(0, 0, 1)                                  = 1944
kread(0, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 648) = 648
lseek(0, 0, 1)                                  = 2592
... (lines omitted) ...
^CPstatus: process is not stopped
```

Because the process we tracked was `init` with process ID 1, `truss` reported that the process was not stopped when we discontinued tracking by using `CTRL + C` to stop `truss`. The output shown after `psargs:` is the parameters that the program got when it was started with one of the `exec` subroutines. In this case it was only the program name itself, which is always the first parameter (`/etc/init`).

12.2.6 Checking program environment variables

To check the environment variables that are set for a program when it is started, you can use the `-e` flag with `truss`. This can be done if you start a program and track it with `truss`. If you only want to see the environment in the `truss` output, you must include the `exec` system call that the process uses. In Example 12-14 on page 206 it is the `execve` system call that is used by the `date` command.

Example 12-14 Using truss to display the environment of a process

```
# truss -e -texecve date 2>&1 >/dev/null|expand|n1
1  execve("/usr/bin/date", 0x2FF22B94, 0x2FF22B9C)  argc: 1
2   envp: _=/usr/bin/truss LANG=en_US LOGIN=root VISUAL=vi
3
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/java130/jre/bin:/
usr/java130/bin:/usr/vac/bin:/usr/samples/kernel:/usr/vac/bin:.:
4   LC_FASTMSG=true CGI_DIRECTORY=/var/docsearch/cgi-bin EDITOR=vi
5   LOGNAME=root MAIL=/usr/spool/mail/root LOCPATH=/usr/lib/nls/loc
6   PS1=root@wlmhost:$PWD: DOCUMENT_SERVER_MACHINE_NAME=localhost
7   USER=root AUTHSTATE=compat DEFAULT_BROWSER=netscape
8   SHELL=/usr/bin/ksh ODMDIR=/etc/objrepos DOCUMENT_SERVER_PORT=49213
9   HOME=/ TERM=ansi MAILMSG=[YOU HAVE NEW MAIL]
10  ITECONFIGSRV=/etc/IMNSearch PWD=/home/roden/src
11  DOCUMENT_DIRECTORY=/usr/docsearch/html TZ=CST6CDT
12  PROJECTDIR=/home/roden ENV=//.kshrc
13  ITECONFIGCL=/etc/IMNSearch/clients ITE_DOC_SEARCH_INSTANCE=search
14  A__z=! LOGNAME
15  NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
```

We discard the output from the **date** command and format the output with the **expand** and **n1** command. The environment variables are displayed between lines 2 and 15 in the output above. To monitor a running process environment use the **ps** command as in Example 12-15 that uses the current shells PID (\$\$). Refer to Chapter 8, “The ps command” on page 127 for more details.

Example 12-15 Using ps to check another process environment

```
# ps euww $$
USER  PID %CPU %MEM  SZ  RSS  TTY STAT  STIME TIME COMMAND
root   34232  0.0  0.0 1020 1052 pts/15 A    11:21:18  0:00 -ksh TERM=vt220
AUTHSTATE=compat SHELL=/usr/bin/ksh HOME=/ USER=root
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/java130/jre/bin:/
usr/java130/bin:/usr/vac/bin TZ=CST6CDT LANG=en_US LOCPATH=/usr/lib/nls/loc
LC_FASTMSG=true ODMDIR=/etc/objrepos ITECONFIGSRV=/etc/IMNSearch
ITECONFIGCL=/etc/IMNSearch/clients ITE_DOC_SEARCH_INSTANCE=search
DEFAULT_BROWSER=netscape DOCUMENT_SERVER_MACHINE_NAME=localhost
DOCUMENT_SERVER_PORT=49213 CGI_DIRECTORY=/var/docsearch/cgi-bin
DOCUMENT_DIRECTORY=/usr/docsearch/html LOGNAME=root LOGIN=root
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
```

12.2.7 Tracking child processes

Another way to use **truss** is to track the interaction between a parent process and child processes. Example 12-16 on page 207 shows how to monitor a running process (**/usr/sbin/inetd**) and, while doing the tracking, opening a telnet session.

Example 12-16 Using truss to track child processes

```
# truss -a -f -tkfork,execv -p 6716
6716: psargs: /usr/sbin/inetd
6716: kfork() = 29042
29042: kfork() = 26546
26546: kfork() = 20026
26546: (sleeping...)
26546: kfork() = 20028
26546: kfork() = 20030
26546: (sleeping...)
^CPstatus: process is not stopped
Pstatus: process is not stopped
Pstatus: process is not stopped
```

The left column shows the process ID that each output belongs to. The lines that start with 6716 are the parent process (`inetd`) because we used `-p 6716` to start the tracking from this process ID. On the far right in the output is the return code from the system call, and for `kfork` it is the process ID of the spawned child. (The parent part of `kfork` will get a return code of zero.) The next child with process ID 29042 is the `telnet` daemon, as can be seen by using the `ps` command as in the sample output in Example 12-17.

Example 12-17 Using ps to search for process ID

```
# ps -eFpid,args|grep 29042|grep -v grep
29042 telnetd -a
```

The `telnet` daemon performs a `fork` system call as well (after authenticating the login user) and the next child is 26546, which is the authenticated users' login shell as can be seen by using the `ps` command as in Example 12-18.

Example 12-18 Using ps to search for process ID

```
# ps -eFpid,args|grep 26546|grep -v grep
26546 -ksh
```

We can see in the `truss` output that the login shell (`ksh`) is forking as well, which is one of the primary things that shells do. To illustrate a point about shells, we track it while we run the `ps`, `ls`, `date`, and `sleep` commands one after the other in our login shell. `truss` shows us that the shell did a `fork` system call every time, as can be seen in the output in Example 12-19.

Example 12-19 Using truss to track ksh with ps, ls, date, and sleep

```
# truss -a -f -tkfork,execv -p 26546
26546: psargs: -ksh
26546: kfork() = 29618
26546: kfork() = 29620
```

```

26546: kfork() = 29622
26546: kfork() = 29624
26546: (sleeping...)
^CPstatus: process is not stopped

```

In the example, process ID 29618 is the **ps** command, process ID 29620 is the **ls** command, process ID 29622 is the **date** command, and process ID 29624 is the **sleep** command.

Example 12-20 shows how many forks are done by running the **make** command to compile one program with the **cc** compiler from the same shell.

Example 12-20 Using truss to track ksh with make

```

# truss -a -f -tkfork,execv -p 26546
26546: psargs: -ksh
26546: kfork() = 26278
26278: kfork() = 29882
29882: kfork() = 28388
29882: kfork() = 28390
29882: kfork() = 28392
28392: kfork() = 29342
26546: (sleeping...)
^CPstatus: process is not stopped

```

It took six processes to compile one program by using **make** and **cc**. By using the summary output with the **-c** flag to **truss**, it nicely summarizes for us, as Example 12-21 shows.

Example 12-21 Using truss to track ksh with make and use summarized output

```

# truss -c -a -f -tkfork,execv -p 26546
psargs: -ksh
^CPstatus: process is not stopped
syscall      seconds    calls  errors
kfork          .00         6
-----
sys totals:    .00         6      0
usr time:     .00
elapsed:     .00

```

The output confirms that the **ksh/make** process tree performed six fork system calls to handle the **make** command for this compile.

12.2.8 Checking user library call

In AIX 5L Version 5.2, the **truss** command can check library call to user subroutines. Example 12-22 shows the trace of malloc subroutine from running the **ls** command.

Example 12-22 User subrouting trace

```
# truss -u libc.a::malloc ls
execve("/usr/bin/ls", 0x2FF22B80, 0x2FF22B88)   argc: 1
sbrk(0x00000000)                             = 0x20000EA8
sbrk(0x00000008)                             = 0x20000EA8
sbrk(0x00010010)                             = 0x20000E80
getuidx(4)                                    = 0
getuidx(2)                                    = 0
getuidx(1)                                    = 0
getgidx(4)                                    = 0
getgidx(2)                                    = 0
getgidx(1)                                    = 0
__loadx(0x01000080, 0x2FF1E940, 0x00003E80, 0x2FF228D0, 0x00000000) = 0xD0079130
->libc.a:malloc(0xc)
<-libc.a:malloc() = 20001058                 0.000000
->libc.a:malloc(0x188)
<-libc.a:malloc() = 20001078                 0.000000
->libc.a:malloc(0x40)
<-libc.a:malloc() = 20001208                 0.000000
->libc.a:malloc(0x3c)
<-libc.a:malloc() = 20001258                 0.000000
__loadx(0x01000180, 0x2FF1E930, 0x00003E80, 0xF015DDF0, 0xF015DD20) = 0x20011378
__loadx(0x07080000, 0xF015DDC0, 0xFFFFFFFF, 0x20011378, 0x00000000) = 0x20012210
. . . (lines omitted) . . .
```

The vmstat command

The **vmstat** command is very useful for reporting statistics about kernel threads, virtual memory, disks, and CPU activity. Reports generated by the **vmstat** command can be used to balance system load activity. These systemwide statistics (among all processors) are calculated as averages for values expressed as percentages or, otherwise, as sums.

The **vmstat** command resides in `/usr/bin` and is part of the `bos.acct` fileset, which is installable from the AIX base installation media.

13.1 vmstat

The syntax of the **vmstat** command is:

```
vmstat [ -fsiItv ] [Drives] [ Interval [Count] ]
```

Flags

- f** Reports the number of forks since system startup.
- s** Writes to standard output the contents of the sum structure, which contains an absolute count of paging events since system initialization. The **-s** option is exclusive of the other **vmstat** command options. These events are described in 13.2, “Examples for vmstat” on page 213.
- i** Displays the number of interrupts taken by each device since system startup.
- l** Displays an I/O-oriented view with the new columns, p under heading kthr, and columns fi and fo under heading page instead of the columns re and cy in the page heading.
- t** Prints the time stamp next to each line of output of **vmstat**. The time stamp is displayed in the HH:MM:SS format, but it will not be printed if the **-f**, **-s**, or **-i** flags are specified.
- v** Writes to standard output various statistics maintained by the Virtual Memory Manager. The **-v** flag can only be used with the **-s** flag.

Both the **-f** and **-s** flags can be entered on the command line, but the system will only accept the first flag specified and override the second flag.

If the **vmstat** command is invoked without flags, the report contains a summary of the virtual memory activity since system startup. If the **-f** flag is specified, the **vmstat** command reports the number of forks since system startup. The **Drives** parameter specifies the name of the physical volume.

Parameters

- Drives** hdisk0, hdisk1, and so forth. Disk names can be listed using the **lspv** command. RAID disks will appear as one logical disk drive.
- Interval** Specifies the update period (in seconds).
- Count** Specifies the number of iterations.

The **Interval** parameter specifies the amount of time in seconds between each report. The first report contains statistics for the time since system startup. Subsequent reports contain statistics collected during the interval since the previous report. If the **Interval** parameter is not specified, the **vmstat** command

generates a single report and then exits. The Count parameter can only be specified with the Interval parameter. If the Count parameter is specified, its value determines the number of reports generated and the number of seconds apart. If the Interval parameter is specified without the Count parameter, reports are continuously generated. A Count parameter of 0 is not allowed.

13.1.1 Information about measurement and sampling

The kernel maintains statistics for kernel threads, paging, and interrupt activity, which the `vmstat` command accesses through the use of the `knlist` subroutine and the `/dev/kmem` pseudo-device driver. The disk input/output statistics are maintained by device drivers. For disks, the average transfer rate is determined by using the active time and number of transfers information. The percent active time is computed from the amount of time the drive is busy during the report.

The `vmstat` command generates five types of reports:

- ▶ Virtual memory activity
- ▶ Forks
- ▶ Interrupts
- ▶ Sum structure
- ▶ Input/Output

13.2 Examples for vmstat

This section shows examples and descriptions of the `vmstat` reports.

13.2.1 Virtual memory activity

The `vmstat` command writes the virtual memory activity to standard output. It is a very useful report because it gives a good summary of the system resources on a single line. Example 13-1 shows the virtual memory report. The first line of this report should be ignored because it is an average since the last system reboot.

Example 13-1 Virtual memory report

```
# vmstat 2 5
kthr  memory                page                faults                cpu
-----
r  b   avm  fre  re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa
0  0 51696 49447  0  0  0  6  36  0 104 188  65  0  1 97  2
0  0 51698 49445  0  0  0  0  0  0 472 1028 326  0  1 99  0
0  0 51699 49444  0  0  0  0  0  0 471  990 327  0  1 99  0
0  0 51700 49443  0  0  0  0  0  0 473  992 330  0  1 99  0
0  0 51701 49442  0  0  0  0  0  0 469  986 329  0  0 99  0
```

The reported fields are:

- kthr Indicates the number of kernel thread state changes per second over the sampling interval.
- r Average number of threads on the run queues per second. These threads are only waiting for CPU time and are ready to run. Each thread has a priority ranging from zero to 127. Each CPU has a run queue for each priority; therefore there are 128 run queues for each CPU. Threads are placed on the appropriate run queue. Refer to 1.2.2, “Processes and threads” on page 6 for more information about thread priorities. The run queue reported by vmstat is across all run queues and all CPUs. Each CPU has its own run queue. The maximum you should see this value increase to is based on the following formula: $5 \times (Nproc - Nbind)$, where Nproc is the number of active processors and Nbind is the number of active processors bound to processes with the bindprocessor command.

Note: A high number on the run queue does not necessarily translate to a performance slowdown because the threads on the run queue may not require much processor time and will therefore be quick to run, thereby clearing the run queue quickly.

- b Average number of threads on block queue per second. These threads are waiting for resource or I/O. Threads are also located in the wait queue (wa) when scheduled, but are waiting for one of their threads pages to be paged in. On an SMP system there will always be one thread on the block queue. If compressed file systems are used, then there will be an additional thread on the block queue.
- memory Information about the use of virtual and real memory. Virtual pages are considered active if they have been accessed. A page is 4096 bytes.
- avm Active Virtual Memory (avm) indicates the number of virtual pages accessed. This is not an indication of available memory.
- fre This indicates the size of the free list. A large portion of real memory is utilized as a cache for file system data. It is not unusual for the size of the free list to remain small. The VMM maintains this free list. The free list entries point to buffers of 4 K pages that are readily available when required. The minimum number of pages is defined by minfree. See “The page replacement algorithm” on page 232 for more information. The default value is 120. If the number of the free list drops below that defined by minfree, then the VMM steals pages until maxfree+8 is reached. Terminating applications release their memory, and those frames are added back to the free list. Persistent pages (files) are not added back to the free list. They remain in memory until the VMM

steals their pages. Persistent pages are also freed when their corresponding file is deleted. A small value of `fre` could cause the system to start thrashing due to overcommitted memory. This does not indicate the amount of unused memory.

- `Page` Information about page faults and paging activity. These are averaged over the interval and given in units per second.
- `re` The number of reclaims per second. During a page fault, when the page is on the free list and has not been reassigned, this is considered a reclaim because no new I/O request has been initiated. It also includes the pages last requested by the VMM for which I/O has not been completed or those prefetched by VMM's read-ahead mechanism but hidden from the faulting segment.

Note: As of AIX Version 4, reclaims are no longer supported as the algorithm is costly in terms of performance. Normally the delivered value will be zero.

- `pi` Indicates the number of page in requests. Those are pages that have been paged to paging space and are paged into memory when required by way of a page fault. Normally you would not want to see more than five sustained pages per second (as a rule of thumb) reported by `vmstat` as paging (particularly page in (`pi`)) effects performance. A system that is paging data in from paging space results in slower performance because the CPU has to wait for data before processing the thread. A high value of `pi` may indicate a shortage of memory or indicate a need for performance tuning. See `vm0` for more information.
- `po` The number of pages out process. The number of pages per second that is moved to paging space. These pages are paged out to paging space by the VMM when more memory is required. They will stay in paging space and be paged in if required. A terminating process will disclaim its pages held in paging space, and pages will also be freed when the process gives up the CPU (is preempted). `po` does not necessarily indicate thrashing, but if you are experiencing high paging out (`po`) then it may be necessary to investigate the application `vm0` command parameters `minfree` and `max free`, and the environmental variable `PSALLOC`. For an overview of "Performance Overview of the Virtual Memory Manager (VMM)," refer to:
http://www16.boulder.ibm.com/pseries/en_US/infocenter/base/aix.htm
- `fr` Number of pages freed. When the VMM requires memory, VMM's page-replacement algorithm is employed to scan the Page Frame Table (PFT) to determine which pages to steal. If a page has not been referenced since the last scan, it can be stolen. If there has been no I/O

for that page then the page can be stolen without being written to disk, thus minimizing the effect on performance.

sr Represents pages scanned by the page-replacement algorithm. When page stealing occurs (when fre of vmstat goes below minfree of vmo), the pages in memory are scanned to determine which can be stolen.

Note: Look for a large ratio of fr to sr (fr:sr), which could indicate overcommitted memory. A high ratio shows that the page stealer has to work hard to find memory to steal.

Example 13-2 shows high pi and po indicating high paging. Note that the wa column is high, indicating we are waiting on the disk I/O, probably for paging. Note the ratio of fr:sr as the page stealers are looking for memory to steal and the number of threads on the b queue waiting for data to be paged in. Also note how wa is reduced when the page stealers have completed stealing memory, and how the fre column increases as a result of page stealing.

Example 13-2 An example of high paging

kthr		memory				page				faults			cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
2	3	298565	163	0	14	58	2047	8594	0	971	217296	1286	23	26	17	34
2	2	298824	124	0	29	20	251	352	0	800	248079	1039	22	28	22	29
1	7	300027	293	0	15	6	206	266	0	1150	91086	479	7	14	9	69
0	13	300233	394	0	1	0	127	180	0	894	6412	276	2	2	0	96
0	14	300453	543	0	4	0	45	82	0	793	5976	258	1	2	0	97
0	14	301488	329	0	2	2	116	179	0	803	6806	282	1	3	0	96
0	14	302207	435	0	5	4	112	159	0	821	12349	402	2	3	0	95
3	9	301740	2240	0	70	9	289	508	0	963	187874	1089	19	31	6	44
1	4	271719	30561	0	39	0	0	0	0	827	203604	1217	21	31	19	30
3	2	269996	30459	0	16	0	0	0	0	764	182351	1387	18	25	34	23

cy This refers to the page replacement algorithm. The value refers to the number of times the page replacement algorithm does a complete cycle through memory looking for pages to steal. If this value is greater than zero, this means severe memory shortages.

The page stealer steals memory until maxfree is reached; see “The page replacement algorithm” on page 232 for more details. This usually occurs before the memory has been completely scanned, hence the value will stay at zero. However if the page stealer is still looking for memory to steal and the memory has already been scanned, then the cy value will increment to one. Each scan will increment cy until maxfree has been satisfied, at which time page stealing will stop and cy will be reset to zero.

You are more likely to see the `cy` value increment when there is less physical memory installed, as it takes a shorter time for memory to be completely scanned and memory shortage is more likely.

<code>Faults</code>	Trap and interrupt rate averages per second over the sampling interval.
<code>in</code>	Number of device or hardware interrupts per second observed in the interval. An example of an interrupt would be the 10 ms clock interrupt or a disk I/O completion. Due to the clock interrupt, the minimum value you see is 100.
<code>sy</code>	Number of system calls per second. These are resources provided by the kernel for the user processes and data exchange between the process and the kernel. This reported value can vary depending on workloads and on how the application is written, so it is not possible to determine a value for this. Any value of 10,000 and more should be investigated.

Tip: You should run `vmstat` when your system is busy and performing to expectations so you can determine the average number of system calls for your system.

<code>cs</code>	Kernel thread context switches per second. A CPU's resource is divided into 10 ms time slices and a thread will run for the full 10 ms or until it gives up the CPU (is preempted). When another thread gets control of the CPU, the previous thread's contexts and working environments must be saved and the new thread's contexts and working environment must be restored. AIX handles this efficiently. Any significant increase in context switches should be investigated. See "Time slice" on page 170 for details about the <code>timeslice</code> parameter.
<code>cpu</code>	Breakdown of percentage use of CPU time. The columns <code>us</code> , <code>sy</code> , <code>id</code> , and <code>wa</code> are averages over all of the processors. I/O wait is a global statistic and is not processor specific.
<code>us</code>	User time. This indicates the amount of time a program is in user mode. Programs can run in either user mode or system mode. In user mode, the program does not require the resources of the kernel to manage memory, set variables, or perform computations.
<code>sy</code>	System time indicates the amount of time a program is in system mode; that is, processes using kernel processes (<code>kprocs</code>) and others that are using kernel resources. Processes requiring the use of kernel services must switch to service mode to gain access to the services, such as to open a file or read/write data.

Note: A CPU bottleneck could occur if `us` and `sy` combined together add up to approximately 80 percent or more.

- `id` CPU idle time. This indicates the percentage of time the CPU is idle without pending I/O. When the CPU is idle, it has nothing on the run queue. When there is a high aggregate value for `id`, it means there was nothing for the CPU to do and there were no pending I/Os. A process called `wait` is bound to every CPU on the system. When the CPU is idle, and there are no local I/Os pending, any pending I/O to a Network File System (NFS) is charged to `id`.
- `wa` CPU wait. CPU idle time during which the system had at least one outstanding I/O to disk (whether local or remote) and asynchronous I/O was not in use. An I/O causes the process to block (or sleep) until the I/O is complete. Upon completion, it is placed on the run queue. A `wa` of over 25 percent could indicate a need to investigate the disk I/O subsystem for ways to improve throughput, such as load balancing. Refer to Chapter 26, “The `fileplace` command” on page 479 for information about placement of files.

`vmstat` marks an idle CPU as wait I/O (`wio`) if an outstanding I/O was started on that CPU. With this method, `vmstat` will report lower `wio` times when more processors are installed, just a few threads are doing I/O, and the system is otherwise idle. For example, a system with four CPUs and one thread doing I/O will report a maximum of 25 percent `wio` time. A system with 12 CPUs and one thread doing I/O will report a maximum of eight percent `wio` time. Network File System (NFS) client reads/writes go through the Virtual Memory Manager (VMM), and the time that NFS block I/O daemons (`biods`) spend in the VMM waiting for an I/O to complete is reported as I/O wait time.

Important: `wa` occurs when the CPU has nothing to do and is waiting for at least one I/O request. Therefore, `wa` does not necessarily indicate a performance bottleneck.

Example 13-3 Virtual memory report

kthr		memory				page				faults				cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa	
4	13	2678903	254	0	0	0	7343	29427	0	6111	104034	17964	22	18	18	42	
6	14	2678969	250	0	0	0	7025	26692	0	6253	216943	17678	29	28	10	33	
8	13	2678969	244	0	0	0	6625	28218	0	6295	273936	17639	32	29	9	30	
8	13	2678969	252	0	0	0	5731	23555	0	5828	264980	16325	35	26	8	31	
8	13	2678970	256	0	0	0	6571	35508	0	6209	278478	18161	34	29	8	28	
6	13	2678970	246	0	0	0	7527	58083	0	6658	214601	20039	31	26	10	33	

10	13	2679402	197	0	0	0	7882	54975	0	6482	285458	18026	40	31	5	25
8	16	2679431	249	0	0	0	9535	40808	0	6582	283539	16851	39	32	5	24
10	13	2679405	255	0	0	0	8328	41459	0	6256	264752	15318	39	32	5	24
9	15	2678982	255	0	0	0	8240	36591	0	6300	244263	17771	32	29	8	31

In Example 13-3 on page 218, you can observe the following:

- ▶ The block queue is high.
- ▶ There is no paging. If paging was occurring on the system you can tune `minfree` and `maxfree`. See 14.1.2, “Recommendations and precautions for `vmo`” on page 235 for details.
- ▶ As can be seen by the `fr:sr` ratio, the page stealers are working hard to find memory, and, as `pi` is zero, the memory is being stolen successfully without the need for paging.
- ▶ There is a lot of context switching, so tuning time slices with `schedo` could be beneficial. See “Time slice” on page 170 for more details.
- ▶ `us+sy` does not exceed 80 percent, so the system is not CPU bound
- ▶ There is I/O wait (`wa`) when the system is not idle. Tuning the disk I/O or NFS (if the system has NFS) could be beneficial. Looking for lock contention in file systems could also be beneficial. Look for busy file I/O with the `filemon` command. See “Analyzing the physical volume reports” on page 464 for more details.

To comment on any other columns in the report, you would need a baseline that was made when the system was performing normally.

13.2.2 Forks report

This writes to standard output the number of forks since the last system startup. (A *fork* is the creation of a new process.) You would not usually want to see more than three forks per second. Use the `sar -P ALL -c 5 2` command to monitor the number of forks per second. See 9.2.5, “Monitoring system calls” on page 151 for more details.

You can monitor the number of forks per second by running this command every minute and making sure the change between the outputs does not exceed 180. An example is shown in Example 13-4.

Example 13-4 Forks report

```
# vmstat -f
          34770 forks
```

13.2.3 Interrupts report

This writes to standard output the number of interrupts per device since the last system startup. Subsequent iterations of **vmstat** within the same command, as in Example 13-5, produce the number of interrupts for the previous iteration. Example 13-5 produces an interrupt report with a delay of two seconds, three times.

Example 13-5 Interrupt report

```
# vmstat -i 2 3
priority level  type  count module(handler)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/planar_pal_chrp(195f770)
 0     254  hardware 12093 i_hwassist_int(1c9468)
 3       1  hardware 106329 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 3       3  hardware 651315 /usr/lib/drivers/pci/cstokdd(1a99104)
 3      10  hardware  9494 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 4       1  hardware   402 /usr/lib/drivers/isa/kbddd_chrp(1ac0710)
 4      12  hardware  1540 /usr/lib/drivers/isa/msedd_chrp(1ac6890)
priority level  type  count module(handler)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/planar_pal_chrp(195f770)
 0     254  hardware    0 i_hwassist_int(1c9468)
 3       1  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 3       3  hardware   11 /usr/lib/drivers/pci/cstokdd(1a99104)
 3      10  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 4       1  hardware    0 /usr/lib/drivers/isa/kbddd_chrp(1ac0710)
 4      12  hardware    0 /usr/lib/drivers/isa/msedd_chrp(1ac6890)
priority level  type  count module(handler)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bc18)
 0      15  hardware    0 /usr/lib/drivers/planar_pal_chrp(195f770)
 0     254  hardware    0 i_hwassist_int(1c9468)
 3       1  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 3       3  hardware    7 /usr/lib/drivers/pci/cstokdd(1a99104)
 3      10  hardware    0 /usr/lib/drivers/pci/s_scsiddpin(198bb10)
 4       1  hardware    0 /usr/lib/drivers/isa/kbddd_chrp(1ac0710)
 4      12  hardware    0 /usr/lib/drivers/isa/msedd_chrp(1ac6890)
```

The reported fields are as follows:

priority This refers to the interrupt priority as defined in `/usr/include/sys/intr.h`. The priorities range from zero to 11, where zero means fully disabled and 11 means fully enabled. (Anyone can interrupt the CPU.) The lower the priority number, the higher the priority. If the CPU is in

interrupt mode at priority 10 when if a priority three interrupt occurs on that CPU, then the interrupt handler for priority 10 is pre-empted. If, for example, a CPU is at priority zero or one and a priority nine interrupt comes in, then the priority nine interrupt will get queued and only gets processed after the previous interrupt has finished its processing.

The priority can be important as higher-priority interrupts may stop the CPU from servicing other, lower-priority interrupts for other services. For example, the streams drivers that handle Ethernet traffic may not be serviced, which in turn may fill the network buffers, causing other problems. The problem is compounded if the higher priority thread stays running on the CPU for a long time. Normally, high-priority interrupts are serviced within a short time frame to prevent this happening, but it is not always possible to overcome this because the priority is not tunable. In this case, on an SMP system, you could bind specific interrupts to specific CPUs using the `bindintcpu` command. Refer to Chapter 18, “The `bindintcpu` and `bindprocessor` commands” on page 289 for more details. This would ensure that the interrupts were serviced within the required time frame.

level	Refers to the bus interrupt level that you can see on a device when doing an <code>lsattr -E1 <device></code> command. The level is not a tunable parameter. It is set by IBM development.
type	Indicates the type of interface.
count	The number of interrupts for that device/interrupt handler.
module(handler)	The device driver software.

There are no recommendations for analyzing the interrupt report. Be aware of how many interrupts to expect on your system; if you notice a higher number than usual, investigate the device as shown in `module handler` further.

13.2.4 VMM statistics report

Reports various statistics maintained by the Virtual Memory Manager. Example 13-6 displays VMM statistics.

Example 13-6 VMM statistics

```
# vmstat -v
2097152 memory pages
```

```

2042335 1ruable pages
1906992 free pages
    1 memory pools
    62216 pinned pages
    80.1 maxpin percentage
    20.0 minperm percentage
    80.0 maxperm percentage
    2.0 numperm percentage
    42511 file pages
    0.0 compressed percentage
    0 compressed pages
    0.0 numclient percentage
    80.0 maxclient percentage
    0 client pages
    0 remote pageouts scheduled
    0 pending disk I/Os blocked with no pbuf
2898524 paging space I/Os blocked with no psbuf
371909 filesystem I/Os blocked with no fsbuf
    0 client filesystem I/Os blocked with no fsbuf
    0 external pager filesystem I/Os blocked with no fsbuf

```

To explain the outputs :

memory pages	Size of real memory in number of 4 KB pages.
1ruable pages	Number of 4 KB pages considered for replacement. This number excludes the pages used for VMM internal pages and the pages used for the pinned part of the kernel text.
free pages	Number of free 4 KB pages.
memory pools	Tuning parameter (managed using vmo) specifying the number of pools.
pinned pages	Number of pinned 4 KB pages.
maxpin percentage	Tuning parameter (managed using vmo) specifying the percentage of real memory that can be pinned.
minperm percentage	Tuning parameter (managed using vmo) in percentage of real memory. This specifies the point below which file pages are protected from the re-page algorithm.
maxperm percentage	Tuning parameter (managed using vmo) in percentage of real memory. This specifies the point above which the page stealing algorithm steals only file pages.
file pages	Number of 4 KB pages currently used by the file cache.
compressed percentage	Percentage of memory used by compressed pages.

compressed pages Number of compressed memory pages.

numclient_percentage Percentage of memory occupied by client pages.

maxclient_percentage Tuning parameter (managed using vmo) specifying the maximum percentage of memory that can be used for client pages.

client pages Number of client pages.

remote pageouts scheduled Number of pageouts scheduled for client filesystems.

pending disk I/Os blocked with no pbuf Number of pending disk I/O requests blocked because no pbuf was available. Pbufs are pinned memory buffers used to hold I/O requests at the logical volume manager layer.

paging space I/Os blocked with no psbuf Number of paging space I/O requests blocked because no psbuf was available. Psbufs are pinned memory buffers used to hold I/O requests at the virtual memory manager layer.

filesystem I/Os blocked with no fsbuf Number of filesystem I/O requests blocked because no fsbuf was available. Fsbuf are pinned memory buffers used to hold I/O requests in the filesystem layer.

client filesystem I/Os blocked with no fsbuf Number of client filesystem I/O requests blocked because no fsbuf was available. NFS (Network File System) and VxFS (Veritas) are client filesystems. Fsbuf are pinned memory buffers used to hold I/O requests in the filesystem layer.

external pager filesystem I/Os blocked with no fsbuf Number of external pager client filesystem I/O requests blocked because no fsbuf was available. JFS2 is an external pager client filesystem. Fsbuf are pinned memory buffers used to hold I/O requests in the filesystem layer.

13.2.5 Sum structure report

This writes to standard output the contents of the sum structure, which contains an absolute count of paging events since system initialization as shown in Example 13-7. The `-s` option is exclusive of the other `vmstat` command options.

Example 13-7 Sum structure report

```
# vmstat -s
18379397 total address trans. faults
 8004558 page ins
 5294063 page outs
   87355 paging space page ins
 699899  paging space page outs
    0 total reclaims
 6139830 zero filled pages faults
 3481200 executable filled pages faults
61905822 pages examined by clock
   493 revolutions of the clock hand
11377921 pages freed by the clock
 315896 backtracks
    0 lock misses
 7178736 free frame waits
    3 extend XPT waits
3665717 pending I/O waits
12920977 start I/Os
 7766830 iodes
 81362747 cpu context switches
134805028 device interrupts
    0 software interrupts
    0 traps
253117680 syscalls
```

This report is not generally used for resolving performance issues. It is, however, useful for determining the how much paging and the type of paging during benchmarking.

These events are described as follows:

address translation faults

Incremented for each occurrence of an address translation page fault. I/O may or may not be required to resolve the page fault. Storage protection page faults (lock misses) are not included in this count.

page ins

Incremented for each page read in by VMM. The count is incremented for page ins from paging space and file space. Along with the page out statistic, this represents the total amount of real I/O initiated by the VMM.

page outs	Incremented for each page written out by the VMM. The count is incremented for page outs to page space and for page outs to file space. Along with the page referenced, this represents the total amount of real I/O initiated by VMM.
paging space page ins	Incremented for VMM-initiated page ins from paging space only.
paging space page outs	Incremented for VMM initiated page outs to paging space only.
total reclaims	Incremented when an address translation fault can be satisfied without initiating a new I/O request. This can occur if the page has been previously requested by VMM but the I/O has not yet completed, or if the page was pre-fetched by VMM's read-ahead algorithm but was hidden from the faulting segment, or if the page has been put on the free list and has not yet been reused.
zero-filled page faults	Incremented if the page fault is to working storage and can be satisfied by assigning a frame and zero-filling it.
executable-filled page faults	Incremented for each instruction page fault.
pages examined by the clock	VMM uses a clock-algorithm to implement a pseudo Least Recently Used (LRU) page replacement scheme. Pages are aged by being examined by the clock. This count is incremented for each page examined by the clock.
revolutions of the clock hand	Incremented for each VMM clock revolution (that is, after each complete scan of memory).
pages freed by the clock	Incremented for each page the clock algorithm selects to free from real memory.
backtracks	Incremented for each page fault that occurs while resolving a previous page fault (the new page fault must be resolved first and then initial page faults can be backtracked).
lock misses	VMM enforces locks for concurrency by removing addressability to a page. A page fault can occur due to a

	lock miss, and this count is incremented for each such occurrence.
free frame waits	Incremented each time a process is waited by VMM while free frames are gathered.
extend XPT waits	Incremented each time a process is waited by VMM due to a commit in progress for the segment being accessed.
pending I/O waits	Incremented each time a process is waited by VMM for a page-in I/O to complete.
start I/Os	Incremented for each read or write I/O request initiated by VMM. This count should equal the sum of page-ins and page-outs.
iodones	Incremented at the completion of each VMM I/O request.
CPU context switches	Incremented for each CPU context switch (dispatch of a new process).
device interrupts	Incremented on each hardware interrupt.
software interrupts	Incremented on each software interrupt. A software interrupt is a machine instruction similar to a hardware interrupt that saves some state and branches to a service routine. System calls are implemented with software interrupt instructions that branch to the system call handler routine.
traps	Not maintained by the operating system.
syscalls	Incremented for each system call.

13.2.6 I/O report

Example 13-8 shows the I/O report in which the **vmstat** command writes to standard output the I/O activity since system startup.

Example 13-8 I/O report

```
# vmstat -It 2 10
-----
# kthr      memory          page        faults          cpu           time
-----
r  b  p   avm   fre  fi  fo  pi  po  fr  sr  in  sy  cs  us  sy  id  wa  hr  mi  se
0  0  0 51694 49443  6  3  0  0  8  48 106 199  64  0  1 96  3 17:43:55
0  0  0 51697 49440  0  0  0  0  0  0 469 991 332  0  0 99  0 17:43:57
0  0  0 51698 49439  0  0  0  0  0  0 468 980 320  0  1 99  0 17:43:59
0  0  0 51699 49438  0  0  0  0  0  0 468 989 327  0  0 99  0 17:44:01
0  0  0 51700 49437  0  0  0  0  0  0 470 992 331  0  0 99  0 17:44:03
0  0  0 51702 49435  0  0  0  0  0  0 471 989 327  0  1 99  0 17:44:05
0  0  0 51703 49434  0  0  0  0  0  0 469 993 329  0  0 99  0 17:44:08
```



```
0 0 0 51704 49433 0 0 0 0 0 0 471 969 320 0 0 99 0 17:44:10
0 0 0 51705 49432 0 0 0 0 0 0 468 986 325 0 1 99 0 17:44:12
0 0 0 51706 49431 0 0 0 0 0 0 470 995 331 0 0 99 0 17:44:14
```

Note: The first line of this report should be ignored because it is an average since the last system reboot.

Refer to 13.2.1, “Virtual memory activity” on page 213 for an explanation of report fields not listed here.

The reported fields are described as follows:

p	Number of threads waiting on actual physical I/O to raw logical volumes as opposed to files within a file system
fi	File page ins per second
fo	File page outs per second
hr	The hour that the last sample completed
mi	The minute that the last sample completed
se	The second that the last sample completed

Tip: It is useful to run `vmstat` when your system is under load and performing normally as a baseline to determine future performance problems.

You should run `vmstat` again when:

- ▶ Your system is experiencing performance problems.
- ▶ You make hardware or software changes to the system.
- ▶ You make changes to the AIX Operating System; for example, when installing upgrades or changing the disk tuning parameters using `vmo`, `ioo`, or `schedo`.
- ▶ You make changes to your application.
- ▶ Your average workload changes; for example, when you add or remove users.

Archived

The vmo, ioo, and vmtune commands

The vmtune sample program is being phased out and will not be supported in future releases. It is being replaced by the **vmo** command (for all pure VMM parameters) and the **ioo** command (for all I/O-related parameters) that can be used to set most of the parameters that were previously set by vmtune. For AIX 5L Version 5.2, a compatibility script calling **vmo** and **ioo** is provided to help the transition.

The **vmtune** script resides in /usr/samples/kernel and is part of the bos.adt.samples fileset, which is installable from the AIX base installation media.

The **vmo** and **ioo** commands reside in /usr/sbin and are part of the bos.perf.tune fileset, which is installable from the AIX base installation media.

14.1 vmo

The syntaxes of the **vmo** command are:

```
vmo [ -p | -r ] { -o Tunable [= Newvalue] }  
vmo [ -p | -r ] { -d Tunable }  
vmo [ -p | -r ] -D  
vmo [ -p | -r ] -a  
vmo -?  
vmo -h Tunable  
vmo -L [ Tunable ]  
vmo -x [ Tunable ]
```

Multiple options for -o, -d, and -L are allowed.

Flags

-?	Displays the vmo command usage statement.
-h Tunable	Displays help about the tunable parameter.
-a	Displays current, reboot (when used in conjunction with -r), or permanent (when used in conjunction with -p) value for all tunable parameters, one per line in pairs Tunable = Value.
-d Tunable	Resets tunable to its default value.
-D	Resets all tunables to their default value.
-o Tunable[=Newvalue]	Displays the value or sets tunable to newvalue.
-p	When used in combination with -o, -d, or -D, makes changes apply to both current and reboot values and updates the /etc/tunables/nextboot file in addition to the updating of the current value. These combinations cannot be used on Bosboot type parameters because their current value cannot be changed.
-r	When used in combination with -o, -d, or -D, makes changes apply to reboot values and updates the /etc/tunables/nextboot file. If any parameter of type Bosboot is changed, the user will be prompted to run bosboot .
-L [Tunable]	Lists the characteristics of one or all tunables, one per line, indicating the current, default, minimum, and maximum values and the tunable types.
-x [tunable]	Generates tunable characteristics in a comma-separated format for loading into a spreadsheet.

The current set of parameters managed by `vmo` only includes Dynamic and Bosboot types.

In the execution:

- ▶ Any attempt to change (with `-o`, `-d`, or `-D`) a parameter of type Bosboot without `-r`, will result in an error or warning message.
- ▶ Displaying a parameter (with `-a` or `-o`) with the `-p` displays a value when the current and reboot values are equal; otherwise NONE is displayed as the value.

14.1.1 Information about measurement and sampling

The `vmo` command is responsible for displaying and adjusting the parameters used by the Virtual Memory Manager (VMM). This command sets or displays current or next boot values for all Virtual Memory Manager tuning parameters. This command can also make permanent changes or defer changes until the next reboot. Whether the command sets or displays a parameter is determined by the accompanying flag. The `-o` flag performs both actions. It can either display the value of a parameter or set a new value for a parameter.

The Virtual Memory Manager (VMM) maintains a list of free real-memory page frames. These page frames are available to hold virtual-memory pages needed to satisfy a page fault. When the number of pages on the free list falls below that specified by the `minfree` parameter, the VMM begins to steal pages to add to the free list. The VMM continues to steal pages until the free list has at least the number of pages specified by the `maxfree` parameter.

If the number of file pages (permanent pages) in memory is less than the number specified by the `minperm%` parameter, the VMM steals frames from either computational or file pages, regardless of repage rates. If the number of file pages is greater than the number specified by the `maxperm%` parameter, the VMM steals frames only from file pages. Between the two, the VMM normally steals only file pages, but if the repage rate for file pages is higher than the repage rate for computational pages, computational pages are stolen as well.

You can also modify the thresholds that are used to decide when the system is running out of paging space. The `npswarn` parameter specifies the number of paging-space pages available at which the system begins warning processes that paging space is low. The `npskill` parameter specifies the number of paging-space pages available at which the system begins killing processes to release paging space.

Important: The `vmo` command is operating system version specific. Using the incorrect version of the `vmo` command can produce inconsistent results or result in the operating system becoming inoperable. Later versions of the OS also support new options that are unavailable on older versions.

Memory pools

The `mempools` value is used to subdivide the memory into pools. The parameter `mempools` has a range from 1 (one) to, but not more than, the value of the number of CPUs in the system. For example, if there are four CPUs in a system, then the maximum value of `mempools` is 4 (four). Setting the value to 0 (zero), restores the default number. In some circumstances, such as when most, but not all, of the system memory is in use, better performance can be obtained by setting this value to 1 (one).

The page replacement algorithm

When the number of pages on the free list is less than `minfree`, the page replacement algorithm attempts to free up memory pages. The algorithm continues until the number of pages in the free list exceeds the `maxfree` value.

The value of `minfree` specifies the minimum number of frames on the free list before the VMM starts to steal pages. The value can range from eight to 819200. The default value is dependant on the amount of memory in the system and is calculated as the `maxfree` value minus eight. In multiprocessor systems, there may be a number of memory pools. Each memory pool has its own `minfree` and `maxfree` value. The values displayed by the `vmo` command are the sum of the `minfree` and `maxfree` values of all of the pools.

The `maxfree` value determines at what point the VMM stops stealing pages. The value of `maxfree` can range from 16 to 204800 but must be greater than the value of `minfree`. The `maxfree` value can be determined as follows:

`maxfree = lesser of (number of memory pages / 128 or 128)`

For many systems, these default values may not be optimal. Assuming that the system has 512 MB of memory, the `minfree` and `maxfree` values are the defaults of 120 and 128 respectively. When only (4096 * 120) bytes of memory are on the free list, only then will the page replacement algorithm free pages. This value equates to less than 0.5 MB of memory and will typically be too low. If the memory demand continues after the `minfree` value is reached, then processes could even be suspended or killed. When the number of free pages equals or exceeds the value of `maxfree`, then the algorithm will no longer free pages. This value is (4096 * 128) bytes, which equates to 0.5 MB. As can be seen, insufficient pages will have been freed up on a system with 512 MB.

The page replacement algorithm subdivides the entire system real memory into sections called buckets. The `lrubucket` parameter specifies the number of pages per bucket. Instead of the page replacement algorithm checking the entire real memory of the system for free frames, it searches one bucket at a time. The page replacement algorithm searches a bucket for free frames and on the second pass checks the same bucket, and any unreferenced pages will be stolen. This speeds up the rate at which pages to be stolen are found. The default value for `lrubucket` is 131,072 pages, which equates to 512 MB of real memory.

Pinning memory

The `maxpin` value determines the maximum percentage of real memory pages that can be pinned. The `maxpin` value must be greater than one and less than 100. The default value for `maxpin` is 80 percent. Always ensure that the kernel and kernel extensions can pin enough memory as needed; as such, it is not advisable to set the `maxpin` value to an extremely low number such as one.

The `v_pinshm` parameter is a Boolean value that, if set to 1 (one), will force pages in shared memory to be pinned by the VMM. This occurs only if the application set the `SHM_PIN` flag. If the value is set to 0 (zero: the default), then shared memory is not pinned.

Note: Ensure that at least 4 MB of real memory is left unpinned for the kernel when the `maxpin` value is changed.

File system caching

The AIX operating system leaves in memory pages that have been read or written to. If these file pages are requested again, this saves an I/O operation. The `minperm` and `maxperm` values control the level of this file system caching. The thresholds set by `maxperm` and `minperm` can be considered as:

- ▶ If the percentage of file pages in memory exceeds `maxperm`, only file pages are taken by the page replacement algorithm.
- ▶ If the percentage of file pages in memory is less than `minperm`, both file pages and computational pages are taken by the page replacement algorithm.
- ▶ If the percentage of file pages in memory is in the range between `minperm` and `maxperm`, the page replacement algorithm steals only the file pages unless the number of file repages is higher than the number of computational repages.

Computational pages can be defined as working storage segments and program text segments. *File pages* are defined as all other page types usually persistent and client pages.

In some instances, the application may cache pages itself. Therefore there is no need for the file system to cache pages as well. In this case, the values of `minperm` and `maxperm` can be set low. For more information about adjusting these values, see “The page replacement algorithm” on page 232.

When set to 1 (one), the `strict_maxperm` value causes the `maxperm` parameter to be a hard limit. This parameter is very useful where double buffering occurs, such as in the case of a database on a JFS file system. The database may be doing its own caching while the VMM may be caching the same pages. When this value is set to 0 (zero), the `maxperm` value is only required when page replacements occur.

The `defps` parameter is used to enable or disable the Deferred Page Space Allocation (DPSA) policy. Setting this parameter to a value of 1 (one) enables DPSA, and setting it to 0 (zero) disables it. The DPSA policy can be disabled to prevent paging space from becoming overcommitted. With DPSA, the disk block allocation of paging space is delayed until it is necessary to page out the page, which results in no wasted paging space allocation. Paging space can, however, be wasted when a page in real memory needs to be paged out and then paged back in. That paging space will be reserved for this process until either the page is no longer required by the process or the process exits.

If `defps` is disabled, the Late Paging Space Allocation (LPSA) policy is used. Using the LPSA, paging space is only allocated if memory pages are touched (modified somehow). However, the paging space pages are not assigned to a process until the memory pages are paged out. A process might find no paging space available if another process uses all of the paging space because paging space was not allocated.

Large page parameters

The `lgpg_regions` value specifies the number of large pages to reserve. This is required when the `shmget()` call uses the `SHM_LGPAGE` flag. The application has to support `SHM_LGPAGE` when calling `shmget()`. This improves performance when there are many Translation Look-Aside Buffer (TLB) misses and large amounts of memory are being accessed.

The `lgpg_size` parameter sets the size in bytes of the hardware-dependant large pages used for the implementation of the `shmget()` system call. The `lgpg_size` and `lgpg_regions` parameters both must be set to enable this function.

JFS2 and NFS client pages

A new `maxclient%` option is available in AIX 5L Version 5.2. This option is tunable using the `vmo -o maxclient%=Number` command. This value determines the point at which the page replacement algorithm starts to free client pages. The value is a percentage of total memory. This value is important for JFS2 and NFS where client pages are used.

14.1.2 Recommendations and precautions for vmo

Do not attempt to use an incorrect version of the `vmo` command on an operating system. Invoking the incorrect version of the `vmo` command can result in the operating system failing. The functionality of the `vmo` command also varies between versions of the operating system.

14.2 Examples for vmo

Example 14-1 displays all reboot values for virtual memory tuning.

Example 14-1 Display all reboot values for virtual Memory Manager tuning parameters

```
#/usr/sbin/vmo -r -a
memory_frames = 2097152
maxfree = 128
minfree = 120
minperm% = 20
minperm = 408467
maxperm% = 80
maxperm = 1633868
strict_maxperm = 0
maxpin% = 80
maxpin = 1677722
maxclient% = 80
lrubucket = 131072
defps = 1
nokilluid = 0
numpsblks = 524288
npskill = 4096
npswarn = 16384
v_pinshm = 0
pta_balance_threshold = 50
pagecoloring = 0
framesets = 2
mempools = 1
lgpg_size = 16777216
lgpg_regions = 20
num_spec_dataseg = 0
```

```
spec_dataseg_int = 512
memory_affinity = 0
```

Example 14-2 shows the use of **vmo -L** to display the current, default, and reboot settings.

Example 14-2 Displaying tunable attributes using vmo -L

# vmo -L	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
NAME							
DEPENDENCIES							

memory_frames	768K		768K			4KB pages	S

pinnable_frames	677112		677112			4KB pages	S

maxfree	128	128	128	16	200K	4KB pages	D
minfree							
memory_frames							

minfree	120	120	120	8	200K	4KB pages	D
maxfree							
memory_frames							

minperm%	20	20	20	1	100	% memory	D
maxperm%							

minperm	137664		137664				S

maxperm%	80	80	80	1	100	% memory	D
minperm%							
maxclient%							

maxperm	550659		550659				S

strict_maxperm	0	0	0	0	1	boolean	D

maxpin%	80	80	80	1	99	% memory	D
pinnable_frames							
memory_frames							

maxpin	629146		629146				S

maxclient%	80	80	80	1	100	% memory	D
maxperm%							

lrubucket	128K	128K	128K	64K		4KB pages	D

defps	1	1	1	0	1	boolean	D

nokilluid	0	0	0	0	2047M	uid	D
numpsblks	128K		128K			4KB pages	S
npskill	1K	1K	1K	1	131071	4KB pages	D
npswarn	4K	4K	4K	0	131071	4KB pages	D
v_pinshm	0	0	0	0	1	boolean	D
pta_balance_threshold	n/a	50	50	1	99	% pta segment	R
pagecoloring	n/a	0	0	0	1	boolean	B
framesets	2	2	2	1	10		B
mempools	1	1	1	1	2		B
lgpg_size lgpg_regions	0	0	0	0	256M	bytes	B
lgpg_regions lgpg_size	0	0	0	0			B
num_spec_dataseg	0	0	0	0			B
spec_dataseg_int	512	512	512	0			B
memory_affinity	1	1	1	0	1	boolean	B

n/a means parameter not supported by the current platform or kernel

Parameter types:

- S = Static: cannot be changed
- D = Dynamic: can be freely changed
- B = Bosboot: can only be changed using bosboot and reboot
- R = Reboot: can only be changed during reboot
- C = Connect: changes are only effective for future socket connections
- M = Mount: changes are only effective for future mountings
- I = Incremental: can only be incremented

Value conventions:

- K = Kilo: 2^{10}
- M = Mega: 2^{20}
- G = Giga: 2^{30}
- T = Tera: 2^{40}
- P = Peta: 2^{50}
- E = Exa: 2^{60}

Example 14-3 shows the setting of mempools value and the message.

Example 14-3 Changing the mempools tunable

```
# /usr/sbin/vmo -o mempools
mempools = 0
# /usr/sbin/vmo -r -o mempools=4
Warning: bosboot must be called and the system rebooted for the mempools change
to take effect
Run bosboot now? [y/n] y

bosboot: Boot image is 16773 512 byte blocks.
Changes will take effect only at next reboot
# /usr/sbin/vmo -r -o mempools
mempools = 4
```

Example 14-4 shows the adjustment of minperm and maxperm values.

Example 14-4 Changing minperm and maxperm tunables

```
# vmo -o minperm% -o maxperm%
minperm% = 20
maxperm% = 50
# vmo -o minperm%=10 -o maxperm%=40
Setting minperm% to 10
Setting maxperm% to 40
```

Example 14-5 shows how to turn on v_pinshm for the next reboot.

Example 14-5 Turning on v_pinshm for the next reboot

```
# vmo -r -o v_pinshm=1
Setting v_pinshm to 1 in nextboot file
Changes will take effect only at next reboot
```

Example 14-6 shows the setting of 16 MB large pages.

Example 14-6 Reserving 16MB large pages

```
# vmo -r -o lgpg_regions=20 -o lgpg_size=16777216
Setting lgpg_size to 16777216 in nextboot file
Setting lgpg_regions to 20 in nextboot file
Warning: some changes will take effect only after a bosboot and a reboot
Run bosboot now? y

bosboot: Boot image is 18212 512 byte blocks.
Warning: changes will take effect only at next reboot
```

14.3 ioo

The following syntax applies to the **ioo** command:

```
ioo [ -p | -r ] { -o Tunable [ =NewValue ] }  
ioo [ -p | -r ] { -d Tunable }  
ioo [ -p | -r ] -D  
ioo [ -p | -r ] -a  
ioo -?  
ioo -h Tunable  
ioo -L [ Tunable ]  
ioo -x [ Tunable ]
```

Flags

- ?** Displays the **ioo** command usage statement.
- h** Displays help about the specified tunable parameter.
- a** Displays current, reboot (when used in conjunction with **-r**), or permanent (when used in conjunction with **-p**) value for all tunable parameters, one per line in pairs `tunable = value`. For the permanent option, a value is only displayed for a parameter if its reboot and current values are equal. Otherwise **NONE** displays as the value.
- d** Resets a tunable to its default value. If a tunable needs to be changed (that is, if it is not set to its default value) and is of type **Bosboot** or **Reboot**, or if it is of type **Incremental** and has been changed from its default value, and **-r** is not used in combination, it is not changed but a warning displays.
- D** Resets all tunables to their default value. If tunables needing to be changed are of type **Bosboot** or **Reboot**, or are of type **Incremental** and have been changed from their default value, and **-r** is not used in combination, they are not changed but a warning displays.
- o Tunable[=Newvalue]** Displays the value or sets tunable to `newvalue`. If a tunable needs to be changed (because the specified value is different from the current value), and is of type **Bosboot** or **Reboot**, or if it is of type **Incremental** and its current value is bigger than the specified value, and **-r** is not used in combination, it will not be changed but a warning will be displayed instead.
- When **-r** is used in combination without a new value, the `nextboot` value for tunable is displayed. When **-p** is used in

combination without a new value, a value is displayed only if the current and next boot values for tunable are the same. Otherwise NONE is displayed as the value.

- p** When used in combination with -o, -d, or -D, makes changes apply to both current and reboot values (that is, turns on the updating of the /etc/tunables/nextboot file in addition to the updating of the current value). These combinations cannot be used on Reboot and Bosboot type parameters because their current value cannot be changed.

When used with -a or -o without specifying a new value, values are displayed only if the current and next boot values for a parameter are the same. Otherwise NONE is displayed as the value.
- r** When used in combination with -o, -d, or -D, makes changes apply to reboot values (for example, turns on the updating of the /etc/tunables/nextboot file). If any parameter of type Bosboot is changed, the user will be prompted to run **bosboot**.

When used with -a or -o without specifying a new value, next boot values for tunables are displayed instead of current values.
- L** Lists the characteristics of one or all tunables, one per line, indicating the current, default, minimum and maximum values and the tunable types
- x [tunable]** Generates tunable characteristics in a comma-separated format for loading into a spreadsheet.

The current set of parameters managed by **ioo** only includes Dynamic, Incremental, and Mount types. In the execution:

- ▶ Any change (with -o, -d or -D) to a parameter of type Mount will result in a message being displayed to warn the user that the change is only effective for future mountings.
- ▶ Any attempt to change (with -o, -d, or -D) a parameter of type Bosboot or Reboot without -r will result in an error message.
- ▶ Any attempt to change (with -o, -d, or -D but without -r) the current value of a parameter of type Incremental with a new value smaller than the current value will result in an error message.
- ▶ Displaying a parameter (with -a or -o) with the -p displays a value when the current and reboot values are equal, otherwise NONE is displayed as the value.

14.3.1 Information about measurement and sampling

The `ioo` command sets or displays current or next boot values for all input/output tuning parameters. This command can also make permanent changes or defer changes until the next reboot. Whether the command sets or displays a parameter is determined by the accompanying flag. The `-o` flag performs both actions. It can either display the value of a parameter or set a new value for a parameter.

If a process appears to be reading sequentially from a file, the values specified by the `minpgahead` parameter determine the number of pages to be read ahead when the condition is first detected. The value specified by the `maxpgahead` parameter sets the maximum number of pages that are read ahead, regardless of the number of preceding sequential reads.

The operating system enables tuning of the number of file system bufstructs (`numfsbuf`) and the amount of data processed by the write-behind algorithm (`numclust`).

Important: The `ioo` command is operating system version specific. Using the incorrect version of the `ioo` command can produce inconsistent results or result in the OS becoming inoperable. Later versions of the OS also support new options that are unavailable on older versions.

The default `ioo` values may differ on different machine configurations as well as on different AIX releases. The machine's workload and the effects of the `ioo` tunables should be considered before changing anything.

Sequential read-ahead

The `minpgahead` value is the value at which sequential read-ahead begins. The value can range from 0 (zero) to 4096, and must be a power of two. The default value is 2 (two).

The `maxpgahead` is the maximum number of pages that can be read ahead. The value of `maxpgahead` can be in the range of zero to 4096. The value must be equal to or greater than `minpgahead`. The default value is 8 (eight).

Figure 14-1 on page 242 shows an illustration of sequential read ahead. Each of the blocks in the diagram represents a 4 KB page. These pages are numbered zero through 23. The steps of sequential read-ahead are described under the labels A through F. The labels A through F also indicate the sequence of page

reads. Pages are read ahead when the VMM detects a sequential pattern. Read ahead is triggered again when the first page in a group of previously read ahead pages is accessed by the application. In the example, minpgahead is set to 2 (two) while maxpgahead is set to 8 (eight).

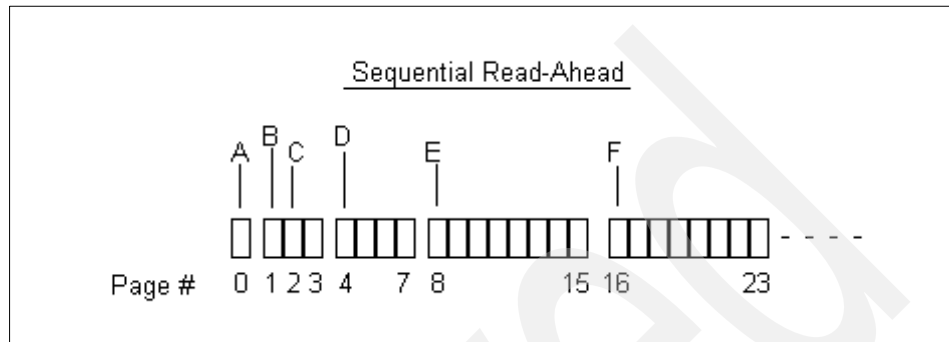


Figure 14-1 Sequential read-ahead

- A** The first page of the file is read in by the program. After this operation, VMM makes no assumptions as to whether the file access is random or sequential.
- B** When page number one is the next page read in by the program, VMM assumes that access is sequential. VMM schedules minpgahead pages to be read in as well. Therefore the access at point B in Figure 14-1 results in three pages being read.
- C** When the program accesses page two next, VMM doubles the value of page ahead from two to four and schedules the pages four to seven to be read.
- D** When the program accesses page four next, VMM doubles the value of page ahead from four to eight, and pages eight through 15 are scheduled to be read.
- E** When the program accesses page eight next, VMM determines that the read-ahead value is equal to maxpgahead and schedules pages 16 through 23 to be read.
- F** VMM will continue to read maxpgahead pages ahead as long as the program accesses the first page of the previous read-ahead group. Sequential read-ahead will be terminated when the program accesses a page other than the first page of the next read-ahead group.

If the program were to deviate from the sequential-access pattern and access a page of the file out of order, sequential read-ahead would be terminated. It would be resumed with minpgahead pages if the VMM detected that the program resumed sequential access.

The minpgahead and maxpgahead values can be changed by using options -o in the ioo command. If you are contemplating changing these values, keep in mind:

- ▶ The values should be from the set: 0, 1, 2, 4, 8, 16, and so on. The use of other values may have adverse performance or functional effects.
 - Values should be powers of 2 because of the doubling algorithm of the VMM.
 - Values of maxpgahead greater than 16 (reads ahead of more than 64 KB) exceed the capabilities of some disk device drivers. In such a case, the read size stays at 64 KB.
 - Higher values of maxpgahead can be used in systems where the sequential performance of striped logical volumes is of paramount importance.
- ▶ A minpgahead value of 0 effectively defeats the mechanism. This can adversely affect performance. However, it can be useful in some cases where I/O is random, but the size of the I/Os cause the VMM's read-ahead algorithm to take effect. Another case where turning off page-ahead is useful is the case of NFS reads on files that are locked. On these types of files, read-ahead pages are typically flushed by NFS so that reading ahead is not helpful. NFS and the VMM have been changed, starting with AIX 4.3.3, to automatically turn off VMM read-ahead if it is operating on a locked file.
- ▶ The maxpgahead values of 8 or 16 yield the maximum possible sequential I/O performance for non-striped file systems.
- ▶ The buildup of the read-ahead value from minpgahead to maxpgahead is quick enough that for most file sizes there is no advantage to increasing minpgahead.
- ▶ The Sequential Read-Ahead can be tuned separately for JFS and Enhanced JFS. JFS Page Read-Ahead can be tuned with minpgahead and maxpgahead whereas j2_minPageReadAhead and j2_maxPageReadAhead are used for Enhanced JFS.

Note: Due to limitations in the kernel, the maxpgahead value should not exceed 512. The difference between minfree and maxfree should always be equal to or greater than the value of maxpgahead.

VMM write-behind

Write-behind involves asynchronously writing modified pages in memory to disk after reaching a threshold rather than waiting for the **syncd** daemon to flush the pages to disk. This is done to limit the number of dirty pages in memory, reduce system overhead, and minimize disk fragmentation. There are two types of write-behind: sequential and random.

Sequential write-behind

The **numclust** value determines the number of 16 KB clusters to be processed by the VMM sequential write-behind algorithm. The value can be set as an integer greater than zero. The default value is one. The write-behind algorithm will write modified pages in memory to disk after the threshold set by **numclust** is reached rather than waiting for the **syncd** daemon to flush the pages if the write pattern is sequential. The advantages of using the write-behind algorithm are:

- ▶ The algorithm reduces the number of dirty pages in memory.
- ▶ It reduces the system overhead because the **syncd** daemon will have fewer pages to write to disk.
- ▶ It minimizes disk fragmentation because entire clusters are written to the disk at a time.

The **numclust** values can be changed by using **-o** options in the **ioo** command.

For enhanced JFS, the **j2_nPagesPerWriteBehindCluster** value is used to specify the number of pages to be scheduled at one time, rather than the number of clusters. The default number of pages is 8.

The **j2_nPagesPerWriteBehindCluster** values can be changed by using **-o** options in the **ioo** command.

Random write-behind

The **maxrandwrt** option specifies the threshold number of pages for random page writes to accumulate in real memory before being flushed to disk by the write-behind algorithm. The default value for **maxrandwrt** is zero, which disables the random write-behind algorithm. Applications may write randomly to memory pages. In this instance, the sequential page write-behind algorithm will not be able to flush dirty memory pages to disk. If the application has written a large number of pages to memory, then when the **syncd** daemon flushes memory to disk, the disk I/O may become excessive. To counter this effect, the random write-behind algorithm will wait until the number of pages modified for a file exceeds the **maxrandwrt** threshold. From this point, all subsequent dirty pages are scheduled to be written to disk. The pages below the **maxrandwrt** are flushed to disk by the **syncd** daemon.

The **maxrandwrt** values can be changed by using **-o** options in the **ioo** command.

Note: Not all applications meet the requirements for random and sequential write-behind. In this instance, the **syncd** daemon will flush dirty memory pages to disk.

For enhanced JFS, the `j2_nRandomCluster` and `j2_maxRandomWrite` values are used to tune random write-behind. Both options have a default of 0. The `j2_maxRandomWrite` option has the same function for enhanced JFS as `maxrandwrt` does for JFS. That is, it specifies a limit for the number of dirty pages per file that can remain in memory. The `j2_nRandomCluster` option specifies how many clusters apart two consecutive writes must be in order to be considered random.

The `j2_nRandomCluster` and `j2_maxRandomWrite` values can be changed by using `-o` options in the `ioo` command.

The syncd daemon

The default value of the `sync_release_ilock` is 0 (zero). At zero, the inode lock will be held and the data is flushed and committed, and only then is the lock released. If the `sync_release_ilock` is set to a non-zero value, then the `syncd` daemon will flush all dirty memory pages to disk without using the inode lock. The lock is then used to commit the data. This minimizes the time that the inode lock is held during the sync operation. This is a Boolean variable; setting it to 0 (zero) disables it, and any other non-zero value enables it. A performance improvement may be achieved if the `sync_release_ilock` parameter is set to a value of 1 (one) on systems with a large amount of memory and a large number of page updates. These types of systems typically have high I/O peaks when the `syncd` daemon flushes memory.

The `sync_release_ilock` values can be changed by using `-o` options in the `ioo` command.

I/O tuning parameters

The `numfsbufs` value specifies the number of file system buffer structures. This value must be greater than 0 (zero). If there are insufficient free buffer structures, the VMM will put the process on a wait list before starting I/O. To determine whether the value of `numfsbufs` is too low, use the `vmstat -a` command and monitor the `fsbufwaitcnt` value displayed. This value is incremented each time an I/O operation has to wait for a file system buffer structure.

Note: When the `numfsbufs` value is changed, it is necessary to **unmount** and **mount** the file system again for the changes to take effect.

The `j2_nBufferPerPagerDevice` value specifies the number of file system bufstructs for Enhanced JFS. If the kernel must wait for a free bufstruct, it puts the process on a wait list before the start I/O is issued and will wake it up once a bufstruct has become available. May be appropriate to increase if striped logical volumes or disk arrays are being used. To determine whether it is necessary for the value of `j2_nBufferPerPagerDevice` to change, use the `vmstat -v` command and monitor if the `xpagerbufwaitcnt` increases fast. The default value is 512.

The `lvm_bufcnt` value specifies the number of LVM buffers for *raw* I/O. This value can range from 1 (one) to 64 and has a default of 9 (nine). Extremely large volumes of I/O are required to cause a bottleneck at the LVM layer. The number of “uphysio” buffers can be increased to overcome this bottleneck. Each uphysio buffer is 128 KB. If I/O operations are larger than 128 KB * 9, then a value larger than the default value of nine should be used.

The `pd_npages` value determines the number of pages that should be deleted in one chunk from real memory when a file is deleted (that is, the pages are deleted in a single VMM critical section with interrupts disabled to INTPAGER). By default, all pages of a file can be removed from memory in one critical section if the file was deleted from disk. To ensure fast response time for real-time applications, this value can be reduced so that a smaller chunk of pages is deleted before returning from the critical section.

The `hd_pbuf_cnt` value determines the number of pbufs assigned to the LVM. This value is sometimes referred to as `numpbuf`. The pbufs are pinned memory buffers used to hold I/O requests that are pending at the LVM layer. When changing this value, the new value must be higher than the previously set value. The value can only be reset by a reboot.

Note: If the value of `hd_pbuf_cnt` is set too high, the only way to reset the value is with a reboot. The value cannot be set lower than the current value.

14.3.2 Recommendations and precautions

Do not attempt to use an incorrect version of the `ioo` command on an operating system. Invoking the incorrect version of the `ioo` command can result in failure of the operating system. The functionality of the `ioo` command also varies between versions of the operating system.

14.4 Examples for ioo

This section shows some examples of the use of the `ioo` command.

14.4.1 Displaying I/O setting

Example 14-7 shows all of the tunable values and characteristics using the `ioo` command.

Example 14-7 Showing ioo tunables characteristics

```
# /usr/sbin/ioo -L
```

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							
minpgahead	2	2	2	0	4K	4KB pages	D
maxpgahead	8	8	8	0	4K	4KB pages	D
pd_npages	64K	64K	64K	1	512K	4KB pages	D
maxrandwrt	0	0	0	0	512K	4KB pages	D
numclust	1	1	1	0	2047M	16KB/cluster	D
numfsbufs	196	196	196	1	2047M		M
sync_release_ilock	0	0	0	0	1	boolean	D
lvm_bufcnt	9	9	9	1	64	128KB/buffer	D
j2_minPageReadAhead	2	2	2	0	128	4KB pages	D
j2_maxPageReadAhead	8	8	8	0	128	4KB pages	D
j2_nBufferPerPageDevice	512	512	512	0	2047M		M
j2_nPagesPerWriteBehindCluster	32	32	32	0	128		D
j2_maxRandomWrite	0	0	0	0	128	4KB pages	D
j2_nRandomCluster	0	0	0	0	2047M	16KB clusters	D
hd_pvs_opn	1		1				S
hd_pbuf_cnt	640	640	640	0	2047M		I

n/a means parameter not supported by the current platform or kernel

Parameter types:

S = Static: cannot be changed
D = Dynamic: can be freely changed
B = Bosboot: can only be changed using bosboot and reboot
R = Reboot: can only be changed during reboot
C = Connect: changes are only effective for future socket connections
M = Mount: changes are only effective for future mountings
I = Incremental: can only be incremented

Value conventions:

K = Kilo: 2¹⁰ G = Giga: 2³⁰ P = Peta: 2⁵⁰
M = Mega: 2²⁰ T = Tera: 2⁴⁰ E = Exa: 2⁶⁰

Example 14-8 shows all of the reboot values for **ioo** that will be used on the next boot of the system.

Example 14-8 Showing all reboot values for ioo

```
# ioo -r -a
.....minpgahead = 2
                maxpgahead = 8
                pd_npages = 65536
                maxrandwrt = 0
                numclust = 1
                numfsbufs = 186
                sync_release_ilock = 0
                lvm_bufcnt = 9
                j2_minPageReadAhead = 2
                j2_maxPageReadAhead = 8
                j2_nBufferPerPagerDevice = 512
j2_nPagesPerWriteBehindCluster = 32
                j2_maxRandomWrite = 0
                j2_nRandomCluster = 0
                hd_pvs_opn = 2
                hd_pbuf_cnt = 384
```

Specific help on each tunable can be displayed using the **-h** flag as shown in Example 14-9.

Example 14-9 Displaying help on j2_nPagesPerWriteBehindCluster

```
# ioo -h j2_nPagesPerWriteBehindCluster
Specifies the number of pages per cluster processed by Enhanced JFS's write
behind algorithm. Default: 8. Useful to increase if there is a need to keep
more pages in RAM before scheduling them for I/O when the I/O pattern is
sequential. May be appropriate to increase if stripped logical volumes or disk
arrays are being used.
```

14.4.2 Changing tunable values

You can set dynamic tunables using the `-o` option. Example 14-10 shows that the `sync_release_ilock` is turned on dynamically.

Example 14-10 Activating `sync_release_ilock`

```
# ioo -o sync_release_ilock=1
Setting sync_release_ilock to 1
```

Sometimes you may want to defer the tunable changes to the next reboot as shown in Example 14-11 where we set the `maxrandwrt` to 4.

Example 14-11 Setting `maxrandwrt` to 4 after the next reboot

```
# ioo -r -o maxrandwrt=4
Setting maxrandwrt to 4 in nextboot file
Warning: changes will take effect only at next reboot
```

Example 14-12 shows resetting all tunables to default value.

Example 14-12 Restoring all `ioo` tunable parameters to default

```
# ioo -p -D
Setting minpgahead to 2 in nextboot file
Setting maxpgahead to 8 in nextboot file
Setting pd_npages to 65536 in nextboot file
Setting maxrandwrt to 0 in nextboot file
Setting numclust to 1 in nextboot file
Setting numfsbufs to 196 in nextboot file
Setting sync_release_ilock to 0 in nextboot file
Setting lvm_bufcnt to 9 in nextboot file
Setting j2_minPageReadAhead to 2 in nextboot file
Setting j2_maxPageReadAhead to 8 in nextboot file
Setting j2_nBufferPerPagerDevice to 512 in nextboot file
Setting j2_nPagesPerWriteBehindCluster to 32 in nextboot file
Setting j2_maxRandomWrite to 0 in nextboot file
Setting j2_nRandomCluster to 0 in nextboot file
Setting hd_pbuf_cnt to 640 in nextboot file
Setting sync_release_ilock to 0
```

14.4.3 Logical volume striping

The following provides suggestions about `ioo` and logical volume striping. Sequential and random accesses benefit from disk striping. The following technique for configuring striped disks is recommended:

- ▶ Spread the logical volume across as many physical volumes as possible.
- ▶ Use as many adapters as possible for the physical volumes.

- ▶ Create a separate volume group for striped logical volumes.
- ▶ Do not mix striped and non-striped logical volumes in the same physical volume.
- ▶ All physical volumes should be the same size within a set of striped logical volumes.
- ▶ Set the stripe unit size to 64 KB.
- ▶ Set the value of minpgahead to 2 (two).
- ▶ Set the value of maxpgahead to 16 times the number of disks.
- ▶ Ensure that the difference between maxfree and minfree is equal to or exceeds the value of maxpgahead.

Setting the minpgahead and maxpgahead values as noted causes page-ahead to be done in units of the stripe-unit size, which is 64 KB times the number of disk drives, resulting in the reading of one stripe unit from each disk drive for each read-ahead operation. We will also need to set the minfree and maxfree tunables.

First, we acquire the tunable values as shown in Example 14-13.

Example 14-13 Displaying minpgahead, maxpgahead, minfree, and maxfree values

```
# ioo -L minpgahead -L maxpgahead
```

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							
minpgahead	2	2	2	0	4K	4KB pages	D
maxpgahead							
DEPENDENCIES							
maxpgahead	8	8	8	0	4K	4KB pages	D
minpgahead							
DEPENDENCIES							

```
# vmo -L minfree -L maxfree
```

NAME	CUR	DEF	BOOT	MIN	MAX	UNIT	TYPE
DEPENDENCIES							
maxfree	128	128	128	16	200K	4KB pages	D
minfree							
memory_frames							
DEPENDENCIES							
minfree	120	120	120	8	200K	4KB pages	D
maxfree							
memory_frames							
DEPENDENCIES							

Assuming that three disks are to be striped, the commands in Example 14-14 are used to set the **ioo** and **vmo** parameters.

Example 14-14 Setting ioo minpgahead and maxpgahead values

```
# ioo -o minpgahead=2
Setting minpgahead to 2
# ioo -o maxpgahead=32
Setting maxpgahead to 32
# vmo -o maxfree=152
Setting maxfree to 152
```

14.4.4 Increasing write activity throughput

If the striped logical volumes are on raw logical volumes and writes larger than 1.125 MB are anticipated, the value of the **lvm_bufcnt** parameter should be increased with the command **ioo -o lvm_bufcnt=10** in order to increase throughput of the write activity. This is shown in Example 14-15.

Example 14-15 Increasing lvm_bufcnt with the ioo command

```
# ioo -o lvm_bufcnt=10
Setting lvm_bufcnt to 10
```

14.5 vmtune

Note: This command in AIX 5L Version 5.2 is just a sample compatibility script that calls the **vmo** or **ioo** commands.

The syntax of the **vmtune** command is:

```
vmtune [ -a ]
vmtune [ -A ]
vmtune [ -b numfsbuf ] [ -B hd_pbuf_cnt ] [ -c numclust ] [ -C 0 | 1 ]
[ -d 0 | 1 ] [ -f minfree ] [ -F maxfree ] [ -g lpgg_regions ] [ -h 0 | 1 ]
[ -i Number ] [ -j Number ] [ -k npskill ] [ -l lrubucket ] [ -L Number ]
[ -m mempools ] [ -M maxpin ] [ -n nokilluid ] [ -N pd_npages ]
[ -p minperm% ] [ -P maxperm% ] [ -q Number ] [ -Q Number ] [ -r MinPgAhead ]
[ -R MaxPgAhead ] [ -s 0|1 ] [ -S 0 | 1 ] [ -t maxclient% ] [ -T Number ]
[ -u lvm_bufcnt ] [ -v framesets ] [ -V Number ] [ -w npswarn ]
[ -W maxrandwrt ] [ -y 0|1 ] [ -z Number ] [ -Z Number ] [ -? ]
```

Flags

-a Calls **vmo -a** and **ioo -a** to display the current values for all statistic counters.

-A	Calls vmstat -v to display the current statistic counters.
-b Number	Calls ioo -o numfsbuf=Number to set the number of file systems.
-B Number	Calls ioo -o hd_pbuf_cnt=Number to set the number of pbufs used by the LVM.
-c Number	Calls ioo -o numclust=Number to set the number of 16 KB clusters processed by write behind.
-C [0 1]	0 1 accepted, but not directly supported. Use vmo -r -o pagecoloring= 0 1 to disable/enable page coloring for specific hardware platforms.
-d [0 1]	0 1 calls vmo -o defps=0 1 to turn on and off deferred paging space allocation.
-f Number	Calls vmo -o minfree=Number to set the number of frames on the free list.
-F Number	Calls vmo -o maxfree=Number to set the number of frames on the free list at which stealing is to stop.
-g Number	Use vmo -r -o lpgg_regions=Number to set the size, in bytes, of the hardware-supported large pages.
-h [0 0]	Calls vmo -o strict_maxperm=0 1 to specify whether maxperm% should be hard limit.
-i Number	Number accepted, but not directly supported. Use vmo -r -o spec_dataseg_int=Number to set the interval to use when reserving the special data segidentifiers.
-j Number	Calls ioo -o j2_nPagesPerWriteBehindCluster=Number to set the number of pages per write-behind cluster.
-J Number	Calls ioo -o j2_maxRandomWrite=Number to set the random-write threshold count.
-k Number	Calls vmo -o npskill=Number to set the number of paging space pages at which processes begin to be killed.
-l Number	Calls vmo -o lruBucket=Number to set the size of the least recently used page replacement bucket size.
-L Number	Accepted, but not directly supported. Use vmo -r -o lpgg_Region= Number -o lpgg_size=Size to set the number of large pages to be reserved.
-m Number	Accepted, but not directly supported. Use vmo -r -o mempools=Number to set the number of memory pools.
-M	Calls vmo -omaxpin=Number to set the maximum percentage of real memory that can be pinned.

-n Number	Calls <code>vmo -o nokilluid=Number</code> to specify the uid range of processes that should not be killed when paging space is low.
-N Number	Calls <code>ioo -o pd_npages=Number</code> to set the number of pages that should be deleted in one chunk from RAM when a file is deleted.
-p Number	Calls <code>vmo -o minperm%=Number</code> to set the point below which file pages are protected from the repage algorithm.
-P Number	Calls <code>vmo -o maxperm%=Number</code> to set the point above which the page stealing algorithm steals only file pages.
-q Number	Calls <code>ioo -o j2_minPageReadAhead=Number</code> to set the minimum number of pages to read ahead.
-Q Number	Calls <code>ioo -o j2_maxPageReadAhead=Number</code> to set the maximum number of pages to read ahead.
-r Number	Calls <code>ioo -o minpageahead=Number</code> to set the number of pages with which sequential read-ahead starts.
-R Number	Calls <code>ioo -o maxpageahead=Number</code> to set the minimum number of pages to be read ahead.
-s [0 1]	Calls <code>ioo -o sync_release_illock=0 1</code> to enable the code that minimizes the time spent holding inode lock during sync.
-S [0 1]	Calls <code>vmo -o v_pinshm=0 1</code> to enable the SHM_PIN flag on the shmget system call.
-t Number	Calls <code>vmo -o maxclient%=Number</code> to set the point above which the page stealing algorithm steals only client file pages.
-T Number	Calls <code>vmo -o pta_balance_threshold=Number</code> to set the point at which a new pta segment will be allocated.
-u Number	Calls <code>vmo -o lvm_bufcnt=Number</code> to set the number of LVM buffers for raw physical I/Os.
-v Number	Accepted, but not directly supported. Use <code>vmo -r -o framesets= Number</code> to set the number of framesets per mempool.
-V Number	Accepted, but not directly supported. Use <code>vmo -r -o num_spec_dataseq= Number</code> to set the number of reserved special data segment IDs.

-w Number	Calls <code>vmo -o npswarn=Number</code> to set a threshold for random writes to accumulate in RAM before pages are synced to disk using a write-behind algorithm.
-W Number	Calls <code>ioo -o maxrandwrt=Number</code> to set the number of free paging-space pages at which SIGDANGER is sent to processes.
-y [0 1]	Use <code>vmo -r -o Memory_Affinity=[0 1]</code> to enable memory affinity on certain hardware.
-z Number	Calls <code>ioo -o j2_nRandonCluster=Number</code> to set random write threshold distance.
-Z Number	Calls <code>ioo -o j2_nBufferPerPagerDevice=Number</code> to set the number of buffers per pager device.
-?	Displays a description of the command and its flags.

Kernel tunables commands

This chapter discusses commands for manipulating kernel tunable files. These commands are supported for AIX 5L Version 5.2. The commands discussed here are:

tuncheck	Used to validate a file.
tunrestore	Used to restore all parameters from a file.
tunsave	Used to save current tunable parameter values into a file.
tundefault	Used to force all tuning parameters to be reset to their default value.
tunchange	Used to update a stanza in the tunables file.

The **tunsave**, **tunrestore**, **tuncheck**, **tundefault**, and **tunchange** commands reside in /usr/sbin and are part of the bos.perf.tune fileset, which is installable from the AIX base installation media.

For discussion on kernel tunables refer to 1.6, “Kernel tunables” on page 43.

15.1 tuncheck

The syntax of the **tuncheck** command is:

```
tuncheck [-r|-p] -f Filename
```

Flags

- r** Checks filename in a boot context.
- p** Checks filename in both current and boot context.
- f Filename** Specifies the name of the tunable file to be checked.

If **-p** or **-r** are not specified, **Filename** is checked according to the current context.

The **tuncheck** command is used to validate a tunable file. All tunables listed in the specified file are checked for range and dependencies. If a problem is detected, a warning is issued.

There are two types of validation:

current context Checks to see whether the tunable file could be applied immediately. Tunables not listed in **Filename** are interpreted as current values. The checking fails if a tunable of type **Incremental** is listed with a smaller value than its current value; it also fails if a tunable of type **Bosboot** or **Reboot** is listed with a different value than its current value.

next boot context Checks to see whether the tunable file could be applied during a reboot, that is, if it could be a valid nextboot file. Decreasing a tunable of type **Incremental** is allowed. If a tunable of type **Bosboot** or **Reboot** is listed with a different value than its current value, a warning is issued but the checking does not fail.

Additionally, warnings are issued if the tunable file contains unknown stanzas, or unknown tunables in a known stanza. However, that does not make the checking fail.

15.1.1 Examples for tuncheck

Example 15-1 shows that nextboot file can be applied immediately.

Example 15-1 Checking nextboot file using tuncheck -f command

```
# tuncheck -f nextboot
Checking successful
```

Example 15-2 shows that the nextboot file can be applied during a reboot.

Example 15-2 Checking nextboot file using tuncheck -r -f command

```
# tuncheck -r -f nextboot
Checking successful
```

The content of the mytunable file is shown in Example 15-3.

Example 15-3 Content of the mytunable file

```
info:
    AIX_level = "5.2.0.5"
    Kernel_type = "MP64"
    Last_validation = "2003-04-22 12:04:26 CDT (current, reboot)"

vmo:
    maxfree = "128"
    minfree = "120"
    maxperm%= "50"
    maxclient%="60"

ioo:
    maxpgahead = "8"

no:
    ipforwarding = "0"

nfso:
    nfs_v2_vm_bufs = "5000"
```

In Example 15-4, we use the mytunable file from Example 15-3 to check whether this file can be applied immediately and after a reboot. The **tuncheck** command issued a message because dependencies exist between **maxperm%** and **maxclient%** tunable parameter. The other tuning parameters were done successfully.

Example 15-4 Using tuncheck -p -f command to check a tunable file

```
# tuncheck -p -f mytunable
Setting maxpgahead to 8 in nextboot file
Setting maxpgahead to 8
invalid tunable value 50
value for tunable maxperm% must be greater than or equal to value of maxclient%
tunable
Setting maxfree to 128 in nextboot file
Setting minfree to 120 in nextboot file
Setting maxfree to 128
Setting minfree to 120
Checking failed
```

Messages should have been provided

We changed the `maxclient` value from 60 to 50 to resolve the dependency. The `tuncheck` command shows that all parameters were checked successfully, as shown in Example 15-5.

Example 15-5 Using `tuncheck -p -f` command with a new `maxclient` value

```
# tuncheck -p -f mytunable
Setting maxpgahead to 8 in nextboot file
Setting maxpgahead to 8
Setting maxfree to 128 in nextboot file
Setting minfree to 120 in nextboot file
Setting maxperm% to 50 in nextboot file
Setting maxclient% to 50 in nextboot file
Setting maxfree to 128
Setting minfree to 120
Setting maxperm% to 50
Setting maxclient% to 50
Checking successful
```

Note: If you create a tunable file with an editor or by copying a file from another machine, you must run the `tuncheck` command to validate it.

15.2 tunrestore

The `tunrestore` command is used to restore all tunable parameters values from a file in `/etc/tunables`. The syntax of the `tunrestore` command is:

```
tunrestore [-r] -f Filename
tunrestore -R
```

Flags

- | | |
|--------------------|--|
| -r | Checks filename in a boot context. |
| -f Filename | Specifies the name of the tunable file to be checked. |
| -R | Restores <code>/etc/tunables/nextboot</code> during boot process; can only be run from <code>/etc/inittab</code> |

Note: The command `tunrestore -R` can only be called from `/etc/inittab`

A new tunable file called `/etc/tunables/lastboot` is automatically generated after a reboot. That file has all the tunables listed with numerical values. The values representing default values are marked with the comment `DEFAULT VALUE`. Its info stanza includes the checksum of the `/etc/tunables/lastboot.log` file to ensure that pairs of `lastboot` and `lastboot.log` files can be identified and verified.

Any problem found or change made is logged in the `/etc/tunables/lastboot.log` file. A new `/etc/tunables/lastboot` file is always created with the list of current values for all parameters.

If filename does not exist, an error message displays. If the `nextboot` file does not exist, an error message displays if `-r` was used. If `-R` was used, all of the tuning parameters of a type other than `Bosboot` will be set to their default value, and a `nextboot` file containing only an info stanza will be created. A warning will also be logged in the `lastboot.log` file.

Except when `-r` is used, parameters requiring a call to `bosboot` and a reboot are not changed, but an error message is displayed to indicate that they could not be changed. When `-r` is used, if any parameter of type `Bosboot` needs to be changed, the user will be prompted to run `bosboot`. Parameters missing from the file are simply left unchanged, except when `-R` is used, in which case missing parameters are set to their default values. If the file contains multiple entries for a parameter, only the first entry will be applied, and a warning will be displayed or logged (if called with `-R`).

15.2.1 Examples for `tunrestore`

In Example 15-6 we restored the system with all tunable values in the `/etc/tunables/mytunable` file (shown in Example 15-3 on page 257) with the `maxclient%` changed to 50.

Example 15-6 Using `tunrestore -f` command

```
lpar05:/etc/tunables>> tunrestore -f mytunable
Setting maxfree to 128
Setting minfree to 120
Setting maxperm% to 50
Setting maxclient% to 50
Setting maxpgahead to 8
```

Example 15-7 shows how to validate the `/etc/tunables/mytunable` file and make it the new `nextboot` file.

Example 15-7 Using `tunrestore -r -f` command

```
lpar05:/etc/tunables>> tunrestore -r -f mytunable
Setting maxpgahead to 8 in nextboot file
Changes will take effect only at next reboot
```

```
Setting maxfree to 128 in nextboot file
Setting minfree to 120 in nextboot file
Setting maxperm% to 50 in nextboot file
Setting maxclient% to 50 in nextboot file
Changes will take effect only at next reboot
Checking successful
```

Example 15-8 shows the content of the new nextboot file.

Example 15-8 Example nextboot file after tunrestore -f -r mytunable

```
info:
    AIX_level = "5.2.0.5"
    Kernel_type = "MP64"
    Last_validation = "2003-04-22 15:47:22 CDT (reboot)"

vmo:
    maxfree = "128"
    minfree = "120"
    maxperm%= "50"
    maxclient%= "50"

ioo:
    maxpgahead = "8"

no:
    ipforwarding = "0"

nfso:
    nfs_v2_vm_bufs = "5000"
```

15.3 tunsave

The **tunsave** command saves all tunable parameter values into a file.

The syntax of the **tunsave** command is:

```
tunsave [-a|-A] -f|-F Filename [-d Description]
```

Flags

- a** Saves all tunable parameters, including those that are set to their default value. These parameters are saved with the special value DEFAULT.
- A** Saves all tunable parameters, including those that are set to their default value. These parameters are saved

numerically, and a comment (# DEFAULT VALUE) is appended to the line to flag them.

- d** (Description) Specifies the text to use for the Description field. Special characters must be escaped or quoted inside the Description field.
- f** (Filename) Specifies the name of the tunable file where the tunable parameters are saved. If Filename already exists, an error message prints. The Filename is relative to /etc/tunables.
- F** (Filename) Specifies the name of the tunable file where the tunable parameters are saved. If Filename already exists, the existing file is overwritten. The Filename is relative to /etc/tunables.

15.3.1 Examples for tunsave

In Example 15-9 we save all tunable parameters, including those that are set to their default value.

Example 15-9 Using tunsave -af command

```
lpar05:/etc/tunables>> tunsave -af mytunable
```

Note: If the mytunable file already exists, this message will appear:
tunsave: mytunable already exists, use -F to overwrite it

Example 15-10 shows the content of the mytunable file.

Example 15-10 Content of the mytunable file

```
lpar05:/etc/tunables>> cat mytunable
(...lines omitted ...)
vmo:
memory_frames = "DEFAULT"
maxfree = "DEFAULT"
minfree = "DEFAULT"
minperm% = "DEFAULT"
minperm = "DEFAULT"
maxperm% = "50"
maxperm = "DEFAULT"
strict_maxperm = "DEFAULT"
maxpin% = "DEFAULT"
maxpin = "DEFAULT"
maxclient% = "50"
lrubucket = "DEFAULT"
defps = "DEFAULT"
```

```
nokilluid = "DEFAULT"  
numpsblks = "DEFAULT"  
npskill = "DEFAULT"  
(...lines omitted ...)
```

Example 15-10 on page 261 illustrates that all of the parameters that have default value in the system show DEFAULT, and those parameters that we changed in the system show the current value, such as maxperm% and maxclient%.

To save all tunables, including those that are set to their default value using all numerical values, but flag the default values with the comment DEFAULT VALUE with the tunsave command, the command in Example 15-11 can be used:

Example 15-11 Using the tunsave command to save tunables

```
lpar05:/etc/tunables>> tunsave -AF mytunable
```

The content of the mytunable file is shown in Example 15-12.

Example 15-12 Content of the mytunable file

```
lpar05:/etc/tunables>> cat mytunable  
(...lines omitted ...)  
vmo:  
memory_frames = "2097152" # DEFAULT VALUE  
maxfree = "128" # DEFAULT VALUE  
minfree = "120" # DEFAULT VALUE  
minperm% = "20" # DEFAULT VALUE  
minperm = "403988" # DEFAULT VALUE  
maxperm% = "50"  
maxperm = "1009970" # DEFAULT VALUE  
strict_maxperm = "0" # DEFAULT VALUE  
maxpin% = "80" # DEFAULT VALUE  
maxpin = "1677722" # DEFAULT VALUE  
maxclient% = "50"  
lrubucket = "131072" # DEFAULT VALUE  
defps = "1" # DEFAULT VALUE  
nokilluid = "0" # DEFAULT VALUE  
numpsblks = "1048576" # DEFAULT VALUE  
npskill = "8192" # DEFAULT VALUE  
npswarn = "32768" # DEFAULT VALUE  
v_pinshm = "0" # DEFAULT VALUE  
framesets = "0"  
mempools = "0"  
lgpg_size = "16777216"  
lgpg_regions = "0" # DEFAULT VALUE  
num_spec_dataseg = "0" # DEFAULT VALUE  
(...lines omitted ...)
```

As you can see in the previous example, those parameters that were changed on the system, such as `maxclient%`, `framesets`, `mempools`, and `lpgg_size`, do not show the comment `# DEFAULT VALUE`.

Use the `tunsave` command to change the description field in the `mytunable` file, as shown in Example 15-13.

Example 15-13 Changing the description field in the mytunable file

```
lpar05:/etc/tunables>> tunsave -d "new tunable file" -f mytunable
lpar05:/etc/tunables>> cat mytunable
info:
    Description = "new tunable file"
    AIX_level = "5.2.0.5"
    Kernel_type = "MP64"
    Last_validation = "2003-04-22 18:41:04 CDT (current, reboot)"
(...lines omitted ...)
```

In Example 15-14, we saved all tunables different from their default value into the `/etc/tunables/mytunable` file.

Example 15-14 Using the tunsave -f command

```
lpar05:/etc/tunables>> tunsave -f mytunable
```

The content of the `mytunable` file is shown in Example 15-15.

Example 15-15 Content of mytunable file.

```
lpar05:/etc/tunables>> cat mytunable
info:
    Description = "tunsave -f mytunable"
    AIX_level = "5.2.0.5"
    Kernel_type = "MP64"
    Last_validation = "2003-04-22 18:48:03 CDT (current, reboot)"

schedo:

vmo:
    maxperm% = "50"
    maxclient% = "50"
    framesets = "0"
    mempools = "0"
    lpgg_size = "16777216"
    spec_dataseg_int = "0"

iio:
    hd_pbuf_cnt = "1152"

no:
```

```
extendednetstats = "1"

nfso:
nfs_v2_vm_bufs = "5000"
nfs_v3_vm_bufs = "5000"
```

15.4 tundefault

The **tundefault** command is used to force all tuning parameters to be reset to their default value. The syntax is:

```
tundefault [ -p | -r ]
```

Flags

- p** Makes the changes permanent by resetting all tunable parameters to their default values and updating the `/etc/tunables/nextboot` file.
- r** Defers the reset to the default value until the next reboot. This clears stanza(s) in the `/etc/tunables/nextboot` file, and if necessary, proposes **bosboot** and warns that a reboot is needed.

This resets all AIX tunable parameters to their default value, except for parameters of type Bosboot and Reboot, and parameters of type Incremental set at values larger than their default value, unless `-r` was specified. Error messages are displayed for any parameter change impossible to make.

15.4.1 Examples for tundefault

To permanently reset all tunable parameters to their default values, use the **tundefault** command as shown in Example 15-16.

Example 15-16 Resetting all tunable parameters to their default values

```
# tundefault -p
```

All of the tuning commands are launched with `-Dp` flags. This resets all tunable parameters to their default value and updates the `/etc/tunables/nextboot` file. This command completely and permanently resets all tunable parameters to their default values.

Example 15-17 shows how to defer the resetting of all tunable parameters until the next boot using the **tundefault** command.

Example 15-17 Using tundefault command

```
# tundefault -r
```

Calls all tuning commands with **-Dr**. This clears all of the stanzas in the `/etc/tunables/nextboot` file and, if necessary, proposes `bosboot` and displays a message warning that a reboot is necessary to make the changes effective.

Use the **tundefault** command without parameters or flags, as in Example 15-18, to reset all tunable parameters to their default value, except the parameters of type `Bosboot` and `Reboot`, and parameters of type `Incremental` set values bigger than their default value.

Example 15-18 tundefault command

```
# tundefault
```

15.5 tunchange

The **tunchange** command is used to update the stanzas in the tunables file. The syntax is:

```
tunchange -f Filename ( -t Stanza ( {-o Parameter[=Value]}|-D ) | -m Filename2)
```

Flags

- f filename** Tunable file to be changed, relative to `/etc/tunables`.
- t Stanza** Command stanza name; can be `vmo`, `ioo`, `schedo`, `nfso`, or `no`.
- o Parameter[=value]** Provides the tunable and value pair to update.
- D** Change the values into the default value.
- m Filename2** Merge `Filename2` into `Filename`.

This command unconditionally changes the stanza without validating the parameter. Use this with caution.

15.5.1 Examples for tunchange

Example 15-19 shows an example of modifying the nextboot file to set a **schedo** parameter **pacefork** to 10.

Example 15-19 Modifying the stanza directly

```
$ tunchange -f nextboot -t schedo -o pacefork=10
$ cat /etc/tunables/nextboot
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
#
# bos520 src/bos/usr/sbin/perf/tune/nextboot 1.1
#
# Licensed Materials - Property of IBM
#
# (C) COPYRIGHT International Business Machines Corp. 2002
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# IBM_PROLOG_END_TAG

vmo:

schedo:
    pacefork = "10"
```

Process-related commands

The following commands work with the content of the /proc filesystem:

procdx	Prints the current working directory of processes.
procfles	Reports information about all file descriptors opened by processes.
proclags	Prints the /proc tracing flags, the pending and held signals, and other /proc status information for each thread in the specified processes.
proccred	Prints the credentials (effective, real, saved user IDs, and group IDs) of processes.
procmmap	Prints the address space map of processes.
procldd	Lists the dynamic libraries loaded by processes, including shared objects explicitly attached using dlopen().
procsig	Lists the signal actions defined by processes.
procstack	Prints the hexadecimal addresses and symbolic names for each of the stack frames of the current thread in processes.
proctop	Stops processes on the PR_REQUESTED event.
procrun	Starts a process that has stopped on the PR_REQUESTED event.
proctwait	Waits for all of the specified processes to terminate.
proctree	Prints the process tree containing the specified process IDs or users.

16.1 procwdx

The **procwdx** command prints the current working directory of processes. The syntax of the **procwdx** command is:

```
procwdx [ -F ] [ ProcessID ] ...
```

Flags

-F Forces **procwdx** to take control of the target process even if another process has control.

Parameters

ProcessID Specifies the process ID.

Examples

Example 16-1 shows the current working directory of the processes.

Example 16-1 Displaying the current working directory of process 454698

```
lpar05:/>> procwdx 454698
454698: /usr/WebSphere/AppServer/
```

16.2 procfiles

The **procfiles** command reports information about all file descriptors opened by process. The syntax of the **procfiles** command is:

```
procfiles [ -F ] [ -n ] [ ProcessID ] ...
```

Flags

-F Forces **procfiles** to take control of the target process even if another process has control.

-n Prints the names of the files referred to by file descriptors.

Parameters

ProcessID Specifies the process ID.

Examples

Example 16-2 on page 269 shows information about the file descriptors opened by process.

Example 16-2 Getting file descriptors information for process 454698

```
lpar05:/>> procfiles 454698
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere
/AppServ
Current rlimit: 32000 file descriptors
0: S_IFCHR mode:00 dev:10,4 ino:12631 uid:0 gid:0 rdev:21,1
O_RDWR
1: S_IFREG mode:0644 dev:10,5 ino:620777 uid:0 gid:0 rdev:0,0
O_WRONLY size:0
2: S_IFREG mode:0644 dev:10,5 ino:620778 uid:0 gid:0 rdev:0,0
O_RDWR size:0
3: S_IFIFO mode:00 dev:65535,65535 ino:623087800 uid:0 gid:0 rdev:0,0
O_RDONLY
4: S_IFREG mode:0555 dev:10,5 ino:540329 uid:0 gid:0 rdev:9,24962
5: S_IFREG mode:0555 dev:10,5 ino:540499 uid:0 gid:0 rdev:9,22707
6: S_IFREG mode:0555 dev:10,5 ino:540493 uid:0 gid:0 rdev:9,21313
7: S_IFREG mode:0555 dev:10,5 ino:540397 uid:0 gid:0 rdev:9,16584
8: S_IFREG mode:0555 dev:10,5 ino:539399 uid:0 gid:0 rdev:8,27459
9: S_IFREG mode:0555 dev:10,5 ino:539400 uid:0 gid:0 rdev:8,27504
10: S_IFREG mode:0555 dev:10,5 ino:539401 uid:0 gid:0 rdev:8,28736
... ( lines omitted)...
```

Example 16-3 shows all information about the file descriptors opened by the process and prints the names of the files referred to by file descriptors.

Example 16-3 Getting file descriptors information and names for process 454698

```
lpar05:/>> procfiles -n 454698
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere
/AppServ
Current rlimit: 32000 file descriptors
0: S_IFCHR mode:00 dev:10,4 ino:12631 uid:0 gid:0 rdev:21,1
O_RDWR name:/dev/pts/1
1: S_IFREG mode:0644 dev:10,5 ino:620777 uid:0 gid:0 rdev:0,0
O_WRONLY size:0
name:/usr/WebSphere/AppServer/logs/server1/native_stdout.log
2: S_IFREG mode:0644 dev:10,5 ino:620778 uid:0 gid:0 rdev:0,0
O_RDWR size:0
name:/usr/WebSphere/AppServer/logs/server1/native_stderr.log
3: S_IFIFO mode:00 dev:65535,65535 ino:623087800 uid:0 gid:0 rdev:0,0
O_RDONLY name:Cannot be retrieved
4: S_IFREG mode:0555 dev:10,5 ino:540329 uid:0 gid:0 rdev:9,24962
```

```
5: S_IFREG mode:0555 dev:10,5 ino:540499 uid:0 gid:0 rdev:9,22707
6: S_IFREG mode:0555 dev:10,5 ino:540493 uid:0 gid:0 rdev:9,21313
7: S_IFREG mode:0555 dev:10,5 ino:540397 uid:0 gid:0 rdev:9,16584
8: S_IFREG mode:0555 dev:10,5 ino:539399 uid:0 gid:0 rdev:8,27459
9: S_IFREG mode:0555 dev:10,5 ino:539400 uid:0 gid:0 rdev:8,27504
... ( lines omitted)...
```

16.3 procflags

The **procflags** command prints the /proc tracing flags, with the pending and held signals, and other /proc status information for each thread in the specified process. The syntax of the **procflags** command is:

```
procflags [ -r ] [ ProcessID ] ...
```

Flags

-r Displays the current machine registers state if a process is stopped in an event of interest.

ProcessID Specifies the process ID.

Examples

Example 16-4 shows the state of a process.

Example 16-4 Using procflags to get state information of the 454698 process

```
lpar05:/>> procflags 454698
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere
/AppServ
data model = _ILP32 flags = PR_FORK
/876673: flags = PR_ASLEEP | PR_NOREGS
... ( lines omitted)...
```

16.4 proccred

The **proccred** command prints the credentials (effective, real, saved user IDs, and group IDs) of processes. The syntax of the **proccred** command is:

```
proccred [ ProcessID ] ...
```

Parameters

ProcessID Specifies the process ID

Examples

Example 16-5 shows the credentials of a process.

Example 16-5 Displaying credentials for process 454698

```
lpar05:/>> proccred 454698
454698: e/r/suid=0 e/r/sgid=0
```

16.5 procmap

The **procmap** command prints the address space map of processes. It displays the starting address and size of each of the mapped segments in the process. It gets all necessary information from the `/proc/pid/map` files. The syntax of the **procmap** command is:

```
procmap [ -F ] [ ProcessID ] ...
```

Flags

-F Forces **procmap** to take control of the target process even if another process has control.

Parameters

ProcessID Specifies the process ID

Examples

Example 16-6 shows output from the **procmap** command.

Example 16-6 Displaying the address space of process 454698

```
lpar05:/>> procmap 454698
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere
/AppServ
10000000          25K read/exec          java
30000a33         2K read/write         java
d3b9d000         6K read/exec
/usr/WebSphere/AppServer/java/jre/bin/liborb.a
42cc5e28         0K read/write
/usr/WebSphere/AppServer/java/jre/bin/liborb.a
```

```

d3b8c000          46K  read/exec
/usr/WebSphere/AppServer/java/jre/bin/libnet.a

426e2cd8          1K  read/write
/usr/WebSphere/AppServer/java/jre/bin/libnet.a

d3b98000          19K  read/exec
/usr/WebSphere/AppServer/bin/libWs50ProcessManagement.so

424601bc          0K  read/write
/usr/WebSphere/AppServer/bin/libWs50ProcessManagement.so
... ( lines omitted)...

```

16.6 procldd

The **procldd** command lists the dynamic libraries loaded by processes, including shared objects explicitly attached using `dlopen()`. All necessary information is gathered from the `/proc/pid/map` files. The syntax of the **procldd** command is:

```
procldd [ -F ] [ ProcessID ] ...
```

Flags

-F Forces **procldd** to take control of the target process even if another process has control.

Parameters

ProcessID Specifies the process ID.

Examples

Example 16-7 shows the list of dynamic libraries loaded by a process.

Example 16-7 Output from procldd command using process 454698

```

lpar05:/>> procldd 454698
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere
/AppServ
/usr/WebSphere/AppServer/java/jre/bin/liborb.a
/usr/WebSphere/AppServer/java/jre/bin/libnet.a
/usr/WebSphere/AppServer/bin/libWs50ProcessManagement.so
/usr/WebSphere/AppServer/java/jre/bin/libjtc.a
/usr/WebSphere/AppServer/java/jre/bin/libzip.a
/usr/WebSphere/AppServer/java/jre/bin/libhpi.a
/usr/WebSphere/AppServer/java/jre/bin/libxhpi.a
/usr/WebSphere/AppServer/java/jre/bin/libjava.a
/usr/WebSphere/AppServer/java/jre/bin/classic/libjvm.a

```

```
/usr/lib/libbsd.a  
/usr/lib/libpthread.a  
/usr/lib/libC.a
```

16.7 procsig

The **procsig** command lists the signal actions defined by processes. The syntax of the **procsig** command is:

```
procsig [ ProcessID ] ...
```

Parameters

ProcessID Specifies the process ID.

Examples

Example 16-8 shows all signal actions defined for a process.

Example 16-8 Output from procsig command for process 454698

```
lpar05:/>> procsig 454698  
454698 :/usr/WebSphere/AppServer/java/bin/java -Xbootclasspath/p:/usr/WebSphere  
/AppServ  
HUP      caught  RESTART | SIGINFO  
INT      caught  RESTART | SIGINFO  
QUIT     caught  RESTART | SIGINFO  
ILL      caught  RESTART | SIGINFO  
TRAP     caught  RESETHAND  
ABRT     caught  RESTART | SIGINFO  
EMT      caught  RESTART | SIGINFO  
FPE      caught  RESTART | SIGINFO  
KILL     default  RESTART  
BUS      caught  RESTART | SIGINFO  
SEGV     caught  RESTART | SIGINFO  
SYS      caught  RESTART | SIGINFO  
PIPE     ignored  
ALRM     default  
TERM     caught  RESTART | SIGINFO  
URG      default  
STOP     default  
TSTP     default  
CONT     default  
CHLD     default  
TTIN     ignored  RESETHAND  
TTOU     ignored  RESETHAND  
IO       default  
XCPU     caught  RESTART | SIGINFO
```

XFSZ	caught	RESTART SIGINFO
MSG	default	
WINCH	default	
PWR	default	
USR1	default	
USR2	caught	RESTART SIGINFO
PROF	default	
DANGER	default	
VTALRM	default	
MIGRATE	default	
PRE	default	RESTART
VIRT	default	
ALRM1	default	
WAITING	default	RESTART
CPUFAIL	default	
KAP	default	
RETRACT	default	
SOUND	default	
SAK	default	

16.8 procstack

The **procstack** command prints the hexadecimal addresses and symbolic names for each of the stack frames of the current thread in processes. The syntax of the **procstack** command is:

```
procstack [ -F ] [ ProcessID ] ...
```

Flag

-F Forces **procstack** to take control of the target process even if another process already has control.

Parameter

ProcessID Specifies the process ID.

Example

Example 16-9 shows the current stack of a process.

Example 16-9 Output from procstack command for process 643298

```
lpar05:/>> procstack 643298
643298 : /usr/java131/jre/bin/java -Xms1024m -Xmx1024m VBDMemBlot 2000000
d41eb494 HashedAndMovedSize ( ?, ? ) + 78
d41e76d8 reverseHandlesAndUpdateForwardRefs ( ?, ?, ? ) + 51c
d41eb68c compactHeap ( ?, ?, ? ) + 17c
```



```

d41ef920 gc0_locked  (? , ? , ? , ?) + 173c
d4218468 gc_locked  (? , ? , ? , ?) + 40
d41f3920 gc0      (? , ? , ? , ?) + 2fc
d41f41cc manageAllocFailure (? , ? , ?) + 3a0
d41bd75c lockedHeapAlloc (? , ? , ? , ? , ?) + 4ac
d41bf090 realObjAlloc (? , ? , ? , ?) + 15c
d41c0214 targetedAllocMiddlewareArray (? , ? , ? , ?) + 9c
d42da8fc _jit_anewarray_quick (? , ? , ? , ? , ?) + 50
d42daa60 _jit_anewarray (? , ? , ? , ? , ?) + 94
75195618 ???????? (30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0,
0, 20, 745125e0, 0, 30319370, 0, 29, 0, 0, 20, 745125e0, 0, 30319370, 0, 29, 0)

```

16.9 procstop

The **procstop** command stops processes on the PR_REQUESTED event. The syntax of the **procstop** command is:

```
procstop [ ProcessID ] ...
```

Parameter

ProcessID Specifies the process ID.

Example

Example 16-10 shows the stop process on the PR_REQUESTED event.

Example 16-10 Stop process 622654 using the procstop command

```
lpar05:/>> procstop 622654
```

16.10 procrun

The **procrun** command starts processes stopped by the previous command, **procstop**. The syntax of the **procrun** command is:

```
procrun [ ProcessID ] ...
```

Parameter

ProcessID Specifies the process ID.

Example

Example 16-11 shows how to restart a process that was stopped on the PR_REQUESTED event.

Example 16-11 Restart process 622654 using the procrun command

```
lpar05:/>> procrun 622654
```

16.11 procwait

The **procwait** command waits for all of the specified processes to terminate. The syntax of the **procwait** command is:

```
procwait [ -v ] [ ProcessID ] ...
```

Flag

-v Specifies verbose output. Reports terminations to standard output.

Parameter

ProcessID Specifies the process ID.

Example

Example 16-12 wait for a exit process and display the status.

Example 16-12 Using procwait command for process 569540

```
lpar05:/>> procwait -v 569540  
569540 : terminated, exit status 0
```

16.12 proctree

The **proctree** command prints the process tree containing the specified process IDs or users. The child processes are indented from their respective parent processes. An argument of all digits is taken to be a process ID; otherwise it is assumed to be a user login name. The default action is to report on all processes, except children of process 0.

The syntax of the **proctree** command is:

```
proctree [ -a ] [ { ProcessID | User } ]
```

Flags

-a Include children of process 0 in the display. The default is to exclude them.

Parameters

ProcessID Specifies the process ID.
User Specifies the User.

Examples

Example 16-13 displays the ancestors and all of the children of a process.

Example 16-13 Displaying process 159936 using proctree command

```
lpar05:/>> proctree 159936
159936 /usr/sbin/srcmstr
  106556 /usr/sbin/aixmibd
  123060 /usr/sbin/muxatmd
  131174 /usr/sbin/rpc.lockd
  155856 /usr/sbin/snmpmibd
  172154 /usr/sbin/inetd
    467096 telnetd -a
      430148 -ksh
    471164 telnetd -a
      434186 -ksh
    626732 proctree 159936
    594124 telnetd -a
      532602 -ksh
    569560 telnetd -a
      565396 -ksh
  209092 /usr/sbin/hostmibd
  213164 /usr/sbin/snmpd
  217146 /usr/sbin/portmap
  221312 /usr/sbin/syslogd
  262272 /usr/sbin/biod 6
  266370 /usr/sbin/nfsd 3891
  274566 /usr/sbin/rpc.mountd
  278664 /usr/sbin/rpc.statd
  286866 /usr/sbin/qdaemon
  290962 /usr/sbin/writesrv
  307360 /usr/sbin/rsct/bin/rmcd -r
  344234 /usr/sbin/rsct/bin/ctcasd
  360640 /usr/sbin/rsct/bin/IBM.ERrmd
  377024 /usr/sbin/rsct/bin/IBM.AuditRMd
  389330 /usr/sbin/rsct/bin/IBM.FSrmcd
  401610 /usr/sbin/rsct/bin/IBM.ServiceRMd
  405704 /usr/sbin/rsct/bin/IBM.CSMAgentRMd
```

Example 16-14 displays the ancestors and children of a process, including children of process 0.

Example 16-14 Displaying process 159936 using proctree -a command

```
lpar05:/>> proctree -a 159936
1 /etc/init
  159936 /usr/sbin/srcmstr
    106556 /usr/sbin/aixmibd
    123060 /usr/sbin/muxatmd
    131174 /usr/sbin/rpc.lockd
    155856 /usr/sbin/snmpibd
    172154 /usr/sbin/inetd
      467096 telnetd -a
        430148 -ksh
      471164 telnetd -a
        434186 -ksh
      626734 proctree -a 159936
        594124 telnetd -a
          532602 -ksh
        569560 telnetd -a
          565396 -ksh
    209092 /usr/sbin/hostmibd
    213164 /usr/sbin/snmpd
    217146 /usr/sbin/portmap
    221312 /usr/sbin/syslogd
    262272 /usr/sbin/biod 6
    266370 /usr/sbin/nfsd 3891
    274566 /usr/sbin/rpc.mountd
    278664 /usr/sbin/rpc.statd
    286866 /usr/sbin/qdaemon
    290962 /usr/sbin/writesrv
    307360 /usr/sbin/rsct/bin/rmcd -r
    344234 /usr/sbin/rsct/bin/ctcasd
    360640 /usr/sbin/rsct/bin/IBM.ERrmd
    377024 /usr/sbin/rsct/bin/IBM.AuditRMd
    389330 /usr/sbin/rsct/bin/IBM.FSrmD
    401610 /usr/sbin/rsct/bin/IBM.ServiceRMd
    405704 /usr/sbin/rsct/bin/IBM.CSMagentRMd
```

CPU-related performance tools

This part describes tools for monitoring the performance-relevant data and statistics for CPU resource. It also contains information about tools that can be used to tune CPU usage.

Other commands that provide statistics on CPU usage that are not listed in this chapter may appear in other chapters of this book such as Chapter 2, “Multi-resource monitoring and tuning tools” on page 67 and Chapter 7, “Tracing performance problems” on page 675.

This part contains detailed information about the following CPU monitoring and tuning tools:

- ▶ CPU monitoring tools:
 - The **alstat** command, described in 17.2, “alstat” on page 283, is used to monitor Alignment exception statistics.
 - The **emstat** command, described in 17.3, “emstat” on page 285, is used to monitor Emulation statistics.
 - The **gprof** command, described in 19.2, “gprof” on page 300, is used to profile applications, showing details of time spent in routines.
 - The **pprof** command, described in 19.3, “pprof” on page 309, is used to monitor processes and threads.
 - The **prof** command, described in 19.4, “prof” on page 320, is used to profile applications, showing details of time spent in routines.
 - The **time** command, described in 21.1, “time” on page 356, is used to report the real time, user time, and system time taken to execute a command.
 - The **timex** command, described in 21.2, “timex” on page 357, is used report the real time, user time, and system time taken to execute a command. It also reports on I/O statistics, context switches, and run queue status, among other statistics.
 - The **tprof** command, described in 19.5, “tprof” on page 324, is used to profile the system or an application.
- ▶ CPU tuning tools:
 - The **bindintcpu** and **bindprocessor** commands, described in Chapter 18, “The bindintcpu and bindprocessor commands” on page 289, is used bind an interrupt or a process to a specific CPU.
 - The **nice** and **renice** commands, described in Chapter 20, “The nice and renice commands” on page 349, are used to adjust the initial priority of a command.

The `alstat` and `emstat` commands

The `alstat` command displays alignment exception statistics. The `emstat` command displays emulation exception statistics.

The `emstat` and `alstat` commands reside in `/usr/bin` and are part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media. The `alstat` command is a symbolic link to `emstat`.

17.1 Alignment and emulation exception

Alignment exceptions may occur when the processor cannot perform a memory access due to an unsupported memory alignment offset (such as a floating point double load from an address that is not a multiple of eight). However, some types of unaligned memory references may be corrected by some processors and do not generate an alignment exception. See 17.2.3, “Detecting and resolving alignment problems” on page 285 for more information.

Many platforms simply abort a program with alignment problems. AIX catches these exceptions and “fixes” them so legacy applications are still able to be run. You may pay a performance price for these operating system “fixes” and should correct them permanently so they do not recur.

Emulation exceptions can occur when some legacy applications or libraries that contain instructions that have been deleted and are being executed on newer processors.

These instructions may cause illegal instruction program exceptions. The operating system kernel has emulation routines that catch these exceptions and emulate the older instruction(s) to maintain program functionality, potentially at the expense of performance. The emulation exception count since the last time the machine was rebooted and the count in the current interval are displayed.

Emulation can cause a severe degradation in performance, and an emulated instruction can cause hundreds of instructions to be generated to emulate it. Refer to 17.3.3, “Detecting and resolving emulation problems” on page 288, which shows what can be done to resolve emulation problems.

The user can, optionally, display alignment exception statistics or individual processor emulation statistics. The default output displays statistics every second. The sampling interval and number of iterations can also be specified.

If a system is underperforming after applications are transferred or ported to a new system, then emulation exceptions and memory alignment should be checked.

Tip: When diagnosing performance problems, you should always check for emulated instructions and alignment exceptions as they can cause the performance of the system to degrade.

17.2 alstat

This is the syntax of the **alstat** command:

```
alstat [[-e] | [-v]] [interval] [count]
```

Flags

-e Displays emulation stats.
-v Specifies verbose (per-CPU) stats.

Parameters

interval Specifies the update period (in seconds).
count Specifies the number of iterations.

17.2.1 Information about measurement and sampling

The **alstat** command displays alignment exception statistics. The default output displays statistics every second. The sampling interval and number of iterations can also be specified by the user.

In terms of performance, alignment exceptions are costly. Alignment exceptions could indicate that an application is not behaving well. Applications causing an increase in the **alstat** count are less disciplined in memory model, or perhaps the data structures do not map well between architectures when the applications are ported between different architectures. The kernel and the kernel extensions may also be ported and exhibit alignment problems. **alstat** looks for structures and memory allocations that do not fall on eight-bytes boundaries.

After identifying a high alignment exception rate, **tprof** should be used to isolate where the alignment exception is occurring.

17.2.2 Examples for alstat

Example 17-1 shows a system with alignment exceptions as displayed by the **alstat** command without options. Each interval will be one second long.

Example 17-1 An example of the output of alstat

```
# alstat
  Alignment  Alignment
  SinceBoot  Delta
    8845591    0
    8845591    0
    8845591    0
```

```
8845591      0
8845591      0
```

The report shows these columns:

Alignment SinceBoot The total number of alignment exceptions since start-up plus the number for the last interval.

Alignment Delta The number of alignment exceptions for the last interval.

To display emulation and alignment exception statistics every two seconds for a total of five times, use the command as in Example 17-2.

Example 17-2 Displaying emulation and alignment statistics per time and interval

```
# alstat -e 2 5
```

Alignment SinceBoot	Alignment Delta	Emulation SinceBoot	Emulation Delta
70091846	0	21260604	0
72193861	2102015	23423104	2162500
74292759	2098898	25609796	2186692
76392234	2099475	27772897	2163101
78490284	2098050	29958509	2185612

Because the **alstat** and **emstat** commands make use of the same binaries, you the output of Example 17-2 will be the same as **emstat -a**; refer to 17.3.2, “Examples for emstat” on page 286.

The report has the following columns:

Emulation SinceBoot The sum of the number of emulated instructions since start-up plus the number in the previous interval.

Emulation Delta The number of emulated instructions in the previous interval.

Alignment SinceBoot The sum of the number of alignment exceptions since start-up plus the number in the previous interval.

Alignment Delta The number of alignment exceptions in the previous interval.

To display alignment statistics every five seconds for each processor, use the command shown in Example 17-3.

Example 17-3 Displaying emulation for each processor

```
# alstat -v 5
```

This produces the following output:

Alignment SinceBoot	Alignment Delta	Alignment Delta00	Alignment Delta01
88406295	0	0	0
93697825	5291530	0	5291530
98930330	5232505	5232505	0
102595591	3665261	232697	3432564
102595591	0	0	0

The report has the following columns:

Alignment SinceBoot	The sum of the number of alignment exceptions since start-up plus number in the last interval.
Alignment Delta	The number of alignment exceptions in the previous interval for all CPUs.
Alignment Delta00	The number of current alignment exceptions in the previous interval for CPU0.
Alignment Delta01	The number of current alignment exceptions in the previous interval for CPU1.

17.2.3 Detecting and resolving alignment problems

Alignment is usually attributed to legacy applications or libraries, kernels, or kernel extensions that have been ported to different platforms. **alstat** indicates that an alignment problem exists. Once you have used the **alstat** command to identify a high alignment exception rate, the best course of action would be to recompile your application.

17.3 emstat

The syntax of the **emstat** command is:

```
emstat [[-a] | [-v]] [interval] [count]
```

Flags

-a	Displays alignment stats.
-v	Specifies verbose (per-CPU) stats.

These are the parameters:

interval	Specifies the update period (in seconds).
count	Specifies the number of iterations.

17.3.1 Information about measurement and sampling

Instructions that have been removed from earlier architectures are caught by the operating system and those instructions are emulated. The emulated exceptions count is reported by the **emstat** command. The default output displays statistics every second. The sampling interval and number of iterations can also be specified by the user.

The first line of the **emstat** output is the total number of emulations detected since the system was rebooted. The counters are stored in per-processor structures.

An average rate of more than 1000 emulated instructions per second may cause a performance degradation. Values of 100,000 or more per second will certainly cause performance problems.

17.3.2 Examples for emstat

Example 17-4 shows a system with emulation as displayed by the **emstat** command with no options. This will display emulations once per second.

Example 17-4 An example of the output of emstat

```
# emstat
emstat total count      emstat interval count
3236560
3236580                  20
3236618                  38
3236656                  38
3236676                  20
3236714                  38
3236752                  38
3236772                  20
3236810                  38
3236848                  38
emstat total count      emstat interval count
3236868                  20
3236906                  38
3236944                  38
3236964                  20
3237002                  38
3237040                  38
...
```

The report has the following columns:

emstat total count	The total number of emulated instructions since start-up plus that of the last interval.
--------------------	--

`emstat interval count` The first line of the report is the total number of emulated instructions since start-up. Subsequent lines show the number of emulations in the last interval.

To display emulation and alignment exception statistics every two seconds, a total of five times, use the command shown in Example 17-5.

Example 17-5 Displaying emulation and alignment statistics per time and interval

```
# emstat -a 2 5
```

Alignment SinceBoot	Alignment Delta	Emulation SinceBoot	Emulation Delta
21260604	0	70091846	0
23423104	2162500	72193861	2102015
25609796	2186692	74292759	2098898
27772897	2163101	76392234	2099475
29958509	2185612	78490284	2098050

The report has the following columns:

Alignment SinceBoot	The sum of the number of alignment exceptions since start-up plus that of the last interval
Alignment Delta	The number of alignment exceptions in the last interval
Emulation SinceBoot	The sum of the number of emulated instructions since start-up plus that of the last interval
Emulation Delta	The number of emulated instructions in the last interval

To display emulation statistics every five seconds for each processor, use the command in Example 17-6. Because `alstat` and `emstat` use the same binaries, using the `alstat -e` command will produce the same output.

Example 17-6 Displaying emulation for each processor

```
# emstat -v 5
```

Emulation SinceBoot	Emulation Delta	Emulation Delta00	Emulation Delta01
88406295	0	0	0
93697825	5291530	0	5291530
98930330	5232505	5232505	0
102595591	3665261	232697	3432564
102595591	0	0	0

The report has the following columns:

Emulation SinceBoot	The sum of the number of emulated instructions since start-up plus that of the last interval
Emulation Delta	The number of emulated instructions in the previous interval for all CPUZ
Emulation Delta00	The number of emulated instructions in the previous interval for cpu0.
Emulation Delta01	The number of emulated instructions in the previous interval for cpu1

17.3.3 Detecting and resolving emulation problems

Emulation is usually attributed to legacy applications or libraries that contain instructions that have been deleted from newer processor architectures.

Emulation occurs when programs have been compiled for specific architecture. For example, a program compiled for the 601 processor will produce emulation problems on a 604-based processor because the 604 chip has to emulate instructions for the 601 processor to maintain program functionality. To maintain functionality across the processors, a program must be compiled for common architecture with `-qarch=com` as flags for the `cc` compiler. Alternatively, the program may be compiled for a specific chip set. If you are a software vendor, then you can compile with a common architecture to avoid having multiple ports of the same code.

The **bindintcpu** and **bindprocessor** commands

The **bindintcpu** and **bindprocessor** commands are used in Symmetrical Multiprocessor (SMP) systems to dedicate a specific processor. The **bindintcpu** command is used to direct an interrupt from a specific hardware device, at a specific interrupt level, to a specific CPU number or numbers. The **bindprocessor** command is used to bind or unbind the threads of a process to a processor on an SMP system.

The **bindintcpu** command is only applicable to certain hardware types. Once an interrupt level has been directed to a CPU, all interrupts on that level will be directed to that CPU until directed otherwise by the **bindintcpu** command.

Only a user with root authority can bind an interrupt to a processor or a thread of a process of which it is not the owner to a processor.

The **bindintcpu** command resides in `/usr/sbin` and is part of the `devices.chrp.base.rte` fileset, which is installable from the AIX base installation media. The **bindprocessor** command resides in `/usr/sbin` and is part of the `bos.mp` fileset, which is installed by default on SMP systems when installing AIX.

18.1 bindintcpu

The syntax of the **bindintcpu** command is:

```
bindintcpu <level> <cpu> [<cpu>...]
```

Parameters

level The bus interrupt level.

cpu The specific CPU number. You may be able to bind an interrupt to more than one CPU.

18.1.1 Examples for bindintcpu

The **bindintcpu** command can be useful for redirecting an interrupt to a specific processor. If the threads of a process are bound to a specific CPU using the **bindprocessor** command, this process could be continually disrupted by an interrupt from a device. Refer to 18.2, “bindprocessor” on page 292 for more details on the **bindprocessor** command.

This continual interruption can become a performance issue if the CPU is frequently interrupted. To overcome this, an interrupt that is continually interrupting a CPU can be redirected to a specific CPU or CPUs other than the CPU where the threads are bound. Assuming that the interrupt is from the Ethernet adapter ent1, the following procedure can be performed.

Note: Not all hardware supports one interrupt level binding to multiple CPUs, and an error may therefore result when using **bindintcpu** on some systems. It is recommended to specify only one CPU per interrupt level. If an interrupt level is redirected to CPU0, then this interrupt level cannot be redirected to another CPU by the **bindintcpu** command until the system has been rebooted.

To determine the interrupt level for a specific device, the **lsattr** command can be used as in Example 18-1. Here we see that the interrupt level is 548.

Example 18-1 How to determine the interrupt level of an adapter

```
# lsattr -El ent1
busmem            0xe8030000        Bus memory address        False
rom_mem           0xe8000000        ROM memory address        False
busintr          548             Bus interrupt level        False
intr_priority     3                 Interrupt priority         False
txdesc_queue_sz   512               TX Descriptor Queue Size   True
rxdesc_queue_sz   512               RX Descriptor Queue Size   True
tx_queue_sz       8192              Software TX Queue Size     True
```


rxbuf_pool_sz	1024	Receive Buffer Pool Size	True
media_speed	Auto_Negotiation	Media Speed	True
use_alt_addr	no	Enable Alternate Ethernet Address	True
alt_addr	0x000000000000	Alternate Ethernet Address	True
tx_preload	1520	TX Preload Value	True
ipsec_offload	no	IPsec Offload	True
rx_checksum	no	Enable RX Checksum Offload	True
large_send	no	Enable TCP Large Send Offload	True
slih_hog	10	Interrupt Events per Interrupt	True
rx_hog	1000	RX Descriptors per RX Interrupt	True
poll_link	no	Enable Link Polling	True
poll_link_timer	500	Time interval for Link Polling	True

To determine which CPUs are available on the system, the **bindprocessor** command can be used as in Example 18-2.

Example 18-2 The bindprocessor command shows available CPUs

```
# bindprocessor -q
The available processors are: 0 1 2 3
```

In order to redirect the interrupt level 548 to CPU1 on the system, use the **bindintcpu** command as follows:

```
bindintcpu 548 1
```

All interrupts from bus interrupt level 548 will be handled by the processor CPU1. The other CPUs of the system will no longer be required to service interrupts from this interrupt level.

In Example 18-3, the system has four CPUs. These CPUs are CPU0, CPU1, CPU2, and CPU3. If a non-existent CPU number is entered, an error message is displayed.

Example 18-3 Incorrect CPU number selected in the bindintcpu command

```
# bindintcpu 548 3 4
Invalid CPU number 4
Usage: bindintcpu <level> <cpu> [<cpu>...]
Assign interrupt at <level> to be delivered only to the indicated cpu(s).
```

The **vmstat** command can be used as shown in Example 18-4 to obtain interrupt statistics.

Example 18-4 Use the vmstat command to determine the interrupt statistics

```
# vmstat -i
priority level    type    count module(handler)
    0         2 hardware 4189 i_hwassist_int(267880.16)
    3        70 hardware  460 /usr/lib/drivers/pci/s_scsiddpin(205cdc8.16)
```

3	74	hardware	225	/usr/lib/drivers/pci/s_scsiddpin(205cdc8.16)
3	547	hardware	372445	/usr/lib/drivers/pci/scentdd(210f144.16)
3	548	hardware	10062	/usr/lib/drivers/pci/scentdd(210f144.16)
3	565	hardware	7532668	/usr/lib/drivers/pci/s_scsiddpin(205cdc8.16)

The column heading `level` shows the interrupt level, and the column heading `count` gives the number of interrupts since system startup. For more information, refer to 13.1, “`vmstat`” on page 212.

18.2 bindprocessor

The syntax of the **bindprocessor** command is:

```
bindprocessor Process [ ProcessorNum ] | -q | -u Process
```

Flags

- q** Displays the processors that are available.
- u** Unbinds the threads of the specified process.

Parameters

- Process** This is the process identification number (PID) for the process to be bound to a processor.
- [ProcessorNum]** This is the processor number as specified from the output of the **bindprocessor -q** command. If the parameter `ProcessorNum` is omitted, then the thread of a process will be bound to a randomly selected processor.

18.2.1 Information about measurement and sampling

The **bindprocessor** command uses the `bindprocessor` kernel service to bind or unbind a kernel thread to a processor. The `bindprocessor` kernel service binds a single thread or all threads of a process to a processor. Bound threads are forced to run on that processor. Processes are not bound to processors; the kernel threads of the process are bound. Kernel threads that are bound to the chosen processor, remain bound until unbound by the **bindprocessor** command or until they terminate. New threads that are created using the `thread_create` kernel service become bound to the same processor as their creator.

The **bindprocessor** command uses logical, not physical processor, numbers.

18.2.2 Examples for bindprocessor

To display the available processors, the command in Example 18-5 can be used.

Example 18-5 Displaying available processors with the bindprocessor command

```
# bindprocessor -q
The available processors are: 0 1 2 3
```

In Example 18-6, there are four CPU-intensive processes consuming all of the CPU time on all four of the available processors. This scenario may result in a poor response time for other applications on the system. The example shows a **topas** output where there is a high CPU usage on all available CPUs. Refer to 11.1, “topas” on page 180 for more information. The process list at the bottom of the **topas** output shows the processes that are consuming the CPU time. The process identification numbers (PID) for the processes obtained from the **topas** command can be used with the **bindprocessor** command.

Example 18-6 Topas showing top processes consuming all CPU resources

```
Topas output shows high CPU usage
Topas Monitor for host:  lpar05
Tue Apr 8 09:42:19 2003 Interval: 2
```

CPU	User%	Kern%	Wait%	Idle%	EVENTS/QUEUES	FILE/TTY
cpu2	100.0	0.0	0.0	0.0	Cswitch 215	Readch 0
cpu0	100.0	0.0	0.0	0.0	Syscall 275	Writech 72
cpu3	100.0	0.0	0.0	0.0	Reads 0	Rawin 0
cpu1	100.0	0.0	0.0	0.0	Writes 0	Ttyout 0
					Forks 0	Igets 0
					Execs 0	Namei 0
					Runqueue 13.5	Dirblk 0
					Waitqueue 0.0	

Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out	PAGING	MEMORY
en0	0.0	0.5	0.5	0.0	0.0	Faults 0	Real,MB 8191
lo0	0.0	0.0	0.0	0.0	0.0	Steals 0	% Comp 9.7

Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ	PgspIn	% Noncomp
hdisk0	0.0	0.0	0.0	0.0	0.0	PgspOut 0	Client 0.5
hdisk1	0.0	0.0	0.0	0.0	0.0	PageIn 0	
						PageOut 0	PAGING SPACE

WLM-Class (Active)	CPU%	Mem%	Disk-I/O%	Sios	Size,MB
Default	97	0	0	0	2048
java	0	2	0	0	9.1
http	0	0	0	0	90.8
System	0	29	0	0	
Shared	0	0	0	0	
Unmanaged	0	3	0	0	
Unclassified	0	1	0	0	

The top 8 processors are displayed :

```
Topas Monitor for host:  lpar05 Interval: 2 Tue Apr 8 09:49:01 2003
```

USER	PID	PPID	PRI	NI	DATA RES	TEXT RES	PAGE SPACE	TIME	CPU%	I/O	OTH	COMMAND
res1	24680	1	153	24	12	12	12	7:44	100.0	0	0	cwhet_c
res1	17970	1	153	24	12	12	12	7:45	100.0	0	0	cwhet_d
res1	31492	1	153	24	12	12	12	7:46	100.0	0	0	cwhet_a
res1	34244	1	153	24	12	12	12	7:46	100.0	0	0	cwhet_b
root	5418	0	36	41	4	0	4	0:03	0.0	0	0	netm
root	5676	0	37	41	14	0	17	7:52	0.0	0	0	gil
root	5934	0	16	41	2	0	4	0:10	0.0	0	0	wlmsched
root	6502	0	50	41	2	0	4	0:00	0.0	0	0	jfsz

The **bindprocessor** commands in Example 18-7 are used to bind the threads of the top processes in Example 18-6 on page 293 to CPU1.

Example 18-7 The bindprocessor command used to bind processes to a CPU

```
# bindprocessor 24690 1
# bindprocessor 27970 1
# bindprocessor 31492 1
# bindprocessor 34244 1
```

Example 18-8 shows statistics obtained from the **topas** command output for CPU and processes after the **bindprocessor** command was used to bind the threads of the top four processes seen in Example 18-6 on page 293 to CPU1. Ultimately the length of time that the four processes will run for on CPU1 will be longer than if they were left to run on all four processors.

Example 18-8 bindprocessor command used to bind processes to a processor

Topas Monitor for host: lpar05						EVENTS/QUEUES		FILE/TTY				
Tue Apr 8 09:54:47 2003						Interval: 2		Cswitch	322	Readch	2	
						Syscall		289	Writech	126		
CPU	User%	Kern%	Wait%	Idle%			Reads	1	Rawin	0		
cpu1	100.0	0.0	0.0	0.0			Writes	1	Ttyout	0		
cpu3	0.0	0.0	0.0	100.0			Forks	0	Igets	0		
cpu0	0.0	0.0	0.0	100.0			Execs	0	Namei	0		
cpu2	0.0	0.0	0.0	100.0			Runqueue	4.0	Dirblk	0		
						Waitqueue		0.0				
Network	KBPS	I-Pack	O-Pack	KB-In	KB-Out			PAGING		MEMORY		
en0	0.4	4.0	1.0	0.3	0.1			Faults	0	Real,MB	8191	
lo0	0.0	0.0	0.0	0.0	0.0			Steals	0	% Comp	9.2	
Disk	Busy%	KBPS	TPS	KB-Read	KB-Writ			PgspIn	0	% Noncomp	25.9	
hdisk0	0.0	0.0	0.0	0.0	0.0			PgspOut	0	% Client	0.5	
hdisk1	0.0	0.0	0.0	0.0	0.0			PageIn	0			
								PageOut	0	PAGING SPACE		
WLM-Class (Active)	CPU%	Mem%	Disk-I/O%					Sios	0	Size,MB	2048	
Default	25	0	0							% Used	9.1	
java	0	2	0					NFS (calls/sec)			% Free	90.8

http	0	0	0	ServerV2	0	
System	0	29	0	ClientV2	0	Press:
Shared	0	0	0	ServerV3	0	"h" for help
Unmanaged	0	3	0	ClientV3	0	"q" to quit
Unclassified	0	1	0			

Name	PID	CPU%	PgSp	Class
cwhet_c	24680	6.2	0.0	Default
cwhet_b	34244	6.2	0.0	Default
cwhet_a	31492	6.2	0.0	Default
cwhet_d	17970	6.2	0.0	Default
java	15712	0.0	0.2	java
topas	21914	0.0	1.7	System
gil	5676	0.0	0.0	System
db2dasrrm	8516	0.0	1.6	Default



The gprof, pprof, prof, and tprof commands

This chapter discusses CPU profiling tools that are available in AIX 5L. The output of these tools can be used for analyzing the behavior of a system or a program at a given time period. This is very useful for:

- ▶ Providing a baseline of CPU activity
- ▶ Debugging performance problems and bottlenecks

19.1 CPU profiling tools

The **gprof** command produces an execution profile of C, Pascal, FORTRAN, or COBOL programs (with or without the source). The effect of called routines is incorporated into the profile of each caller. The **gprof** command is useful in identifying how a program consumes CPU resource. To find out which functions (routines) in the program are using the CPU, you can profile the program with the **gprof** command. **gprof** is a subset of the **prof** command.

The **gprof** command is useful for determining the following:

- ▶ Shows how much CPU time a program uses
- ▶ Helps to identify active areas of a program
- ▶ Profiles a program by routine
- ▶ Profiles parent-child

The **gprof** command resides in `/usr/ccs/bin/gprof`, is linked from `/usr/bin/gprof`, and is part of the `bos.adt.prof` fileset, which is installable from the AIX base installation media.

A similar profiler, named **xprofiler**, providing a Graphical User Interface (GUI) is available as part of the IBM Parallel Environment for AIX. The **xprofiler** can be used to profile both serial and parallel applications. From the **xprofiler** GUI, the same command line flags as for **gprof** can be used. The **xprofiler** command resides in `/usr/lpp/ppe.xprofiler/bin`, is linked to `/usr/bin`, and is part of the `ppe.xprofiler` fileset, which is installable from the IBM Parallel Environment installation media.

The **pprof** command reports on all kernel threads running within an interval using the **trace** utility. The **pprof** command is useful for determining the CPU usage for processes and their associated threads.

The **pprof** command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

The **prof** command displays object file profile data. This is useful for determining where in an executable most of the time is spent. The **prof** command interprets profile data collected by the monitor subroutine for the object program file (`a.out` by default).

The **prof** command resides in `/usr/ccs/bin`, is linked from `/usr/bin`, and is part of the `bos.adt.prof` fileset, which is installable from the AIX base installation media.

The **tprof** command reports CPU usage for individual programs and the system as a whole. This command is a useful tool for anyone with a Java, C, C++, or

FORTRAN program that might be CPU-bound and who wants to know which sections of the program are most heavily using the CPU.

The **tprof** command can charge CPU time to object files, processes, threads, subroutines (user mode, kernel mode, and shared library), and even to source lines of programs or individual instructions. Charging CPU time to subroutines is called profiling and charging CPU time to source program lines is called micro-profiling.

For subroutine-level profiling, the **tprof** command can be run without modifying executable programs (that is, no recompilation with special compiler flags is necessary). This is still true if the executables have been striped, unless the back tables have also been removed. However, recompilation is required to get a micro-profile, unless a listing file is already available. To perform micro-profiling on a program, either the program should be compiled with **-g** and the source files should be accessible to **tprof** or the program should be compiled with **-qlist** and either both the object listing files and the source files or just the object listing files should be accessible to **tprof**. To take full advantage of **tprof** micro-profiling capabilities, it is best to provide both the **.lst** and the source file.

The **tprof** command resides in **/usr/bin** and is part of the **bos.perf.tools** fileset, which is installable from the AIX base installation media.

19.1.1 Comparison of **tprof** versus **prof** and **gprof**

The most significant differences between these three commands is that **tprof** collects data with no impact on the execution time of the programs being profiled and works on optimized and striped binaries without any need for recompilation except to generate micro-profiling reports. Neither **gprof** nor **prof** have micro-profiling capabilities or work on optimized or striped binaries, but they do require special compilation flags and induce a slowdown in the execution time that can be significant.

The **prof** and **gprof** tools are standard, supported profiling tools on many UNIX systems, including in AIX 5L. Both **prof** and **gprof** provide subprogram profiling and exact counts of the number of times every subprogram is called. The **gprof** command also provides a very useful call graph showing the number of times each subprogram was called by a specific parent and the number of times each subprogram called a child. The **tprof** command provides neither subprogram call counts nor call graph information.

All of these tools, **tprof**, **prof**, and **gprof** commands, obtain the CPU consumption estimates for each subprogram by sampling the program counter of the user program.

19.2 gprof

The syntax of the **gprof** command is:

```
gprof [-b][-s][-z][-e Name][-E Name][-f Name][-F Name][-L PathName][gmon.out .]
```

Flags

- b** Suppresses the printing of a description of each field in the profile. This is very useful when you have learned what the descriptions for each field are.
- E Name** Suppresses the printing of the graph profile entry for routine Name and its descendants, similar to the -e flag, but excludes the time spent by routine Name and its descendants from the total and percentage time computations.
- e Name** Suppresses the printing of the graph profile entry for routine Name and all of its descendants (unless they have other ancestors that are not suppressed). More than one -e flag can be given. Only one routine can be specified with each -e flag.
- F Name** Prints the graph profile entry of the routine Name and its descendants similar to the -f flag, but uses only the times of the printed routines in total time and percentage computations. More than one -F flag can be given. Only one routine can be specified with each -F flag. The -F flag overrides the -E flag.
- f Name** Prints the graph profile entry of the specified routine Name and its descendants. More than one -f flag can be given. Only one routine can be specified with each -f flag.
- L PathName** Uses an alternate pathname for locating shared objects.
- s** Produces the gmon.sum profile file, which represents the sum of the profile information in all of the specified profile files. This summary profile file may be given to subsequent executions of the **gprof** command (using the -s flag) to accumulate profile data across several runs of an a.out file.
- z** Displays routines that have zero usage (as indicated by call counts and accumulated time).

These are the parameters:

- Name** Suppresses reporting or displays profile of the Name routine.
- PathName** Pathname for locating shared objects.
- gmon.out** Call graph profile file.

19.2.1 Information about measurement and sampling

The profile data is taken from the call graph profile file (`gmon.out` by default) created by programs compiled with the `cc` command using the `-pg` flags. These flags also link in versions of library routines compiled for profiling, and read the symbol table in the named object file (`a.out` by default), correlating it with the call graph profile file. If more than one profile file is specified, the `gprof` command output shows the sum of the profile information in the given profile files.

The `-pg` flag causes the compiler to insert a call to the `mcount` subroutine into the object code generated for each recompiled function of your program. During program execution, each time a parent calls a child function, the child calls the `mcount` subroutine to increment a distinct counter for that.

Note: Symbols from C++ object files have their names demangled before they are used.

The `gprof` command produces three items:

- ▶ A listing showing the functions sorted according to the time they represent, including the time of their call-graph descendents. (See “Detailed function report” on page 302.) Below each function entry are its (direct) call-graph children, with an indication of how their times are propagated to this function. A similar display above the function shows how the time of the function and the time of its descendents are propagated to its (direct) call-graph parents.
- ▶ A flat profile (see “Flat profile” on page 305) similar to that provided by the `prof` command (19.4, “prof” on page 320). This listing gives total execution times and call counts for each of the functions in the program, sorted by decreasing time. The times are then propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of that cycle.
- ▶ A summary of cross-references found during profiling. (See “Listing of cross references” on page 306.)

19.2.2 Profiling with the fork and exec subroutines

Profiling using the `gprof` command is problematic if your program runs the `fork` or `exec` subroutine on multiple, concurrent processes. Profiling is an attribute of the environment of each process, so if you are profiling a process that forks a new process, the child is also profiled. However, both processes write a `gmon.out` file in the directory from which you run the parent process, overwriting one of them.

tprof is recommended for multiple-process profiling. See 19.5, “tprof” on page 324 for more details.

If you must use the **gprof** command, one way around this problem is to call the `chdir` subroutine to change the current directory of the child process. Then, when the child process exits, its `gmon.out` file is written to the new directory.

19.2.3 Examples for gprof

This section shows an example of the **gprof** command in use. Two scenarios are shown: one is when the source code of the program we wish to profile is available and the other is when it is unavailable.

Profiling when the source code is available

The following example uses the source file `cwhet.c`, which is a standard benchmarking program. The source code is provided in “`cwhet.c`” on page 968.

The first step is to compile the `cwhet.c` source code into a binary using:

```
xlc -o cwhet_pg -pg -qarch=auto -qtune=auto -lm -O3 cwhet.c
```

Then create the `gmon.out` file, which will be used by **gprof**, by running **cwhet**:

```
cwhet_pg
```

Then run **gprof** on the executable using:

```
gprof cwhet_g > cwhet_pg.gprof
```

Detailed function report

Now the `cwhet_pg.gprof` file can be examined. Lines in the report have been removed to keep the report to a presentable size in Example 19-1.

Example 19-1 Output of gprof run on cwhet with source

```
$ cat cwhet_pg.gprof
...(lines omitted)...
granularity: each sample hit covers 4 byte(s) Time: 1060.13 seconds
```

index	%time	self	descendents	called/total called+self called/total	parents name children	index
		186.94	692.05	1/1	._start [2]	
[1]	82.9	186.94	692.05	1	.main [1]	
		164.69	0.00	140000000/140000000	.mod3 [3]	
		154.18	0.00	1920000000/1920000000	.cos [5]	
		134.64	0.00	930000000/930000000	.log [6]	
		90.87	0.00	930000000/930000000	.exp [7]	

```

66.03      0.00 640000000/640000000   .atan [8]
56.24      0.00 640000000/640000000   .sin [9]
18.43      0.00 1865032704/1865032704 .mod9 [10]
6.97       0.00 400065408/400065408   .mod8 [11]
0.00       0.00      10/10           .pout [20]

```

```

-----
                                <spontaneous>
[2]   82.9   0.00   878.99   .__start [2]
      186.94  692.05   1/1     .main [1]
      0.00   0.00   1/1     .exit [33]
...(lines omitted)...

```

In the example, the first index [1] in the left-hand column shows that the `.main` function is the current function. It was started by `.__start` (the parent function is above the current function), and it, in turn, calls `.mod3` and `.mod8` (the child functions are beneath the current function). All time of `.main` is propagated to `.__start` (in this case 186.94s). The self and descendents columns of the children of the current function should add up to the descendents' entry for the current function.

The following descriptions apply to the report in Example 19-1 on page 302.

The sum of self and descendents is the major sort for this listing. The following fields are included:

index	The index of the function in the call graph listing, as an aid to locating it
%time	The percentage of the total time of the program accounted for by this function and its descendents
self	The number of seconds spent in this function itself
descendents	The number of seconds spent in the descendents of this function on behalf of this function
called	The number of times this function is called (other than recursive calls)
self	The number of times this function calls itself recursively
name	The name of the function, with an indication of its membership in a cycle, if any
index	The index of the function in the call graph listing, as an aid to locating it

The following parent listings are included:

self1	The number of seconds of this function's self time that is due to calls from this parent.
descendents1	The number of seconds of this function's descendent time that is due to calls from this parent.
called2	The number of times this function is called by this parent. This is the numerator of the fraction that divides up the function's time to its parents.
total1	The number of times this function was called by all of its parents. This is the denominator of the propagation fraction.
parents	The name of this parent, with an indication of the parent's membership in a cycle, if any.
index	The index of this parent in the call graph listing as an aid in locating it.

The following children listings are included:

self ¹	The number of seconds of this child's self time that is due to being called by this function.
descendent1	The number of seconds of this child's descendent's time that is due to being called by this function.
called	The number of times this child is called by this function. This is the numerator of the propagation fraction for this child. Static-only parents and children are indicated by a call count of zero.
total1	The number of times this child is called by all functions. This is the denominator of the propagation fraction.
children	The name of this child and an indication of its membership in a cycle, if any.
index	The index of this child in the call graph listing, as an aid to locating it.
cycle listings	The cycle as a whole is listed with the same fields as a function entry. Below it are listed the members of the cycle, and their contributions to the time and call counts of the cycle.

¹ This field is omitted for parents (or children) in the same cycle as the function. If the function (or child) is a member of a cycle, the propagated times and propagation denominator represent the self time and descendent time of the cycle as a whole.

Flat profile

The flat profile sample is the second part of the `cwhet_pg.gprof` report. Example 19-2 is a flat file produced by the `gprof` command.

Example 19-2 Flat profile report of profiled cwhet.c

```
$ cat cwhet_pg.gprof
...(lines omitted)...
granularity: each sample hit covers 4 byte(s) Total time: 1060.13 seconds

%   cumulative   self           self   total
time seconds  seconds  calls  ms/call  ms/call  name
17.6   186.94    186.94         1 186940.00 878990.00  .main [1]
15.5   351.63    164.69 140000000      0.00    0.00  .mod3 [3]
15.2   512.46    160.83
      666.64    154.18 192000000      0.00    0.00  .__mcount [4]
14.5   666.64    154.18 192000000      0.00    0.00  .cos [5]
12.7   801.28    134.64 930000000      0.00    0.00  .log [6]
8.6    892.15     90.87 930000000      0.00    0.00  .exp [7]
6.2    958.18     66.03 640000000      0.00    0.00  .atan [8]
5.3   1014.42     56.24 640000000      0.00    0.00  .sin [9]
1.7   1032.85     18.43 1865032704      0.00    0.00  .mod9 [10]
0.7   1039.82      6.97 400065408      0.00    0.00  .mod8 [11]
0.6   1046.63      6.81
      1053.41      6.78
      1060.13      6.72
0.0   1060.13      0.00      44     0.00    0.00  .__stack_pointer [14]
      .mf2x2 [15]

...(lines omitted)...
```

The example shows the flat profile, which is less complex than the call-graph profile. It is very similar to the output of `prof`. See 19.4, “`prof`” on page 320. The primary columns of interest are the `self seconds` and the `calls` columns, as these reflect the CPU seconds spent in each function and the number of times each function was called. The `self ms/call` is the amount of CPU time used by the body of the function itself, and `total ms/call` means time in the body of the function plus any descendent functions called.

To enhance performance you would normally investigate the functions at the top of the report. You should also consider how many calls are made to the function. Sometimes it may be easier to make slight improvements to a frequently called function than to make extensive changes to a piece of code called once.

These descriptions apply to the flat profile report in Example 19-2:

<code>% time</code>	The percentage of the total running time of the program used by this function.
<code>cumulative seconds</code>	A running sum of the number of seconds accounted for by this function and those listed above it.

self seconds	The number of seconds accounted for by this function alone. This is the major sort for this listing.
calls	The number of times this function was invoked, if this function is profiled. Otherwise this column remains blank.
self ms/call	The average number of milliseconds spent in this function per call, if this function is profiled. Otherwise this column remains blank.
total ms/call	The average number of milliseconds spent in this function and its descendents per call, if this function is profiled. Otherwise this column remains blank.
name	The name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parentheses it shows where it would appear in the gprof listing if it were to be printed.

Listing of cross references

A cross-reference index, as shown in Example 19-3, is the last item produced summarizing the cross references found during profiling.

Example 19-3 Cross-references index report of profiled cwhet.c

```
$ cat cwhet_pg.gprof
...(lines omitted)...
Index by function name

[27] ._flsbuf           [7] .exp                [36] .moncontrol
[28] ._ioctl            [24] .free               [37] .monitor
[4]  ._mcount          [25] .free_y            [16] .myecvt
[17] ._nl_langinfo_std  [26] .free_y_heap       [19] .nl_langinfo
[14] ._stack_pointer    [34] .ioctl             [20] .pout
[18] ._doprnt           [35] .isatty            [38] .pre_ioctl
[29] ._findbuf          [6] .log                [21] .printf
[30] ._flsbuf           [1] .main              [12] .qincrement
[31] ._wrtchk           [22] .mf2x1             [13] .qincrement1
[32] ._xflsbuf          [15] .mf2x2             [9] .sin
[8]  .atan              [3] .mod3              [23] .splay
[5]  .cos               [11] .mod8
[33] .exit              [10] .mod9
```

The report is an alphabetical listing of the cross references found during profiling.

Profiling when the source code is unavailable

If you do not have the source code for your program (in this case `cwhet.c`, which can be seen in “`cwhet.c`” on page 968.) then you can profile using the `gprof` command without recompiling, but you will still need the object for `cwhet`. You

must be able to relink your program modules with the appropriate compiler command (for example, `cc` for C program source files). If you do not recompile, you do not get call frequency counts, although the flat profile is still useful without them. The following sequence explains how to perform the profiling:

The first step is to compile `whet.c` source into a binary using:

```
cc -c -o whet.o whet.c
```

We then rename the source code:

```
mv whet.c whet.c_disappaer
```

Re-link the object file with the `-pg` option:

```
cc -pg -lm whet.o -L/lib -L/usr/lib -o whet_nosrc
```

Then create the `gmon.out` (which will be used by `gprof`) by running `whet_nosrc` as follows:

```
whet_nosrc > whet.out
```

Then run `gprof` on the executable using:

```
gprof whet_nosrc > whet_nosrc.gprof
```

You will get the following error, which can be ignored:

```
Warning: mon.out file has no call counts. Program possibly not compiled with profiled libraries.
```

Now the `whet_nosrc.gprof` file can be examined. Example 19-4 shows an excerpt of the file. Lines in the report have been removed to keep the report to a presentable size.

Example 19-4 Report of profiled whet.c without call counts

```
$ cat whet_nosrc.gprof
... (lines omitted)...
```

```
granularity: each sample hit covers 4 byte(s) Time: 1063.20 seconds
```

index	%time	self	descendents	called/total called+self called/total	parents name children	index
[1]	26.3	279.25	0.00		<spontaneous> .main [1]	

[2]	18.5	196.95	0.00		<spontaneous> .mod3 [2]	

```

-----
[3]    11.5  122.04    0.00    <spontaneous>
      .log [3]
-----
[4]     9.9  105.77    0.00    <spontaneous>
      .sqrt [4]
....( line omitted) ....
-----
[8]     4.8   51.20    0.00    <spontaneous>
      .mod9 [8]
-----
[9]     2.9   30.43    0.00    <spontaneous>
      .mod8 [9]
... (lines omitted)...
the gprof listing if it were to be printed.
^L
granularity: each sample hit covers 4 byte(s) Total time: 1063.20 seconds

```

%	cumulative	self	self	total	name
time	seconds	seconds	calls	ms/call	ms/call
26.3	279.25	279.25			.main [1]
18.5	476.20	196.95			.mod3 [2]
11.5	598.24	122.04			.log [3]
9.9	704.01	105.77			.sqrt [4]
9.3	802.58	98.57			.cos [5]
8.8	896.14	93.56			.exp [6]
6.0	960.13	63.99			.atan [7]
4.8	1011.33	51.20			.mod9 [8]
2.9	1041.76	30.43			.mod8 [9]
2.0	1063.19	21.43			.sin [10]
0.0	1063.20	0.01			.nl_langinfo [11]

```

^L
Index by function name
      [7] .atan          [1] .main          [11] .nl_langinfo
      [5] .cos          [2] .mod3          [10] .sin
      [6] .exp          [9] .mod8          [4] .sqrt
      [3] .log          [8] .mod9

```

In this example, look at the first index [1] in the left-hand column. This shows that the .main function is the current function. It, in turn, calls .mod3, mod8, and .mod9 (the child functions are beneath the current function).

As can be seen by comparing Example 19-4 on page 307 with the one generated in Example 19-1 on page 302, where the source code was available, the report does not produce statistics on the average number of milliseconds spent in a function per call and its descendents' per call.

19.3 pprof

The syntax of the **pprof** command is:

```
pprof { <time> | -I <pprof.flow file> | -i <tracefile> | -d } [ -s ]  
[ -n ] [ -f ] [ -p ] [ -T <size> ] [ -v ]
```

Flags

-l	Generate reports from a previously generated pprof.flow.
-i	Generate reports from a previously generated tracefile.
-d	Waits for the user to execute trcon and trcstop commands from the command line.
-T	Sets trace kernel buffer size (default 32000 bytes).
-v	Verbose mode (print extra details).
-n	Just generate pprof.namecpu.
-f	Just generate pprof.famcpu and pprof.famind.
-p	Just generate pprof.cpu.
-s	Just generate pprof.start.
-w	Just generate pprof.flow.

Parameters

time	Amount of seconds to trace the system
pprof.flow file	Name of the previously generated pprof.flow file
tracefile	Name of the previously generated trace file
size	Kernel buffer size (default 32000 bytes)

19.3.1 Information about measurement and sampling

The **pprof** command reports on all kernel threads running within an interval using the trace utility. The raw process information is saved in the file; five reports are also generated.

The following types of reports are produced by **pprof**:

▶ **pprof.cpu**

Lists all kernel-level threads sorted by actual CPU time. Contains:

- Process Name
- Process ID
- Parent Process ID
- Process State at Beginning and End
- Thread ID
- Parent Thread ID
- Actual CPU Time
- Start Time
- Stop Time
- The difference between the Stop time and the Start time

▶ **pprof.start**

Lists all kernel threads sorted by start time. Contains:

- Process Name
- Process ID
- Parent Process ID
- Process State Beginning and End
- Thread ID
- Parent Thread ID
- Actual CPU Time
- Start Time
- Stop Time
- The difference between the Stop time and the Start time

▶ **pprof.namecpu**

Lists information about each type of kernel thread (all executable with the same name). Contains:

- Process Name
- Number of Threads
- CPU Time
- % of Total CPU Time

▶ **pprof.famind**

Lists all processes grouped by families (processes with a common ancestor). Child process names are indented with respect to the parent. Contains:

- Start Time
- Stop Time
- Actual CPU Time
- Process ID
- Parent Process ID

- Thread ID
 - Parent Thread ID
 - Process State at Beginning and End
 - Level
 - Process Name
- `pprof.famcpu`
- Lists the information for all families (processes with a common ancestor). The Process Name and Process ID for the family is not necessarily the ancestor. Contains:
- Start Time
 - Process Name
 - Process ID
 - Number of Threads
 - Total CPU Time

The trace hooks used by `pprof` are 000, 001, 002, 003, 005, 006, 135, 106, 10C, 134, 139, 465, 467, and 00A. This can be seen when you run the `pprof` command: if you execute `ps`, it will display a `trace` process running, tracking those trace hooks. See Appendix B, “Trace hooks” on page 973 for details of the trace hooks used.

The `trace` process is automatically started and stopped by `pprof` if you are not postprocessing an existing trace file.

19.3.2 Examples for `pprof`

To profile the CPU or CPUs of a system for 60 seconds, use the `pprof 60` command to generate the reports shown in this section.

The `pprof.cpu` report

Example 19-5 shows the `pprof.cpu` file produced when running the `pprof 60` command.

Example 19-5 The `pprof.cpu` report

```
# cat pprof.cpu
```

```
Pprof CPU Report
```

```
Sorted by Actual CPU Time
```

```
From: Thu Apr 10 16:39:12 2001
```

```
To: Thu Apr 10 16:40:12 2001
```

E = Exec'd F = Forked
 X = Exited A = Alive (when traced started or stopped)
 C = Thread Created

Pname	PID	PPID	BE	TID	PTID	ACC_time	STT_time	STP_time	STP-STT
dc	25294	19502	AA	39243	0	32.876	0.005	60.516	60.511
dc	26594	26070	AA	45947	0	29.544	0.020	60.521	60.501
cpu	27886	29420	AA	48509	0	29.370	0.011	60.532	60.521
cpu	29156	29420	AA	49027	0	29.119	0.000	60.529	60.529
cpu	28134	29420	AA	40037	0	28.629	0.008	60.532	60.525
cpu	29420	26326	AA	47483	0	26.157	0.015	60.526	60.511
dc	25050	21710	AA	36785	0	24.466	0.005	60.504	60.499
cpu	28646	29420	AA	48767	0	17.772	0.013	60.514	60.501
dc	26834	25812	AA	46187	0	17.654	0.023	60.494	60.471
seen+done	28904	23776	EA	48005	40515	0.932	29.510	60.533	31.023
X	4224	4930	AA	5493	0	0.849	0.210	44.962	44.751
xmtrand	20804	19502	AA	31173	0	0.754	0.305	59.665	59.360
seen+done	30964	23776	EA	50575	40515	0.749	33.780	60.533	26.753
seen+done	31218	23776	EA	50829	40515	0.635	36.328	60.533	24.205
aixterm	24786	5510	AA	40257	0	0.593	0.215	42.394	42.179
seen+done	30446	23776	EA	49799	40515	0.544	35.124	60.513	25.389
java	22376	23036	AA	33523	0	0.358	42.416	44.957	2.541
netm	1548	0	AA	1549	0	0.144	41.582	60.533	18.952
...(lines omitted)...									
gil	2322	0	AA	3355	0	0.022	0.100	60.359	60.259
trcstop	30188	-1	EA	49541	0	0.020	60.510	60.534	0.024
sadc	18272	22170	AX	33265	0	0.007	49.282	49.309	0.028
trace	30444	30186	AX	49797	0	0.007	0.001	0.005	0.004
nfsd	10068	1	AA	16777	0	0.006	0.069	60.148	60.079
ksh	28902	23776	FE	48003	40515	0.006	19.230	19.238	0.008
j2pg	5972	0	AA	11875	0	0.005	30.123	58.400	28.277
j2pg	5972	0	AA	11123	0	0.004	29.963	57.532	27.569
j2pg	5972	0	AA	9291	0	0.004	30.875	60.234	29.359
j2pg	5972	0	AA	8533	0	0.004	30.063	57.722	27.659
...(lines omitted)...									
pprof	29930	26326	AA	49283	0	0.000	0.000	0.000	0.000
						=====			
						242.116			

The following values are displayed for this report:

- Pname The name of the process.
- PID The Process ID as it would appear with the **ps** command.
- PPID The Parent Process ID (the process to which it belongs).

BE	The thread when profiling with pprof began (B) and when profiling ended (E). The following options apply to this field:
E	The thread was executed.
F	The process was forked.
X	The process exited.
A	The process was alive (when traced started or stopped).
C	The thread was created.
TID	Thread ID.
PTID	Parent Thread ID; that is, which thread it belongs to.
ACC_time	Actual CPU time in milliseconds.
STT_time	Process starting time in milliseconds.
STP_time	Process stop time in milliseconds.
STP-STT	Process stop time less the process start time.

This report lists all kernel level threads sorted by actual CPU time, showing that the processes called `dc` and `cpu` were consuming the most CPU time. By looking at the process ID, 25294, we can see that the STP-STT is 60.511 ms and the ACC_time is 32.876 ms, indicating that since the process started, 50 percent of that time has been used running on the CPU. The report also shows with (BE=AA) that the thread was active both at the beginning of the trace and at the end.

The most important fields in this report are ACC_time and STP-STT. If the CPU time (ACC_TIME) was high in proportion to the length of time the thread was running (STP-STT), as is this case for the `dc` and `cpu` processes in the above example, then the process should be investigated further with the **gprof** command to look at the amount of CPU time the functions of the process are using. This will help in any diagnosis to improve performance of the process. Refer to Chapter 19, “The gprof, pprof, prof, and tprof commands” on page 297 for more details.

In the report above, you will see that some of the process names are listed as UNKNOWN and the PPID is -1. This is because **pprof** reports on all kernel threads running within an interval using the **trace** utility. If some processes had executed or the thread had been created before **trace** utility started, their processes’ name and PPID would not be caught in thread record hash tables that are read by **pprof**. In this case, **pprof** would assign -1 as PPID to those processes.

The pprof.start report

Example 19-6 on page 314 shows the pprof.start file produced when running the **pprof 60** command.

Example 19-6 The pprof.start report

```
# cat pprof.start
```

Pprof START TIME Report

Sorted by Start Time

From: Thu Apr 10 16:39:12 2001

To: Thu Apr 10 16:40:12 2001

E = Exec'dF = Forked
X = ExitedA = Alive (when traced started or stopped)
C = Thread Created

Pname	PID	PPID	BE	TID	PTID	ACC_time	STT_time	STP_time	STP-STT
cpu	29156	29420	AA	49027	0	29.119	0.000	60.529	60.529
pprof	29930	26326	AA	49283	0	0.000	0.000	0.000	0.000
UNKNOWN	29672	-1	EA	47739	0	0.000	0.001	0.005	0.004
UNKNOWN	28386	-1	EA	48253	0	0.000	0.001	0.001	0.000
UNKNOWN	28386	-1	CA	50313	48253	0.012	0.001	57.780	57.780
trace	30444	30186	AX	49797	0	0.007	0.001	0.005	0.004
dc	25050	21710	AA	36785	0	24.466	0.005	60.504	60.499
dc	25294	19502	AA	39243	0	32.876	0.005	60.516	60.511
cpu	28134	29420	AA	40037	0	28.629	0.008	60.532	60.525
init	1	0	AA	259	0	0.011	0.010	49.379	49.369
cpu	27886	29420	AA	48509	0	29.370	0.011	60.532	60.521
cpu	28646	29420	AA	48767	0	17.772	0.013	60.514	60.501
cpu	29420	26326	AA	47483	0	26.157	0.015	60.526	60.511
dc	26594	26070	AA	45947	0	29.544	0.020	60.521	60.501
dc	26834	25812	AA	46187	0	17.654	0.023	60.494	60.471
nfsd	10068	1	AA	16777	0	0.006	0.069	60.148	60.079
i4llmd	17552	9034	AA	5979	0	0.121	0.069	60.450	60.381
java	22376	23036	AA	37949	0	0.129	0.091	60.310	60.219
gil	2322	0	AA	3355	0	0.022	0.100	60.359	60.259
gil	2322	0	AA	2839	0	0.025	0.175	60.147	59.972
X	4224	4930	AA	5493	0	0.849	0.210	44.962	44.751
aixterm	24786	5510	AA	40257	0	0.593	0.215	42.394	42.179
ksh	23776	24786	AA	40515	0	0.046	0.220	39.129	38.909
gil	2322	0	AA	2581	0	0.024	0.230	60.379	60.149
swapper	0	0	AA	3	0	0.052	0.238	59.757	59.520
gil	2322	0	AA	3097	0	0.024	0.240	59.939	59.699
syncd	5690	1	AA	7239	0	0.135	0.275	0.498	0.223
UNKNOWN	30960	-1	EX	50571	0	0.000	0.275	0.285	0.009
xmtrend	20804	19502	AA	31173	0	0.754	0.305	59.665	59.360

...(lines omitted)...

=====
242.116

This report lists all kernel level threads sorted by start time. It shows the process and thread status at the beginning of the trace.

For a description of the report fields and analysis, refer to “The pprof.cpu report” on page 311.

The pprof.namecpu report

Example 19-7 shows the pprof.namecpu file produced when running the **pprof 60** command.

Example 19-7 The pprof.namecpu report

```
# cat pprof.namecpu
```

```
Pprof PROCESS NAME Report
```

```
Sorted by CPU Time
```

```
From: Thu Apr 10 16:39:12 2001
```

```
To: Thu Apr 10 16:40:12 2001
```

Pname	#ofThreads	CPU_Time	%
=====	=====	=====	=====
cpu	5	131.047	54.126
dc	4	104.540	43.178
seen+done	4	2.860	1.181
X	1	0.849	0.351
xmtrend	1	0.754	0.311
aixterm	2	0.594	0.245
java	3	0.489	0.202
netm	2	0.146	0.060
syncd	1	0.135	0.056
i411md	2	0.121	0.050
ksh	6	0.113	0.047
gil	5	0.095	0.039
UNKNOWN	7	0.073	0.030
j2pg	17	0.068	0.028
swapper	1	0.052	0.021
dtwm	1	0.050	0.021
snmpd	1	0.034	0.014
trcstop	1	0.020	0.008
ls	1	0.020	0.008

init	1	0.011	0.005
ttssession	1	0.008	0.003
trace	1	0.007	0.003
sadc	1	0.007	0.003
rpc.lockd	3	0.007	0.003
nfsd	3	0.006	0.002
bsh	1	0.004	0.002
dtstyle	1	0.002	0.001
IBM.AuditRMD	1	0.001	0.000
cron	1	0.001	0.000
rmcd	2	0.001	0.000
sendmail	1	0.001	0.000
PM	1	0.000	0.000
pprof	1	0.000	0.000
IBM.ERrmd	1	0.000	0.000
rtcnd	1	0.000	0.000
hostmibd	1	0.000	0.000
=====	=====	=====	=====
	87	242.116	100.000

This report lists information about each type of kernel thread using these fields:

Pname	The name of the process
#ofThreads	The number of threads created by the process
CPU_Time	The amount of CPU time consumed by the thread
%	The percentage of CPU time consumed by the thread

The report shows that the `cpu` and `dc` processes are using the most CPU time. Each line of the report represents all processes called `Pname` on the system. For example, the five threads called `cpu` are combined to show as one in the report. The number of threads per process is shown under the `#ofThreads` column.

The `pprof.famind` report

Example 19-8 shows the `pprof.famind` file produced when running the `pprof 60` command.

Example 19-8 The `pprof.famind` report

Pprof PROCESS FAMILY Report - Indented

Sorted by Family and Start Time

From: Thu Apr 10 16:39:12 2001

To: Thu Apr 10 16:40:12 2001

E = Exec'd F = Forked
X = Exited A = Alive (when traced started or stopped)
C = Thread Created

STT	STP	ACC	PID	PPID	TID	PTID	BE	LV	PNAME
=====	=====	=====	=====	=====	=====	=====	==	==	=====
0.010	49.379	0.011	1	0	259	0	AA	0	init
0.238	59.757	0.052	0	0	3	0	AA	0	swapper
0.000	60.529	29.119	29156	29420	49027	0	AA	2	.. cpu
0.008	60.532	28.629	28134	29420	40037	0	AA	2	.. cpu
0.011	60.532	29.370	27886	29420	48509	0	AA	2	.. cpu
0.013	60.514	17.772	28646	29420	48767	0	AA	2	.. cpu
0.015	60.526	26.157	29420	26326	47483	0	AA	1	. cpu
0.000	0.000	0.000	29930	26326	49283	0	AA	2	.. pprof
0.001	0.005	0.000	29672	-1	47739	0	EA	2	.. UNKNOWN
0.001	0.001	0.000	28386	-1	48253	0	EA	2	.. UNKNOWN
0.001	57.780	0.012	28386	-1	50313	48253	CA	2	..- UNKNOWN
0.001	0.005	0.007	30444	30186	49797	0	AX	2	.. trace
0.005	60.504	24.466	25050	21710	36785	0	AA	2	.. dc
0.005	60.516	32.876	25294	19502	39243	0	AA	2	.. dc
0.020	60.521	29.544	26594	26070	45947	0	AA	2	.. dc
0.023	60.494	17.654	26834	25812	46187	0	AA	2	.. dc
0.069	60.148	0.006	10068	1	16777	0	AA	2	.. nfsd
7.453	7.453	0.000	10068	1	17035	0	AA	2	.. nfsd
7.537	38.046	0.000	10068	1	17289	0	AA	2	.. nfsd
0.069	60.450	0.121	17552	9034	5979	0	AA	2	.. i411md
14.930	14.930	0.000	5204	17552	26071	0	AA	3	... i411md
0.091	60.310	0.129	22376	23036	37949	0	AA	2	.. java
42.410	44.948	0.002	22376	23036	36885	0	AA	2	.. java
42.416	44.957	0.358	22376	23036	33523	0	AA	2	.. java
0.100	60.359	0.022	2322	0	3355	0	AA	2	.. gil
0.175	60.147	0.025	2322	0	2839	0	AA	2	.. gil
0.230	60.379	0.024	2322	0	2581	0	AA	2	.. gil
0.240	59.939	0.024	2322	0	3097	0	AA	2	.. gil
0.210	44.962	0.849	4224	4930	5493	0	AA	2	.. X

This report includes the following fields:

STT	The process start time.
STP	The process stop time.
ACC	The actual CPU time.
PID	The Process ID as it would appear with the ps command.
PPID	The Parent Process ID; that is, which process it belongs to.
TID	The Thread ID.
PTID	The Parent Thread ID; that is, which thread it belongs to.
BE	The state of the thread when profiling with pprof began (B) and when profiling ended (E). The following options apply to this field: E The thread was Executed. F The process was Forked. X The process Exited. A The process was alive (when trace started or stopped). C The thread was Created.
LV	The run level has the value of 0 - 9. It tells the init command to place the system in one of the run levels 0-9. When the init command requests a change to run levels 0-9, it kills all processes at the current run levels, then starts any processes associated with the new run levels. The value of these levels: 0-1 Reserved for future use of the operating system. 2 Contains all of the terminal processes and daemons that are run in the multiuser environment. In the multiuser environment, the <code>/etc/inittab</code> file is set up so that the init command creates a process for each terminal on the system. The console device driver is also set to run at all run levels so the system can be operated with only the console active. 3-9 Can be defined according to the user's preferences. More about run levels can be found in <i>AIX 5L Version 5.2 Commands Reference Volume 5</i> for the telinit command or at: http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/telinit.htm
PNAME	The name of the process.

The report shows the processes sorted by their ancestors (parents) and process name. It is useful for determining which processes have forked other processes.

By looking at the ACC column, you can ascertain how much CPU time was consumed by the process.

The pprof.famcpu report

Example 19-9 shows the pprof.famcpu file produced when running the **pprof 60** command.

Example 19-9 The pprof.famcpu report

Pprof PROCESS FAMILY SUMMARY Report

Sorted by CPU Time

From: Tue May 29 16:39:12 2001

To: Tue May 29 16:40:12 2001

Stt-Time	Pname	PID	#Threads	Tot-Time
=====	=====	=====	=====	=====
0.0000	cpu	29156	5	131.047
0.0051	dc	25294	1	32.876
0.0201	dc	26594	1	29.544
0.0048	dc	25050	1	24.466
0.0226	dc	26834	1	17.654
0.2151	aixterm	24786	12	3.584
0.2101	X	4224	1	0.849
0.3051	xmtrend	20804	1	0.754
0.0912	java	22376	3	0.489
41.5815	netm	1548	1	0.144
0.2751	syncd	5690	1	0.135
0.0690	i411md	17552	2	0.121
0.1001	gil	2322	4	0.095
29.6299	j2pg	5972	17	0.070
0.2376	swapper	0	1	0.052
16.3183	UNKNOWN	30708	1	0.050
12.2293	dtwm	23240	1	0.050
6.9476	snmpd	7746	1	0.034
60.5083	UNKNOWN	30188	2	0.022
0.0006	UNKNOWN	28386	2	0.012
0.0101	init	1	1	0.011
49.2815	sadc	18272	2	0.010
0.4880	UNKNOWN	30962	1	0.010
42.3540	ttsession	20172	1	0.008
0.4748	rpc.lockd	8016	3	0.007
0.0009	trace	30444	1	0.007
0.0690	nfsd	10068	3	0.006
0.8952	netm	2064	1	0.002
42.3864	dtstyle	4844	1	0.002
7.6887	rmcd	6464	2	0.001

11.0259	sendmail	9832	1	0.001
48.1527	cron	11370	1	0.001
42.3944	aixterm	16872	1	0.001
7.4702	IBM.AuditRmd	17034	1	0.001
8.0185	gil	1806	1	0.000
2.3701	IBM.ERrmd	17302	1	0.000
37.2309	hostmibd	10582	1	0.000
4.9364	PM	13676	1	0.000
0.2752	UNKNOWN	30960	1	0.000
0.5870	rtcmd	8258	1	0.000
0.0005	UNKNOWN	29672	1	0.000
0.0001	pprof	29930	1	0.000

=====
87 242.116

This report lists the processes with a common ancestor and shows them sorted by their ancestors (parents). It is useful for determining how many threads per process are running and how much CPU time the threads are consuming.

The following fields are listed:

Stt-Time	The process starting time
Pname	The name of the process
PID	The Process ID as it would appear with the ps command
#Threads	Number of threads created by the process
Tot-Time	The process stop time less the process start time

19.4 prof

The syntax of the **prof** command is:

```
prof [ -t | -c | -a | -n ] [ -o | -x ] [ -g ] [ -z ] [ -h ] [ -s ] [ -S ]
[ -v ] [ -L PathName ] [ Program ] [ -m MonitorData ... ]
```

Flags

The mutually exclusive flags **-a**, **-c**, **-n**, and **-t** determine how the **prof** command sorts the output lines; if multiple flags are specified, it uses only the first one:

-a	Sorts by increasing symbol address
-c	Sorts by decreasing number of calls
-n	Sorts alphabetically by symbol name
-t	Sorts by decreasing percentage of total time (default)

The mutually exclusive flags **o** and **x** specify how to display the address of each symbol monitored. If multiple flags are specified, it uses only the first one.

-o	Displays each address in octal, along with the symbol name
-----------	--

-x Displays each address in hexadecimal, along with the symbol name

Additional flags:

-g Includes non-global symbols (static functions).

-h Suppresses the heading normally displayed on the report. This is useful if the report is to be processed further.

-L PathName Uses alternate path name for locating shared objects.

-m MonitorData Takes profiling data from MonitorData instead of mon.out.

-s Produces a summary file called mon.sum. This is useful when more than one profile file is specified.

-S Displays a summary of monitoring parameters and statistics on standard error.

-v Suppresses all printing and sends a graphic version of the profile to standard output for display by the plot filters. When plotting, low and high numbers (by default, zero and 100) can be given to cause a selected percentage of the profile to be plotted with accordingly higher resolution.

-z Includes all symbols in the profile range, even if associated with zero calls and zero time.

Parameters

PathName Specifies the alternate path name for locating shared objects. Refer to the -L flag.

Program The name of the object file name to profile.

MonitorData Takes profiling data from MonitorData instead of mon.out.

19.4.1 Information about measurement and sampling

The **prof** command reads the symbol table in the Program object file and correlates it with the profile file (mon.out by default). The **prof** command displays, for each external text symbol, the percentage of execution time spent between the address of that symbol and the address of the next, the number of times that function was called, and the average number of milliseconds per call.

Note: Symbols from C++ object files have their names demangled before they are used.

To tally the number of calls to a function, you must have compiled the file using the **cc** command with the **-p** flag. The **-p** flag causes the compiler to insert a call to the **mcount** subroutine into the object code generated for each recompiled function of your program. While the program runs, each time a parent calls a child function the child calls the **mcount** subroutine to increment a distinct counter for that parent-child pair. Programs not recompiled with the **-p** flag do not have the **mcount** subroutine inserted and therefore keep no record of which function called them.

The **-p** flag also arranges for the object file to include a special profiling startup function that calls the **monitor** subroutine when the program begins and ends. The call to the **monitor** subroutine when the program ends actually writes the **mon.out** file. Therefore, only programs that explicitly exit or return from the main program cause the **mon.out** file to be produced.

The location and names of the objects loaded are stored in the **mon.out** file. If you do not select any flags, **prof** will use these names. You must specify a program or use the **-L** option to access other objects.

Note: Imported external routine calls, such as a call to a shared library routine, have an intermediate call to local **glink** code that sets up the call to the actual routine. If the timer clock goes off while running this code, time is charged to a routine called **routine.gl**, where **routine** is the routine being called. For example, if the timer goes off while in the **glink** code to call the **printf** subroutine, time is charged to the **printf.gl** routine.

19.4.2 Examples for **prof**

The examples in this section use the **cwhet.c** program that is shown in “**cwhet.c**” on page 968.

The first step of creating the following examples explaining **prof** is to compile the **cwhet.c** source into a binary using:

```
cc -o cwhet -p -lm cwhet.c
```

The **-p** flag of the **cc** compiler creates profiling support.

Then run **cwhet**. It creates **mon.out**, which **prof** will use for post processing. Run **prof** on the executable using:

```
prof -xg -s > cwhet.prof
```

This command creates two files:

cwhet.prof The **cwhet.prof** file, as specified in the command line, is shown in the following example.

mon.sum This is a summary report.

The cwhet.prof report

Example 19-10 shows the cwhet.prof file produced when running **prof**.

Example 19-10 The cwhet.prof report

```
# cat cwhet.prof
Address      Name                Time Seconds  Cumsecs  #Calls msec/call
10000868 .main                28.3      0.63     0.63      1      630.0
100005e8 .mod8                26.0      0.58     1.21 8990000    0.0001
10001528 .__mcount            7.6       0.17     1.38
10002930 .sqrt                7.2       0.16     1.54
10000550 .mod9                7.2       0.16     1.70 6160000    0.0000
10002328 .log                 5.8       0.13     1.83 930000    0.0001
10001e20 .cos                 4.5       0.10     1.93 1920000    0.0001
10000688 .mod3                4.5       0.10     2.03 140000    0.0007
100026a0 .exp                 2.7       0.06     2.09 930000    0.0001
10002088 .atan                2.2       0.05     2.14 640000    0.0001
10001660 .qincrement1        1.8       0.04     2.18
10001688 .qincrement          1.8       0.04     2.22
10001be8 .sin                 0.4       0.01     2.23 640000    0.0000
100007ac .pout                0.0       0.00     2.23      10      0.0
d24d4ab4 .__nl_langinfo_std  0.0       0.00     2.23      10      0.0
d24db95c .free                0.0       0.00     2.23       2      0.0
d24dd99c .isatty              0.0       0.00     2.23       1      0.0
d24ddf04 .__ioctl             0.0       0.00     2.23       1      0.0
d24de0c8 .pre_ioctl           0.0       0.00     2.23       1      0.0
d24de3f0 .ioctl               0.0       0.00     2.23       1      0.0
d24df8e8 ._flsbuf             0.0       0.00     2.23      90      0.0
..... (more lines).....
```

In this example, we can see that most of the calls are to the `.mod8` and `.mod9` routines; notice the time spent for each call. You could look at the `.mod8` and `.mod9` routines from the source code to see whether they could be rewritten more efficiently, or use the `inline` option in compiler to increase the performance.

The following columns are reported:

Address	The virtual address where the function is located
Name	The name of the function
Time	The percentage of the total running time of the time program used by this function
Seconds	The number of seconds accounted for by this function alone

Cumsecs	A running sum of the number of seconds accounted for by this function
#Calls	The number of times this function was invoked, if this function is profiled
msec/call	The average number of milliseconds spent in this function and its descendents per call, if this function is profiled

19.5 tprof

The syntax of the **tprof** command is:

```
tprof [ -c ] [ -C { all | CPUList } ] [ -d ] -D ] [ -e ] [ -F ] [ -j ] [ -k ]
[ -l ] [ -m ObjectsList ] [ -M SourcePathList ] [ -p ProcessList ]
[ -P { all | PIDsList } ] [ -s ] [ -S SearchPathList ] [ -t ] [ -T BufferSize ]
[ -u ] [ -v ] [ -V VerboseFileName ] [ -z ]
{ { -r RootString } | { [ -A { all | CPUList } ] [ -r RootString ] -x Program } }
```

In this syntax diagram, the following applies:

- ▶ All of the list type inputs are separated by a comma except for pathlist, which is separated by a colon.
- ▶ Per-CPU profiling mode is automatically disabled while running in realtime mode.
- ▶ Microprofiling is automatically disabled if per-CPU profiling is turned on.
- ▶ If the -x flag is specified without the -A flag, tprof runs in realtime mode.
- ▶ If the -x flag is specified with the -A flag, tprof runs in automated offline mode.
- ▶ If the -x flag is omitted tprof runs in post-processing mode or manual offline mode, depending on the presence of cooked files and the -F flag.

Flags

- A {all | CPUList}** Turns on automatic offline mode. No argument turns off per-CPU tracing. all enables tracing of all CPUs. CPUList is a comma separated list of CPU IDs to be traced.
- c** Turns on generation of cooked files.
- C all | CPU** Turns on per-CPU profiling. Specify all to generate profile reports for all CPUs. CPU numbers should be separated with a comma if you give a CPUlist (for example, 0,1,2). Per-CPU profiling is possible only if per-CPU trace is either on (in automated offline mode), or has been used (in manual offline mode). It is not possible at all in online mode.

- d** Turns on deferred tracing mode (defers data collection until **trcon** is called).
- D** Turns on detailed profiling, which displays CPU usage by instruction offset under each subroutine.
- e** Turns on kernel extension profiling.
- F** Overwrites cooked files if they exist. If used without the **-x** flag, this forces the manual offline mode.
- j** Turns on Java class and methods of profiling.
- k** Enables kernel profiling.
- l** Enables long names reporting. By default tprof truncates the subroutine, program, and source file names if they do not fit into the available space in the profiling report. This flag disables truncation.
- m ObjectList** Enables micro-profiling of objects in the comma-separated list ObjectList. Executables, shared libraries, and kernel extensions can be micro-profiled. Specify the archive name for libraries and kernel extensions. To enable micro-profiling of programs, user mode profiling (**-u**) must be turned on. To enable micro-profiling of shared libraries, shared library profiling (**-s**) must be turned on. To enable micro-profiling of kernel extensions, kernel extension profiling (**-e**) must be turned on.
- M PathList** Specifies the source path list. The PathList is a colon-separated list of paths that are searched for source files and .lst files that are required for micro-profiling. By default the source path list is the object search path list.
- p ProcessList** Enables process-level profiling of the process names specified in the ProcessList. ProcessList is a comma-separated list of process names. Process level profiling is enabled only if at least one of the profiling modes (**-u**, **-s**, **-k**, **-e**, or **-j**) is turned on.
- P {all | PIDList}** Enables process-level profiling of all processes encountered or for processes specified with PIDList, a comma-separated list of process IDs. Process level profiling is enabled only if at least one of the profiling modes (**-u**, **-s**, **-k**, **-e**, or **-j**) is turned on.
- r RootString** Specifies the RootString. tprof input and report files all have names in the form of RootString.suffix. If **-r** is not specified, RootString defaults to the program name specified with the **-x** flag.
- s** Enables shared library profiling.

- S PathList** Specifies the object search PathList, a colon-separated list of paths that are searched for executables, shared libraries, and kernel extensions. The default object search PathList is the environment path list (\$PATH).
- t** Enables thread level profiling. If -p or -P are not specified with the -t flag, -t is equivalent to -P all -t. Otherwise, it enables thread level reporting for the selected processes. Thread level profiling is enabled only if at least one of the profiling modes (-u, -s, -k, -e, -j) is enabled.
- T BufferSize** Specifies the trace BufferSize. This flag has meaning only in real-time or automated offline modes.
- u** Enables user mode profiling.
- v** Enables verbose mode.
- V** File Stores the verbose output in the specified File.
- x** Specifies the program to be executed by **tprof**. Data collection stops when the program completes or **trace** is manually stopped with either **trcoff** or **trcstop**. The -x flag must be the last flag in the list of flags specified in **tprof**. By default the program name is the RootString used for the filenames unless overridden by the -r flag.
- z** Enables compatibility mode with the previous version of tprof. By default CPU usage is only reported in percentages. When -z is used, **tprof** also reports ticks. This flag also adds the Address and Bytes columns in subroutine reports.

19.5.1 Information about measurement and sampling

In the AIX operating system, a decremter interrupt is issued whenever a timer expires on one of the processors. At least one timer per processor is active. The granularity of this timer is 10 milliseconds, and it is used to run a housekeeping kernel routine. However, if high-resolution timers are used, a decremter interrupt is issued each time a high-resolution timer expires. This increases the number of decremter interrupts per second.

The formula for the decremter interrupts is:

$$di/sec = (\#CPUs * 100) + et$$

where:

di/sec	Decremter interrupts per second
#CPU	Number of processors
et	Decremter interrupts issued by expired high-resolution timers

The **tprof** command uses this decremter interrupt to record the Process ID (PID) and the address of the instruction executing when the interrupt occurs. Using these data pairs (PID + Address), the **tprof** command can charge CPU time to processes, threads, subroutines, and source code lines. Refer to Example 19-11 for an example of the trace data used by **tprof**. Source code line profiling is called micro profiling.

The **tprof** command gathers a sample each time the decremter interrupt is issued. This may not be sufficiently accurate for short programs. However, the accuracy is sufficient for programs running several minutes or longer.

The **tprof** command uses the AIX trace facility. Only one user can use the AIX trace facility at a time. Thus, only one **tprof** command can be active in the system at a time.

The **tprof** command can run in the following four modes:

- ▶ Realtime or online: Using the **-x** and without **-A**, it generates the profile report immediately for the currently running system.
- ▶ Automated offline: Using **-A** and **-x**, it generates the symbolic list and the trace file. If **-c** is specified, the resulting files are **.csyms** and **.ctrc** files, which means cooked (processed) files.
- ▶ Manual offline: Using the **syms** and **trc** files, with **-F** flag (to force manual offline) and without **-x** or **-A**. The **syms** file can be generated by **tprof** or **gensyms** commands.
- ▶ Post-processing: Using the cooked **csyms** and **ctrc** files to generate profiling report.

Example 19-11 shows how **tprof** used the trace hook 234 to gather the necessary data.

Example 19-11 Trace data used by tprof

```
Mon Jun  4 15:16:22 2001
System: AIX datahost Node: 5
Machine: 000BC6AD4C00
Internet Address: 010301A4 1.3.1.164
The system contains 4 cpus, of which 4 were traced.
Buffering: Kernel Heap
This is from a 32-bit kernel.
Tracing all hooks.

/usr/bin/trace -a -C all
```

```
ID  PROCESS CPU PID  TID  ELAPSED  KERNEL  INTERRUPT
```

```

(... lines omitted ...)

100 wait    3   1290  1291  0.002359      DECREMENTER INTERRUPT iar=25C88 cpuid=03
234 wait    3   1290  1291  0.002364  clock:   iar=25C88 lr=26BF0
100 wait    3   1290  1291  0.002879      DECREMENTER INTERRUPT iar=25CAC cpuid=03
234 wait    3   1290  1291  0.002880  clock:   iar=25CAC lr=26BF0 [516 usec]
100 wait    0   516   517   0.004866      DECREMENTER INTERRUPT iar=26BA8 cpuid=00
234 wait    0   516   517   0.004868      clock:   iar=26BA8 lr=26BF0 [1988 usec]
100 wait    1   774   775   0.007352      DECREMENTER INTERRUPT iar=26BCC cpuid=01
234 wait    1   774   775   0.007355      clock:   iar=26BCC lr=26BF0 [2486 usec]
100 ksh     2   4778  34509 0.009856      DECREMENTER INTERRUPT iar=22C5C cpuid=02
234 ksh     2   4778  34509 0.009860      clock:   iar=22C5C lr=22BB0 [2505 usec]

(... lines omitted ...)

100 ksh     3   13360 42871 0.012359      DECREMENTER INTERRUPT iar=D01D4260 cpuid=03
234 ksh     3   13360 42871 0.012361  clock:   iar=D01D4260 lr=D01C722C [2501 usec]

(... lines omitted ...)

100 wait    0   516   517   0.014862      DECREMENTER INTERRUPT iar=25D54 cpuid=00
234 wait    0   516   517   0.014864      clock:   iar=25D54 lr=26BF0 [2502 usec]

(... lines omitted ...)

100 wait    1   774   775   0.017356      DECREMENTER INTERRUPT iar=25D40 cpuid=01
234 wait    1   774   775   0.017360      clock:   iar=25D40 lr=26BF0 [2495 usec]
100 wait    2   1032  1033  0.019861      DECREMENTER INTERRUPT iar=25D30 cpuid=02
234 wait    2   1032  1033  0.019865      clock:   iar=25D30 lr=26BF0 [2505 usec]
100 wait    3   1290  1291  0.022355      DECREMENTER INTERRUPT iar=25CAC cpuid=03
234 wait    3   1290  1291  0.022358  clock:   iar=25CAC lr=26BF0 [2492 usec]
100 wait    0   516   517   0.024857      DECREMENTER INTERRUPT iar=25E50 cpuid=00
234 wait    0   516   517   0.024858      clock:   iar=25E50 lr=26BF0 [2500 usec]

(... lines omitted ...)

```

This example shows trace hook 100 for the decremter interrupt. However, **tprof** only uses the trace hook 234. Focusing on the decremter interrupts and following trace hook 234 for CPU number 3 shows that the second interrupt at 0.002879 was not issued by the normal 10 millisecond timer. A high resolution timer was causing this decremter interrupt. The interrupts for the 10 ms timer for this processor were issued at 0.002359, 0.012359, and 0.022355.

19.5.2 Examples for tprof

This section shows some examples for using **tprof**.

The tprof report

The **tprof** report is divided into sections and subsections. Some sections are created depending on the flags specified, such as kernel routines (-k), kernel extension (-e), shared libraries (-s), Java profiling enabled (-j), user mode profiling enabled (-u). The report file has the suffix `.prof` from the root string. The root string is by default the command name that is being profiled. The report in the `.prof` file contains:

- ▶ Summary report section
 - CPU usage summary by process name
 - CPU usage summary by threads (tid)
- ▶ Global profile section, which pertains to the execution of all processes on system. Parts of this section are activated by specific flags as indicated in the parentheses.
 - CPU usage of user mode routines (-u)
 - CPU usage of kernel routines (-k)
 - CPU usage summary for kernel extensions (-e)
 - CPU usage of each kernel extension's subroutines. (-e)
 - CPU usage summary for shared libraries (-s)
 - CPU usage of each shared library's subroutines. (-s)
 - CPU usage of each Java class. (-j)
 - CPU usage of each Java methods of each Java class. (-j)
- ▶ Process and thread level profile sections. There is one section for each process or thread.
 - CPU usage of user mode routines for this process/thread
 - CPU usage of kernel routines for this process/thread.
 - CPU usage summary for kernel extensions for this process/thread.
 - CPU usage of each kernel extension's subroutines for this process/thread.
 - CPU usage summary for shared libraries for this process/thread.
 - CPU usage of each shared library's subroutines for this process/thread.
 - CPU usage of each Java class for this process/thread.
 - CPU usage of Java methods of each Java class for this process/thread.

The summary report section is always present in the `RootString.prof` report file. Based on the profiling flags the various subsections of the global profile section can be turned on and off.

The process and thread level profile sections are created for processes and threads selected with the `-p`, `-P`, and `-t` flags. The subsections present within each

of the per-process or per-thread sections are identical to the subsections present in the global section, they are selected using the profiling flags (-u,-s,-k,-e,-j).

Optionally, when **tprof** is called with the -C flag, it also generates per-CPU profiling reports. The generated **tprof** reports have the same structure and are named using the convention: RootString.prof-*cpuid*.

Global profiling with tprof

In this example, we use **tprof** to perform global profiling over 20 seconds of time. The **tprof** command is invoked with sleep command as a timer. We show this for online, offline, and post-processing modes.

Online profiling

Example 19-12 shows the use of the **tprof** command to profile the system by using **tprof -kes -x sleep 20** to create the summary report for the whole system. This report includes profile information for kernel (-k), shared library (-s), and kernel extensions (-e). The -x sleep 20 is used to control the sample time of the **tprof** command, 20 seconds in this case.

Example 19-12 Running tprof to profile the system

```
# tprof -kes -x sleep 20
Wed Apr 9 14:33:24 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Starting Command sleep 20
stopping trace collection.
Generating sleep.prof.
```

The result is the file sleep.prof in the current directory. The content of sleep.prof is shown in Example 19-13.

Example 19-13 Result of tprof in sleep.prof

Process	Freq	Total	Kernel	User	Shared	Other
=====	====	=====	=====	====	=====	=====
wait	4	49.68	49.68	0.00	0.00	0.00
cwhet_a	1	25.00	0.00	25.00	0.00	0.00
cwhet_c	1	25.00	0.00	25.00	0.00	0.00
/usr/bin/topas	1	0.22	0.21	0.00	0.01	0.00
/usr/bin/sh	3	0.03	0.02	0.01	0.00	0.00
5	2	0.02	0.01	0.00	0.01	0.00
/usr/bin/trcstop	1	0.01	0.01	0.00	0.00	0.00
db2dasrmm	1	0.01	0.00	0.00	0.01	0.00
/usr/bin/tprof	1	0.01	0.00	0.00	0.01	0.00
gil	1	0.01	0.01	0.00	0.00	0.00
=====	====	=====	=====	====	=====	=====
Total	16	100.00	49.94	50.02	0.05	0.00

Process	PID	TID	Total	Kernel	User	Shared	Other
=====	===	===	=====	=====	=====	=====	=====
cwhet_a	21770	53081	25.00	0.00	25.00	0.00	0.00
cwhet_c	8826	73389	25.00	0.00	25.00	0.00	0.00
wait	516	517	24.95	24.95	0.00	0.00	0.00
wait	1290	1291	24.54	24.54	0.00	0.00	0.00
/usr/bin/topas	23970	8407	0.22	0.21	0.00	0.01	0.00
wait	1032	1033	0.17	0.17	0.00	0.00	0.00
wait	774	775	0.02	0.02	0.00	0.00	0.00
5	0	68403	0.01	0.00	0.00	0.01	0.00
/usr/bin/sh	34774	50721	0.01	0.01	0.00	0.00	0.00
/usr/bin/sh	25640	69131	0.01	0.00	0.01	0.00	0.00
/usr/bin/sh	25638	69129	0.01	0.01	0.00	0.00	0.00
db2dasrrm	8516	10879	0.01	0.00	0.00	0.01	0.00
gil	5676	6451	0.01	0.01	0.00	0.00	0.00
/usr/bin/tprof	18046	76375	0.01	0.00	0.00	0.01	0.00
/usr/bin/trcstop	34776	50723	0.01	0.01	0.00	0.00	0.00
5	0	17663	0.01	0.01	0.00	0.00	0.00
=====	===	===	=====	=====	=====	=====	=====
Total			100.00	49.94	50.02	0.05	0.00

Total Samples = 8775 Total Elapsed Time = 21.94s

Total % For All Processes (KERNEL) = 49.93

Subroutine	%	Source
=====	=====	=====
.waitproc_find_run_queue	35.54	rne1/proc/dispatch.c
.waitproc	12.59	rne1/proc/dispatch.c
.h_cede_native	0.76	ne1/ml/POWER/stubs.c
.h_cede	0.72	hcalls.s
.memset_overlay	0.09	low.s
.upd_vminfo	0.08	rne1/vmm/vmgetinfo.c
h_get_term_char_end_point	0.05	hcalls.s
.trchook	0.02	trchka.s
pcs_glue	0.02	vmvcs.s
.unlock_enable_mem	0.01	low.s
.nlcLookup	0.01	bos/kernel/lfs/nlc.c
.pfslowtimo	0.01	kernel/uipc/domain.c
.vnop_rdwr	0.01	s/kernel/lfs/vnops.c

Total % For All Processes (KEX) = 0.01

Kernel Ext	%
=====	=====
/usr/lib/drivers/ldterm[ldterm32]	0.01

Profile: /usr/lib/drivers/ldterm[ldterm32]

Total % For All Processes (/usr/lib/drivers/ldterm[ldterm32]) = 0.01

Subroutine	%	Source
===== .ldtty_putc	===== 0.01	===== ers/ldterm[ldterm32]

Total % For All Processes (SH-LIBS) = 0.05

Shared Object	%
===== /usr/lib/libtrace.a[shr.o]	===== 0.01
/usr/WebSphere/AppServer/java/jre/bin/classic/libjvm.a/	0.01
/usr/lib/libxcurses.a[shr4.o]	0.01
/usr/lib/libpthreads.a[shr_xpg5.o]	0.01

Profile: /usr/lib/libtrace.a[shr.o]

Total % For All Processes (/usr/lib/libtrace.a[shr.o]) = 0.01

Subroutine	%	Source
===== .eread	===== 0.01	===== btrace/trcgetevent.c

Profile: /usr/WebSphere/AppServer/java/jre/bin/classic/libjvm.a/

Total % For All Processes
(/usr/WebSphere/AppServer/java/jre/bin/classic/libjvm.a/) = 0.01
..... (more lines).....

Offline profiling

To perform profiling similar to Example 19-12 on page 330 but use offline processing, **-A** is specified for **tprof** to run in automated offline mode. Three files are generated with the **trc**, **prof**, and **syms** extensions, as shown in Example 19-14.

Example 19-14 Gather the data to run tprof in offline mode

```
# tprof -A -x sleep 20
Starting Command sleep 20
stopping trace collection.
Wed Apr 9 15:06:10 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Generating sleep.trc
Generating sleep.prof
Generating sleep.syms
# ls -l
total 19260
-rw-r--r-- 1 root system 1889 Apr 9 15:06 sleep.prof
-rw-r--r-- 1 root system 9542925 Apr 9 15:06 sleep.syms
```

```
-rw-rw-rw- 1 root system 315212 Apr 9 15:06 sleep.trc
```

The generated files can be reprocessed in manual offline to generate the report later. This process is shown in Example 19-15.

Example 19-15 Run tprof in offline mode

```
# tprof -r sleep -kes
Wed Apr 9 15:06:10 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Generating sleep.prof
```

Now we have the necessary sleep.prof, which is shown in Example 19-16.

Example 19-16 Offline mode sleep.prof

Process			Freq	Total	Kernel	User	Shared	Other
=====			====	=====	=====	====	=====	=====
cwhet_d			1	25.00	0.00	25.00	0.00	0.00
cwhet_a			1	24.99	0.00	24.99	0.00	0.00
cwhet_b			1	24.98	0.00	24.98	0.00	0.00
cwhet_c			1	24.93	0.00	24.93	0.00	0.00
/usr/bin/topas			1	0.07	0.05	0.00	0.02	0.00
.....(more lines)....								
/usr/bin/sh	25200	34045	0.01	0.01	0.00	0.00	0.00	0.00
wlmsched	5934	7225	0.01	0.01	0.00	0.00	0.00	0.00
=====	===	===	=====	=====	=====	=====	=====	=====
Total			100.00	0.08	99.90	0.02	0.00	

Total Samples = 8756 Total Elapsed Time = 21.89s

Total % For All Processes (KERNEL) = 0.07

Subroutine	%	Source
=====	=====	=====
.memset_overlay	0.03	low.s
pcs_glue	0.01	vmvcs.s
.lock_done	0.01	low.s
.... (more lines)....		

Post-processing with tprof

Example 19-17 shows that if the -c flag is also specified, then RootString.prof, RootString.csyms, and RootString.ctrc are generated for post-processing mode.

Example 19-17 tprof example using -c flag for post-processing

```
# tprof -c -A -x sleep 20
Starting Command sleep 20
```

```

stopping trace collection.
Wed Apr 9 15:14:57 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Generating sleep.ctrac
Generating sleep.prof
Generating sleep.csyms
# tprof -r sleep
Wed Apr 9 15:14:57 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Generating sleep.prof

```

Example 19-18 shows the %CPU accumulated for the kernel routines from the file sleep.prof.

Example 19-18 Accumulated %CPU in kernel routines

Total % For All Processes (KERNEL) = 49.93

Subroutine	%	Source
=====	=====	=====
.waitproc_find_run_queue	35.54	rn1/proc/dispatch.c
.waitproc	12.59	rn1/proc/dispatch.c
.h_cede_native	0.76	ne1/m1/POWER/stubs.c
.h_cede	0.72	hcalls.s
.memset_overlay	0.09	low.s
.upd_vminfo	0.08	rn1/vmm/vmgetinfo.c
h_get_term_char_end_point	0.05	hcalls.s
.trchhook	0.02	trchka.s
pcs_glue	0.02	vmvcs.s
.unlock_enable_mem	0.01	low.s
.nlcLookup	0.01	bos/kernel/lfs/nlc.c
.pfslowtimo	0.01	kernel/uipc/domain.c
.vnop_rdwr	0.01	s/kernel/lfs/vnops.c

Example 19-19 shows the %CPU accumulated for the kernel extensions.

Example 19-19 Accumulated %CPU in kernel extensions

Total % For All Processes (KEX) = 0.01

Kernel Ext	%
=====	=====
/usr/lib/drivers/ldterm[ldterm32]	0.01

Profile: /usr/lib/drivers/ldterm[ldterm32]

Total % For All Processes (/usr/lib/drivers/ldterm[ldterm32]) = 0.01

Subroutine	%	Source
------------	---	--------

```
=====  
.ldtty_putc
```

```
=====  
0.01 ers/ldterm[ldterm32]
```

Single and multiple process level profiling

Example 19-20 shows the command used for single process profiling and extracting the user mode profile using the `-u` flag.

Example 19-20 Example of single process level profiling

```
#tprof -u -p workload -x workload
```

Example 19-21 shows the profiling for the `startall.sh` shell command, which invokes the `send` and `receive` commands. The output file `startall.prof` contains two process level profile sections: `send` and `receive`. Both shared library (`-s`) and kernel extension (`-e`) profiles are enabled.

Example 19-21 Example of multiple process level profiling

```
# cat startall.sh  
#!/bin/sh  
send  
receive  
exit 0  
  
# tprof -se -p send,receive -x startall.sh
```

Profiling an application

The `tprof` command can be used to profile any application. No recompiling or relinking of the application is necessary. To take the full advantage of `tprof` microprofiling capability, it is best to provide both `.lst` and source files. A report similar to the summary report is generated. If `-m` is specified, `tprof` generates micro-profiling reports with the name `RootString.source.mprof`, where `source` is the source file name. The micro-profiling report contains a hot line profile section, which has all of the line numbers from the source file executed by profiling samples sorted by CPU usage. In addition, it contains a source line profile section for each of the functions in the source file that have CPU usage. This section contains the source line number, CPU usage, and source code.

First we generate the object file and the listing of the application using a modified `cwhet` source provided in “`cwhet.c`” on page 968. The command that is used is:

```
xlc -qarch=auto -qtune=auto -o cwhet_100K -g -qsource -qlist -lm -03  
-qstrict cwhet_100K.c  
xlc -qarch=auto -qtune=auto -o cwhet_100K -lm -03 -qstrict cwhet_100K.c
```

Example 19-22 shows the commands used to run **tprof** to profile an application with **-m** flag. As shown, it generates the **cwhet_100K.prof** and **cwhet_100K.cwhet_100K.c.mprof**.

Example 19-22 Microprofiling of an application

```
# tprof -m ./cwhet_100K -u -x "cwhet_100K >/dev/null"
Wed Apr 9 16:30:42 2003
System: AIX 5.2 Node: lpar05 Machine: 0021768A4C00
Starting Command cwhet_100K >/dev/null
stopping trace collection.
Generating cwhet_100K.prof
Generating cwhet_100K.cwhet_100K.c.mprof
# ls -ltr
-rw-r--r-- 1 root system 76006 Apr 9 16:29 cwhet_100K.lst
-rwxr-xr-x 1 root system 48118 Apr 9 16:29 cwhet_100K
-rw-r--r-- 1 root system 3974 Apr 9 16:30 cwhet_100K.prof
-rw-r--r-- 1 root system 135921 Apr 9 16:30 cwhet_100K.cwhet_100K.c.mprof
```

Example 19-23 shows the resulting microprofiling output in **cwhet_100K.cwhet_100K.c.mprof**.

Example 19-23 Output of microprofiling an application

```
Hot Line Profile of cwhet_100K.c

Line    % PID
156  6.04 ALL
153    1.79 ALL
0      1.77 ALL
166    1.32 ALL
114    0.89 ALL
160    0.70 ALL
183    0.26 ALL
181    0.19 ALL
184    0.11 ALL
182    0.09 ALL
----- ( lines omitted) -----

Line    % Source
0      1.77 -
- 0000BC lwz      8082001C 1    L4A    gr4=.t(gr2,0)
- 0000C0 lfd      C8230018 1    LFL    fp1=(*)double(gr3,24)
- 0000C4 lwz      80A20018 1    L4A    gr5=.t2(gr2,0)
..... ( lines omitted).....
- 000CD0 stw      90980000 1    ST4A   i(gr24,0)=gr4
154    - -
-
- 000CB0 add      7C13A214 1    A      gr0=gr19,gr20
CL.265:
```

```

155 - -
    - 000CB8 add      7C130214  1   A      gr0=gr19,gr0
156 6.04 -
    6.04 000CB4 ori    62740000  1   LR     gr20=gr19
157 - -
    - 000CBC subf    7C140050  1   S      gr0=gr0,gr20
    - 000CC0 subf    7E740050  1   S      gr19=gr0,gr20
    - 000CCC stw     92720000  1   ST4A   k(gr18,0)=gr19
158 - -
..... ( lines omitted ).....
193 -   y = t * (x + y);
    - 000078 fadd    FC40102A  4   AFL     fp2=fp0, fp2, fcr
    - 00007C fmul    FC2100B2  4   MFL     fp1=fp1, fp2, fcr
194 -   *z = (x + y) / t2;
    - 000064 lwz    80620018  1   L4A     gr3=.t2(gr2,0)
.....( lines omitted ) .....
Total % for .mod9 = 0.64

```

Line	% Source
200	- e1[j] = e1[k];
	- 000000 lwz 80620004 1 L4A gr3=.k(gr2,0)
	- 000004 lwz 80820008 1 L4A gr4=.j(gr2,0)
	- 000008 lwz 80C2000C 1 L4A gr6=.e1(gr2,0)
	- 000010 lwz 80040000 1 L4A gr0=j(gr4,0)
	- 000014 lwz 80630000 1 L4A gr3=k(gr3,0)

The output of the `cwhet_100K.lst` is shown in Example 19-24.

Example 19-24 Output of microprofiling an application

```

----- ( lines omitted ) .....|
149 | /**** Module 10: Integer Arithmetic ****/
150 |
151 |     j = 2;
152 |     k = 3;
153 |     for (i = 1; i <= n10; i++) {
154 |         j = j + k;
155 |         k = j + k;
156 |         j = k - j;
157 |         k = k - j - j;
158 |     }
159 | #ifdef POUT
160 |     pout(n10, j, k, x1, x2, x3, x4);
----- ( lines omitted ) .....|
188 | /**** Module 8 Routine ****/
189 | mod8(x, y, z)
190 | double x, y, *z;
191 | {

```

```

192 |     x = t * (x + y);
193 |     y = t * (x + y);
194 |     *z = (x + y) / t2;
195 | }
196 |
..... ( lines omitted ..... )

```

```

83|                                     CL.265:
154| 000CB0 add      7C13A214 1   A   gr0=gr19,gr20
156| 000CB4 ori      62740000 1   LR   gr20=gr19
155| 000CB8 add      7C130214 1   A   gr0=gr19,gr0
157| 000CBC subf    7C140050 1   S   gr0=gr0,gr20
157| 000CC0 subf    7E740050 1   S   gr19=gr0,gr20
0| 000CC4 bc      4320FFEC 0   BCT  ctr=CL.265,

```

The output of the `cwhet_100K.prof` is shown in Example 19-25. Notice the profiling at subroutine and function level.

Example 19-25 Output of microprofiling an application

Process	Freq	Total	Kernel	User	Shared	Other
=====	====	=====	=====	====	=====	=====
wait	4	77.75	77.75	0.00	0.00	0.00
./cwhet_100K	1	21.94	0.00	21.94	0.00	0.00
/usr/bin/topas	1	0.09	0.09	0.00	0.00	0.00
..... (lines omitted).....						
Profile: ./cwhet_100K						

Total % For All Processes (./cwhet_100K) = 21.94

Subroutine	%	Source
=====	=====	=====
.main	13.33	cwhet_100K.c
.log	2.18	r/ccs/lib/libm/log.c
.exp	1.48	r/ccs/lib/libm/exp.c
.mod3	1.21	cwhet_100K.c
.cos	1.12	r/ccs/lib/libm/cos.c
.mod8	0.97	cwhet_100K.c
.atan	0.70	/ccs/lib/libm/atan.c
.mod9	0.64	cwhet_100K.c
.sin	0.31	r/ccs/lib/libm/sin.c

This example shows the routines and source codes where CPU is consumed, the outputs sort the routines, and source line by %CPU usage. The hot lines and the source code profiles can be used to improve the performance of the application.

Reporting CPU usage by ticks

The `tprof` command by default gives CPU usage in percentages, and flag `-z` reports CPU usage in ticks. Example 19-26 shows the use of the `-z` flag to display the CPU ticks for source lines sorted by time ticks instead of CPU percentage that was shown in Example 19-22 on page 336.

Example 19-26 tprof reports CPU in ticks

```
# tprof -z -m ./cwhet_100K -u -x "cwhet_100K >/dev/null"
# Hot Line Profile of cwhet_100K.c
# more cwhet_100K.cwhet_100K.c.mprof
..... ( lines omitted) .....
  Line  Ticks  PID
    156   379  ALL
     0    128  ALL
    153   110  ALL
    166    82  ALL
    114    58  ALL
..... ( lines omitted) .....
```

```
#more cwhet_100K.prof
Process                               FREQ  Total  Kernel  User  Shared  Other
=====                               =====
wait                                   3    3378   3378     0     0     0
cwhet_10M                              2    1601     0    1601     0     0
./cwhet_100K                           1    1406     0    1406     0     0
/usr/bin/topas                          2     10     7     0     3
..... ( lines omitted) .....
```

Profiling of Java applications

The `-j` flag turns on Java classes and methods of profiling. Example 19-27 shows a profiling report with a new column named Java.

Example 19-27 Example shows profiling a Java application

```
# tprof -j -x "cd /JavaTools; /usr/java131/jre/bin/java -Xms1024m -Xmx1024m
VBDMemBlot 5000"
# cat cd.prof
Process                               Freq  Total  Kernel  User  Shared  Other  Java
=====                               =====
wait                                   4  74.83  74.83   0.00   0.00   0.00   0.00
/usr/java131/jre/bin/java              1  24.61   0.03   0.00   0.12  24.46   0.00
/usr/bin/tprof                         1   0.18   0.02   0.00   0.16   0.00   0.00
.....( lines omitted) .....
```

Total	61	100.00	75.18	0.02	0.34	24.46	0.00
-------	----	--------	-------	------	------	-------	------

Process	PID	TID	Total	Kernel	User	Shared	Other	Java
131/jre/bin/java	23396	77831	24.61	0.03	0.00	0.12	24.46	0.00

```

wait          1290    1291  24.47  24.47  0.00  0.00  0.00  0.00
wait          516     517  23.96  23.96  0.00  0.00  0.00  0.00
..... ( line omitted) .....

```

Using tprof to detect a resource bottleneck

In this example the main user application of a company, **sem_app**, is used on smaller Uni Processor (UP) systems with up to 100 users per system. However, maintaining all of these systems is no longer possible and the decision was made to replace all 20 UP server systems with one SMP server. During the switch from the old UP server systems to the new SMP server, which is done on a step-by-step basis, the performance on the new SMP server goes down as more users start to use it. With half of the users moved to the new SMP server the performance of the user application is very slow.

The first step of the solution is to run **vmstat** and **iostat** on the new SMP server system to detect possible CPU or I/O bottlenecks. The **iostat** command shows no bottleneck with disk I/O. In fact, most of the disks are idle. Only the CPU usage with more than 80 percent reported in system (kernel) mode and less than 10 percent in user mode, with a few percent CPU left in idle, gives a first indication of the problem source. The system spends too much CPU time in kernel subroutines. The output of the **vmstat** command in this situation is shown in Example 19-28.

Example 19-28 Output of the vmstat command on CPU bound system

```

# vmstat 1 10
kthr    memory          page          faults          cpu
-----
 r  b   avm   fre  re  pi  po  fr   sr  cy  in  sy  cs us  sy id wa
517  0 73041 67983   0   0   0   0   0   0 462 1728 76642 11 86  3  0
418  0 73043 67981   0   0   0   0   0   0 450 1377 79056  9 87  4  0
962  0 73045 67979   0   0   0   0   0   0 446 1399 91215  8 88  4  0
198  0 73047 67977   0   0   0   0   0   0 441 1493 78038 13 82  5  0

```

The CPU time spent in system (kernel) mode is more than 80 percent. The number of threads on the run queue is between 198 and 962. The number of context switches is very high. However, with this number of threads on the run queue it is not unusual to have some context switches.

The **tprof** command is used to determine the reason why the CPU time spent in system mode gets this high and to determine what causes this behavior.

The **tprof -z -kes -x sleep 5** command is used to collect the process summary for all processes. The data collected by **tprof** is shown in Example 19-29 on page 341.

Example 19-29 Output of tprof on a CPU bound system

Process	PID	TID	Total	Kernel	User	Shared	Other	
===== tprof	547514	563769	91	19	58	14	0	0
wait	516	517	41	41	0	0	0	0
wait	774	775	37	37	0	0	0	0
wait	1290	1291	36	36	0	0	0	0
wait	1032	1033	32	32	0	0	0	0
sem_app	430374	446371	7	3	4	0	0	0
swapper	0	3	6	6	0	0	0	0
sem_app	431406	447403	6	5	1	0	0	0
sem_app	116366	132107	5	5	0	0	0	0
sem_app	132106	148103	5	5	0	0	0	0
sem_app	157390	173131	5	5	0	0	0	0
sem_app	183966	199707	5	2	3	0	0	0

(... lines omitted ...)

sem_app	544928	560669	1	1	0	0	0	0
sem_app	546476	562217	1	1	0	0	0	0
PID.547774	547774	562481	1	1	0	0	0	0
sleep	547774	562481	1	1	0	0	0	0
===== Total	===== Total	===== Total	===== 2071	===== 1944	===== 112	===== 15	===== 0	===== 0

Process	FREQ	Total	Kernel	User	Shared	Other
===== sem_app	1142	1798	1744	54	0	0
wait	4	146	146	0	0	0
tprof	1	91	19	58	14	0
trclogio	1	22	22	0	0	0
swapper	1	6	6	0	0	0
gil	2	3	3	0	0	0
aixterm	2	2	1	0	1	0
wlmsched	1	1	1	0	0	0
PID.547774	1	1	1	0	0	0
sleep	1	1	1	0	0	0
===== Total	===== 1156	===== 2071	===== 1944	===== 112	===== 15	===== 0

Total System Ticks: 2071 (used to calculate function level CPU)

Total Ticks For All Processes (KERNEL) = 1943

Subroutine	Ticks	%	Source	Address	Bytes
.slock_ppc	690	33.3	simple_lock.c	1df990	354
.e_block_thread	505	24.4	sleep2.c	425d8	548
.e_assert_wait	190	9.2	sleep2.c	42eb8	18c
.sunlock_ppc	124	6.0	simple_lock.c	1df898	f8
.waitproc_find_run_queue	91	4.4	dispatch.c	25c88	210
.kwakeup	86	4.2	sleep.c	438f0	1c0
.waitproc	55	2.7	dispatch.c	26b54	12c
.compare_and_swap	31	1.5	low.s	a4c0	100
.disable_lock	30	1.4	low.s	9004	2fc
.atomic	28	1.4	ipc/sem.c	465e64	8bc
.my_csa	18	0.9	low.s	b408	20
.exbcopy_ppc	16	0.8	misc_ppc.s	1d2dc0	bc
.e_sleep_thread	14	0.7	sleep2.c	43044	118
.simple_unlock_mem	13	0.6	low.s	9918	1e8
.simple_lock	12	0.6	low.s	9500	400
.uiocopyout_ppc	8	0.4	copyx_ppc.s	1d4720	2a0

(... lines omitted ...)

There are 1142 user processes **sem_app** active on the system. These processes account for the most time spent in system mode, which is 1744 time ticks out of 1944 time ticks. The most-used kernel subroutines are from the systems lock management functions. There is a subroutine named **.atomic** out of the source file **ipc/sem.c**.

The next steps are to find out whether the application is using semaphores, and whether the application is using only a few, causing all 1142 processes to fight for these semaphores.

To show the relationship between the number of users running the application **sem_app** and the CPU usage, a monitoring script that runs every five minutes counts the number of user processes named **sem_app**, runs the **sar** command for a short time, and stores this data into a file installed on the system. To start clean, the system is rebooted.

The script used to collect the data is shown in Example 19-30.

Example 19-30 Script to monitor CPU bound system

```
#!/usr/bin/ksh

OUTFILE=/var/adm/ras/server.load
TIME=300

while true
do
```

```

date >>$OUTFILE
UPROC=`ps -ef|grep sem_app|wc -l`
echo "$UPROC sem_app processes in process table" >>$OUTFILE
sar -quw 1 3 >>$OUTFILE
echo "===== " >>$OUTFILE
sleep $TIME
done

```

Example 19-31 is an extract of the data collected by the monitoring script.

Example 19-31 Output of the monitoring script

(... lines omitted ...)

Mon May 21 7:15:07 CDT 2001

8 sem_app processes in process table

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

07:15:08	runq-sz	%runocc	swpq-sz	%swpocc
	%usr	%sys	%wio	%idle
	cswch/s			

07:15:09	9.0	100		
	53	4	0	43
	5672			

07:15:10	3.0	100		
	57	4	0	39
	5631			

07:15:11	9.0	100		
	61	2	0	37
	5642			

Average	7.0	94		
Average	57	3	0	40
Average	5648			

(... lines omitted ...)

Mon May 21 7:35:25 CDT 2001

17 sem_app processes in process table

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

07:35:29	runq-sz	%runocc	swpq-sz	%swpocc
----------	---------	---------	---------	---------

	%usr cswch/s	%sys	%wio	%idle
07:35:30	17.0 49 11052	100 9	0	42
07:35:31	17.0 49 11047	100 7	0	44
07:35:32	17.0 48 11090	100 7	0	45
Average	17.0	94		
Average	49	8	0	44
Average	11063			

(... lines omitted ...)

Mon May 21 7:55:59 CDT 2001
34 sem_app processes in process table

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

	runq-sz %usr cswch/s	%runocc %sys	swpq-sz %wio	%swpocc %idle
07:56:02	runq-sz 9.0 54 19753	%runocc 100 15	swpq-sz 0	%swpocc 30
07:56:03	22.0 54 19761	100 14	0	32
07:56:04	32.0 56 19636	100 15	0	29
07:56:05	21.0	94		
Average	55	15	0	30
Average	19717			

(... lines omitted ...)

Mon May 21 8:15:45 CDT 2001

67 sem_app processes in process table

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

	runq-sz	%runocc	swpq-sz	%swpocc
	%usr	%sys	%wio	%idle
	cswch/s			

08:15:50	31.0	100		
	49	40	0	11
	45493			

08:15:51	89.0	100		
	52	34	0	14
	45075			

08:15:52	80.0	100		
	54	36	0	10
	46057			

Average	66.7	94		
Average	52	37	0	12
Average	45540			

(... lines omitted ...)

Mon May 21 8:30:13 CDT 2001

123 sem_app processes in process table

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

	runq-sz	%runocc	swpq-sz	%swpocc
	%usr	%sys	%wio	%idle
	cswch/s			

08:30:17	115.0	100		
	53	44	0	3
	53857			

08:30:18	86.0	100		
	55	41	0	4
	53593			

08:30:19	122.0	100		
	50	45	0	5
	54206			

```

Average 107.7 94
Average 52 43 0 4
Average 53886

```

(... lines omitted ...)

```

Mon May 21 8:45:21 CDT 2001
263 sem_app processes in process table

```

```

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

```

```

08:45:24 runq-sz %runocc swpq-sz %swpocc
        %usr  %sys  %wio  %idle
        cswch/s

```

```

08:45:25 172.0 100
          45  51  0  3
          63418

```

```

08:45:26 249.0 100
          46  50  0  4
          63738

```

```

08:45:27 119.0 100
          45  52  0  3
          64341

```

```

Average 180.0 93
Average 46 51 0 3
Average 63832

```

(... lines omitted ...)

```

Mon May 21 9:00:23 CDT 2001
499 sem_app processes in process table

```

```

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

```

```

09:00:27 runq-sz %runocc swpq-sz %swpocc
        %usr  %sys  %wio  %idle
        cswch/s

```

```

09:00:28 307.0 100
          35  64  0  1
          68880

```



```

09:00:29  262.0    100
           35     64     0     1
           66714

09:00:30  278.0    100
           31     67     0     1
           67414

Average   282.3    93
Average  34    65    0    1
Average   67664

```

(... lines omitted ...)

```

Mon May 21  9:26:33 CDT 2001
976 sem_app processes in process table

```

```

AIX wlmhost 1 5 000BC6AD4C00 05/21/01

```

```

09:26:37 runq-sz %runocc swpq-sz %swpocc
           %usr   %sys   %wio  %idle
           cswch/s

09:26:38  347.0    100
           9     87     0     5
           76772

09:26:39  436.0    100
           7     89     0     4
           74820

09:26:40  635.0    100
           10    87     0     3
           76949

Average   472.7    92
Average  8    88    0    4
Average   76194

```

(... lines omitted ...)

This output shows that CPU time spent in system mode increases as more **sem_app** user applications are running. At about 500 user processes the CPU time spent in system mode is 65 percent and the time spent in user mode is down to 34 percent. Even more dramatic are the values with close to 1000 user

processes running on the system. Only 8 percent CPU time is spent in user mode, but 88 percent CPU time is spent in system mode.

The application supplier is contacted and this turned out to be true. The application does a `fork()` and the parent and child processes are using a semaphore to synchronize with each other. However, the key used for the `semget()` subroutine is a hard-coded positive number that causes all `sem_app` programs to access the same systemwide semaphore. A change in the program source to use the `IPC_PRIVATE` key solved the problem.

The nice and renice commands

The **nice** command enables a user to adjust the dispatching priority of a command. Non-root authorized users can only degrade the priority of their own commands. A user with root authority can improve the priority of a command as well. A process, by default, has a nice value of 20. Numerically increasing this value results in degraded priority of the threads in this process. Therefore, to request lower priority you would increase the nice value from anything between 21 and 39 by specifying an increment value of between 0 (zero) and 19. To decrease the nice value anywhere downward of 20, the increment value would be -1 (one) to -20.

The **renice** command is used to change the nice value of one or more processes that are running on a system. The **renice** command can also change the nice values of a specific process group.

The **nice** and **renice** commands reside in /usr/bin and are part of the bos.rte.control fileset, which is installed by default from the AIX base installation media.

20.1 nice

The syntax of the **nice** command is:

```
nice [ -Increment | -n Increment ] Command [ Argument ... ]
```

Flags

-Increment Moves a command's priority up or down. You can specify a positive or negative number. Positive increment values degrade priority, and negative increment values improve priority. Only users with root authority can specify a negative increment. If you specify an increment value that would cause the nice value to exceed the range of 0 (zero) to 39, the nice value is set to the value of the limit that was exceeded.

-n Increment This flag is equivalent to the **-Increment** flag.

The **-n** flag and the **-** flag are synonymous.

Parameters

Increment A decimal integer in the range of -1 to -20 is used to improve the priority of a command. A decimal integer in the range of 0 (zero) to 19 is used to degrade the priority of a command.

Command This is the actual command that will run with the modified nice value.

Argument ... This is the argument of the command that will be running with the modified nice value.

20.1.1 Information about measurement and sampling

The **nice** command changes the value of the priority of a thread by changing the nice value of its process, which is used to determine the overall priority of that thread. A child process will inherit the nice value from the parent process. The nice value can be viewed using the **-l** flag with the **ps** command. See Chapter 8, "The **ps** command" on page 127. The nice values are displayed under the column heading **NI**. Threads with numerically lower nice values (higher priority) tend to run ahead of those with higher values (lower priority). Only users with root authority can change the priority of a command to an improved value (lower value of nice). Any attempt by any other user does not change the nice value.

The priority of a thread is not only determined by the nice value, but also by the **schedo** parameters if they have been set. Specifically, the **sched_D** option with the **schedo** command, the decay value of a thread, and the **sched_R** penalty factor of a thread can affect the priority of a thread. Refer to 10.1, “**schedo**” on page 166 for more information about the **schedo** command.

Tip: 1.2.2, “Processes and threads” on page 6 explains how process priorities are calculated on AIX.

Background processes that are run from the korn shell (**ksh**) will automatically have *four* added to their nice value. If, for example, a thread were to be run with its default nice value in background mode, then the nice value would actually be 24.

When a thread is running, the default scheduler policy is **SCHED_OTHER**. This means that the more CPU time a process gets the more it gets penalized. As the CPU usage increases for this thread, the priority value increases until it reaches a maximum value. The thread, therefore, becomes less favored to run again as CPU usage increases. See 10.1, “**schedo**” on page 166 for definitions of the scheduling types.

20.2 Examples for nice

The nice value for a user process that is started in the foreground is (by default) 20, as can be seen in Example 20-1.

Example 20-1 Default nice value

```
# ps -l
  F S UID      PID PPID  C PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
  240001 A    0 18892 14224  2  61 20 af15 1012      pts/3  0:00 ksh
  200001 A    0 20646 18892  4  62 20 a714  444      pts/3  0:00 ps
```

The priority of a process is listed in the **PRI** column of the **ps** output. As shown in Example 20-1 the priority of the **ps** command is calculated to be 62. Because it has used some CPU time, the priority has been degraded by two. At the instance of launch the process’ priority was 60. As with the nice value the higher the **PRI** value of a thread process the lower the priority.

If the process is launched in the background, the nice value is 24 by default, as demonstrated in Example 20-2.

Example 20-2 Default nice value, background

```
# ps -l &
  F S UID      PID PPID  C PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
```

240001	A	0	18892	14224	1	60	20	af15	1012	7030ae44	pts/3	0:01	ksh
200001	A	0	23462	18892	4	70	24	9f13	448		pts/3	0:00	ps

As stated before, if a process is started in the background, four is added to the *nice* value. Due to the increased nice value, the PRI value of the process (70) is also adjusted from 62 in the previous example. Remember that the higher the PRI value, the lower the priority.

20.2.1 Reducing the priority of a process

The priority of the process can be reduced by *increasing* the nice value. When using the `nice` command without any increment, it increase the nice value of a process with 10. Example 20-3 shows that we change the `ps` command.

Example 20-3 Using the nice command

```
lpar05:/>> nice ps -l
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
200001	A	0	23446	32122	2	65	30	19982	1296		pts/6	0:00	ps
240001	A	0	32122	28162	0	60	20	f991e	588		pts/6	0:00	ksh

You can specify the increment, such as:

```
nice -10 ps -l
```

20.2.2 Improving the priority of a process

The priority of a process can be improved by *decreasing* the nice value. To decrease the nice value by 10, enter:

```
nice --10 ps -l
```

Example 20-4 shows the output of the command. The priority of the process is improved and is now 51 (lower numerical value means higher priority).

Example 20-4 Decreasing the nice value

```
lpar05:/>> nice --10 ps -l
```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
200001	A	0	13904	18892	3	51	10	3706	512		pts/3	0:00	ps
240001	A	0	18892	14224	0	60	20	af15	1012		pts/3	0:01	ksh

20.3 renice

The syntax of the `renice` command is:

```
renice [ -n Increment ] [ -g | -p | -u ] ID ...
```

Flags

- g** Interprets all IDs as unsigned decimal integer process group IDs.
- n Increment** Specifies the number to add to the nice value of the process. The value of Increment can only be a decimal integer from -20 to 20. Positive increment values degrade priority. Negative increment values require appropriate privileges and improve priority.
- p** Interprets all IDs as unsigned integer process IDs. The -p flag is the default if you specify no other flags.
- u** Interprets all IDs as user name or numerical user IDs.

Parameters

- ID** Where the -p option is used, this will be the value of the process identification number (PID). In the case where the -g flag is used, the value of ID will be the process group identification number (PGID). Finally, where the -u flag is used, this value denotes the user identification number (UID). Alternately, when using the -u flag, the user's name can also be used as the argument.
- Increment** A decimal integer in the range of -1 to -20 is used to improve the priority of a command. A decimal integer in the range of 0 (zero) to 20 is used to degrade the priority of a command.

20.3.1 Information about measurement and sampling

The priority of a thread that is currently running on the system can be changed by using the **renice** command to change the nice value for the process that contains the thread. The nice value can be displayed by using -l flag with the **ps** command. See Example 20-5 on page 354 for a detailed output of the **ps -l** command. Any user can use the **renice** command on any of his own running processes to decrease the nice value. A user with root authority can increase or decrease the nice value of any process.

For detailed information about how thread priorities are calculated on AIX refer to 1.2.2, "Processes and threads" on page 6.

20.4 Examples for renice

The following examples show the use of the -n Increment flag applied by a user with root authority.

In Example 20-5, we run `ps -l`. It can be seen that the thread with PID 18220 (`sleep`) is initially running with a nice value of 24. This is a typical value for a thread spawned from the korn shell that is running in the background.

Example 20-5 The effect of the nice value on priority

```
# ps -l
  F S UID    PID  PPID  C PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
240001 A 207 17328 19766  0  67 20 d2fe 1016 70023a44 pts/7 0:00 ksh
200001 A 207 18220 17328  0  68 24 f31b  236 30bf65d8 pts/7 0:00 sleep
```

In the next step, the `renice` command is used to increase the nice value of the process by 10 and therefore degrades its priority, as shown in Example 20-6.

Example 20-6 Degrading a thread's priority using renice

```
# renice -n 10 -p 18220
# ps -lu fred
  F S UID    PID  PPID  C PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
240001 A 207 17328 19766  0  67 20 d2fe 1016 70023a44 pts/7 0:00 ksh
200001 A 207 18220 17328  0  88 34 f31b  236 30bf65d8 pts/7 0:00 sleep
```

After this, the nice value is displayed as 34. The root user then invokes the `renice` command again using an increment value of -20 as shown in Example 20-7.

Example 20-7 Improving a thread's priority using renice

```
# renice -n -20 -p 18220
# ps -lu fred
  F S UID    PID  PPID  C PRI NI ADDR  SZ  WCHAN  TTY  TIME CMD
240001 A 207 17328 19766  0  67 20 d2fe 1016 70023a44 pts/7 0:00 ksh
200001 A 207 18220 17328  0  54 14 f31b  236 30bf65d8 pts/7 0:00 sleep
```

The result is that the nice value for this thread now decreases to 14 and the priority of the thread improves.

Refer to 1.2.2, “Processes and threads” on page 6 for detailed information about calculating a thread's priority.

The time and timex commands

The **time** command reports the real time, the user time, and the system time taken to execute a command. This command can be useful for determining the length of time a command takes to execute. To use this tool effectively, it is necessary to have a second report generated on the system for comparison. It is also important to take into consideration the workload on the system at the time the command is run.

Attention: The **time** command mentioned here is found in `/usr/bin`. If the **time** command is executed without the pathname, then the shell's own **time** command will be executed.

The **timex** command reports the real time, user time, and system time to execute a command. Additionally, the **timex** command has the capability of reporting various statistics for the command being executed. The **timex** command can output the same information that can be obtained from the **sar** command by using the `-s` flag. The output of the **timex** command is sent to standard error.

The **time** command resides in `/usr/bin` and is part of the `bos.rte.misc_cmds` fileset. The **timex** command resides in `/usr/bin` and is part of the `bos.acct` fileset. Both are installable from the AIX base installation media.

21.1 time

The syntax of the **time** command is:

```
/usr/bin/time [ -p ] Command [ Argument ... ]
```

Flags

-p Writes the timing output to standard error. Seconds are expressed as a floating-point number with at least one digit following the radix character.

Parameters

Command The command that will be timed by the **time** command.
Argument The command's arguments.

21.1.1 Information about measurement and sampling

The **time** command simply counts the CPU ticks from when the command that was entered as an argument is started until that command completes.

21.1.2 Examples for time

In Example 21-1, the **time** command is used to determine the length of time to calculate 999^{9999} .

Example 21-1 Using the time command to determine the duration of a calculation

```
# /usr/bin/time bc <<! >/dev/null  
> 999^9999  
> !  
real    0m27.55s  
user    0m27.24s  
sys     0m0.28s
```

The result shows that the CPU took 27.55 seconds of real time to calculate the answer. The output of the command has purposely been redirected to `/dev/null` so that the answer to the calculation is not displayed. The time values are displayed because the **time** command forces its output to standard error, which is the screen display. The time results are split into 0.28 seconds of system time and 27.24 seconds of user time.

System time	This is the time that the CPU spent in kernel mode.
User time	This is the time the CPU spent in user mode.
Real time	This is the elapsed time.

On SMP systems, the real time reported can be less than the sum of the user and system times. The reason that this can occur is that the process threads can be executed over multiple CPUs. The user time displayed by the `time` command in this case is derived from the sum of all of the CPU user times. In the same way, the system time as displayed by the `time` command is derived from the sum of all of the CPU system times.

21.2 `timex`

The syntax of the `timex` command is:

```
timex [ -o ] [ -p ] [ -s ] Command
```

Flags

- o** Reports the total number of blocks read or written, and total characters.
- p** Lists process accounting records for a command and all of its children. The number of blocks read or written and the number of characters transferred are reported. The `-p` flag takes the `f`, `h`, `k`, `m`, `r`, and `t` arguments defined in the `acctcom` command to modify other data items.
- s** Reports total system activity during the execution of the command. All data items listed in the `sar` command are reported.

Parameters

Command The command that the `timex` command will time and determine process statistics for.

21.2.1 Information about measurement and sampling

The `timex -s` command uses the `sar` command to acquire additional statistics. The output of the `timex` command, when used with the `-s` flag, produces a report similar to the output obtained from the `sar` command with various flags. For further information, refer to 9.1, “`sar`” on page 140. Because the `sar` command is intrusive, the `timex -s` command is also intrusive. The data reported by the `timex -s` command may not precisely reflect the behavior of a program in an unmonitored system. Using the `time` or `timex` commands to measure the user or system time of a string of commands, connected by pipes, entered on the command line is not recommended. A potential problem is that syntax oversights can cause the `time` or `timex` commands to measure only one of the commands and no error will be indicated. The syntax is technically correct; however the `time` or `timex` command may not measure the entire command.

21.2.2 Examples for timex

Example 21-2 shows the format of the `timex -s` command.

Example 21-2 The timex command showing sar-like output with the -s flag

```
# timex -s bc <<! >/dev/null
> 999^9999
> !
real 27.33
user 27.20
sys 0.12

AIX wlmhost 1 5 000BC6AD4C00 05/07/01

08:12:44 %usr %sys %wio %idle
08:13:11 23 0 0 76

08:12:44 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
08:13:11 0 0 0 0 0 0 0 0 0

08:12:44 slots cycle/s fault/s odio/s
08:13:11 241210 0.00 10.52 0.00

08:12:44 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
08:13:11 0 0 0 0 0 0

08:12:44 scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
08:13:11 1786 77 960 0.09 0.13 265227 1048

08:12:44 cswch/s
08:13:11 280

08:12:44 iget/s lookupp/s dirblk/s
08:13:11 0 8 0

08:12:44 runq-sz %runocc swpq-sz %swpocc
08:13:11 1.0 100

08:12:44 proc-sz inod-sz file-sz thrd-sz
08:13:11 90/262144 473/42034 655/853 166/524288

08:12:44 msg/s sema/s
08:13:11 0.00 0.00
```

The following fields hold the information that **sar** displays when used with the **-a** flag. This information pertains to the use of file system access routines:

<code>dirblk/s</code>	Number of 512-byte blocks read by the directory search routine to locate a directory entry for a specific file.
<code>iget/s</code>	Calls to any of several inode lookup routines that support multiple file system types. The <code>iget</code> routines return a pointer to the inode structure of a file or device.
<code>lookupn/s</code>	Calls to the directory search routine that finds the address of a vnode given a path name.

The following fields from the **timex -s** report show the **sar -b** equivalent information. The information pertains to buffer activity for transfers, access and caching:

<code>bread/s, bwrit/s</code>	Reports the number of block I/O operations. These I/Os are generally performed by the kernel to manage the block buffer cache area, as discussed in the description of the <code>lread/s</code> value.
<code>lread/s, lwrit/s</code>	Reports the number of logical I/O requests. When a logical read or write to a block device is performed, a logical transfer size of less than a full block size may be requested. The system accesses the physical device units of complete blocks and buffers these blocks in the kernel buffers that have been set aside for this purpose (the block I/O cache area). This cache area is managed by the kernel so that multiple logical reads and writes to the block device can access previously buffered data from the cache and require no real I/O to the device. Application read and write requests to the block device are reported statistically as logical reads and writes. The block I/O performed by the kernel to the block device in management of the cache area is reported as block reads and block writes.
<code>pread/s, pwrit/s</code>	Reports the number of I/O operations on raw devices. Requested I/O to raw character devices is not buffered, as it is for block devices. The I/O is performed to the device directly.
<code>%rcache, %wcache</code>	Reports caching effectiveness (cache hit percentage). This percentage is calculated as: $[100 \times (lreads - breads) / lreads]$.

The following fields displayed by **timex -s** command are the equivalent of the **sar -c** command. The information is *not* processor specific:

exec/s, fork/s	Reports the total number of fork and exec system calls.
sread/s, swrit/s	Reports the total number of read/write system calls.
rchar/s, wchar/s	Reports the total number of characters transferred by read/write system calls.
scall/s	Reports the total number of system calls.

The following fields of the **timex -s** command show the same information as the **sar -m** command. The fields show the message and semaphore information for the process:

msg/s	Reports the number of IPC message primitives.
sema/s	Reports the number of IPC semaphore primitives.

The following fields are the **timex -s** commands equivalent to the **sar -q** output. The queue statistics for the process are displayed:

runq-sz	Reports the average number of kernel threads in the run queue.
%runocc	Reports the percentage of the time the run queue is occupied.
swpq-sz	Reports the average number of kernel threads waiting to be paged in.
%swpocc	Reports the percentage of the time the swap queue is occupied.

The following **timex -s** output fields show paging statistics. The output is similar to that from the **sar -r** command. However, information displayed is for the process executed as the **timex -s** argument:

cycle/s	Reports the number of page replacement cycles per second.
fault/s	Reports the number of page faults per second. This is not a count of page faults that generate I/O because some page faults can be resolved without I/O.
slots	Reports the number of free pages on the paging spaces.
odio/s	Reports the number of non-paging disk I/Os per second.

The following fields of the **timex -s** command are the process equivalent of the **sar -u** command. The fields display CPU usage:

%usr	Reports the percentage of time the CPU or CPUs spent in execution at the user (or application) level.
%sys	Reports the percentage of time the CPU or CPUs spent in execution at the system (or kernel) level.
%wio	Reports the percentage of time the CPU or CPUs were idle while the system had outstanding disk/NFS I/O requests.
%idle	Reports the percentage of time the CPU or CPUs were idle with no outstanding disk I/O requests.

The following fields show the status of the kernel process, kernel thread, inode, and file tables. This output from the **timex** command is the equivalent of the **sar -v** command except that the **timex** output is process-specific:

file-sz, inod-sz, proc-sz, thrd-sz Reports the number of entries in use for each table.

The following **timex -s** field shows the system switch activity and is the process equivalent of the **sar -w** command:

pswch/s Reports the number of context switches per second.

The following fields of the **timex -s** command are the process equivalent of the **sar -y** command. The fields show tty device activity per second for the process:

canch/s	Reports tty canonical input queue characters. This field is always 0 (zero) for AIX Version 4 and later versions.
mdmin/s	Reports tty modem interrupts.
outch/s	Reports tty output queue characters.
rawch/s	Reports tty input queue characters.
revin/s	Reports tty receive interrupts.
xmtin/s	Reports tty transmit interrupts.

Memory-related performance tools

This part describes the tools that tune and monitor the performance data and statistics relevant to memory. Other memory-related commands not listed here might appear in the Chapter 2, “Multi-resource monitoring and tuning tools” on page 67.

This part contains detailed information about the following memory monitoring and tuning tools:

- ▶ The **ipcs** command described in Chapter 22, “The ipcs command” on page 365 is used to report the status information of active Inter Process Communications (IPC) facilities.

- ▶ The **rmss** command described in Chapter 23, “The rmss command” on page 379 is used to ascertain the effects of reducing the amount of available memory on a system without the need to physically remove memory from the system.
- ▶ The **svmon** command described in Chapter 24, “The svmon command” on page 387 is useful for determining which processes, users, programs, and segments are consuming the most paging space and real and virtual memory.

The `ipcs` command

The `ipcs` command reports status information about active Inter Process Communication (IPC) facilities. If you do not specify any flags, the `ipcs` command writes information in a short form about currently active message queues, shared memory segments, and semaphores.

This command is not a performance tool per se, but it can be useful in the following two scenarios:

- ▶ For application developers who use IPC facilities and need to verify the allocation and monitoring of IPC resources
- ▶ For system administrators who need to clean up after an application program that uses IPC mechanisms that have failed to release previously allocated IPC facilities¹

`ipcs` resides in `/usr/bin` and is part of the `bos.rte.control` fileset, which is installed by default from the AIX base installation media.

Other commands related to `ipcs` are `ipcrm` and `slibclean`. Consult *AIX 5L Version 5.2 Commands Reference* for more information about these commands.

¹ Terminating a process with the SIGTERM signal prevents orderly cleanup of the process resources such as shared memory segments.

22.1 ipcs

The syntax of the **ipcs** command is:

```
ipcs [ -m ] [ -q ] [ -s ] [ -a | -b -c -o -p -t ] [ -CCoreFile ] [ -N Kernel ]
```

Flags

-a	Uses the -b, -c, -o, -p, and -t flags.
-b	Reports the maximum number of bytes in messages on queue for message queues, the size of segments for shared memory, and the number of semaphores in each semaphores set.
-c	Reports the login name and group name of the user who made the facility.
-CCoreFile	Uses the file specified by the CoreFile parameter in place of the /dev/mem file.
-m	Reports information about active shared memory segments.
-NKernel	Uses the specified Kernel. (The /usr/lib/boot/unix file is the default.)
-o	Reports message queue and shared memory segment information.
-p	Reports process number information.
-q	Reports information about active message queues.
-s	Reports information about active semaphore set.
-t	Reports time information.

22.1.1 Information about measurement and sampling

The **ipcs** command uses /dev/mem to obtain information about IPC facilities in the system. The sampling is performed once every time the command is run, but **ipcs** executes as a user process and the IPC information can change while **ipcs** is running, so the information it gives is guaranteed to be accurate only at the time it was retrieved.

22.1.2 Examples for ipcs

Examples for using the **ipcs** command follow.

Checking IPC message queues

You can use `ipcs` to check IPC message queues, semaphores, and shared memory. The default report shows basic information about all three IPC facilities, as shown in Example 22-1.

Example 22-1 Using the ipcs command

```
# ipcs
IPC status from /dev/mem as of Wed May 23 17:25:03 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP
Message Queues:
q      0 0x4107001c -Rrw-rw----   root   printq

Shared Memory:
m      0 0x580508f9 --rw-rw-rw-   root   system
m      1 0xe4663d62 --rw-rw-rw-  imnadm  imnadm
m      2 0x9308e451 --rw-rw-rw-  imnadm  imnadm
m      3 0x52e74b4f --rw-rw-rw-  imnadm  imnadm
m      4 0xc76283cc --rw-rw-rw-  imnadm  imnadm
m      5 0x298ee665 --rw-rw-rw-  imnadm  imnadm
m     131078 0xffffffff --rw-rw----   root   system
m      7 0x0d05320c --rw-rw-rw-   root   system
m     393224 0x7804129c --rw-rw-rw-   root   system
m     262153 0x780412e3 --rw-rw-rw-   root   system
m     393226 0xffffffff --rw-rw----   root   system
m     393227 0xffffffff --rw-rw----   root   system

Semaphores:
s     262144 0x580508f9 --ra-ra-ra-   root   system
s      1 0x440508f9 --ra-ra-ra-   root   system
s     131074 0xe4663d62 --ra-ra-ra-  imnadm  imnadm
s      3 0x62053142 --ra-r--r--   root   system
...(lines omitted)...
s      20 0xffffffff --ra-----   root   system
s      21 0xffffffff --ra-----   root   system
```

The default `ipcs` report column headings and meanings are:

T The type of facility. There are three facility types:

- q** Message queue
- m** Shared memory segment
- s** Semaphore

ID The identifier for the facility entry.

KEY The key used as a parameter to the `msgget` subroutine, the `semget` subroutine, or the `shmget` subroutine to make the facility entry.

MODE The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows. The first two characters can be any of the following:

- R** If a process is waiting on a msgrcv system call.
- S** If a process is waiting on a msgsnd system call.
- D** If the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches from it.
- C** If the associated shared memory segment is to be cleared when the first attach is run.
 - If the corresponding special flag is not set.

The next nine characters are interpreted as three sets of 3 bits each. The first set refers to the owner's permissions, the next to permissions of others in the user group of the facility entry, and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows:

- r** If read permission is granted
- w** If write permission is granted
- a** If alter permission is granted
- If the indicated permission is not granted

OWNER The login name of the owner of the facility entry.

GROUP The name of the group that owns the facility entry.

Checking processes that use shared memory

To find out which processes use shared memory, we can use the -m (memory) and -p (processes) flags together, shown in Example 22-2.

Example 22-2 Using ipcs -mp

```
# ipcs -mp
IPC status from /dev/mem as of Thu May 24 23:30:47 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP      CPID      LPID
Shared Memory:
m      0 0x580508f9 --rw-rw-rw-   root      system     5428     5428
m      1 0xe4663d62 --rw-rw-rw-  imnadm    imnadm    14452    14452
m      2 0x9308e451 --rw-rw-rw-  imnadm    imnadm    14452    14452
m      3 0x52e74b4f --rw-rw-rw-  imnadm    imnadm    14452    14452
m      4 0xc76283cc --rw-rw-rw-  imnadm    imnadm    14452    14452
m      5 0x298ee665 --rw-rw-rw-  imnadm    imnadm    14452    14452
m      6 0xffffffff --rw-rw----   root      system     5202     5202
m      7 0x7804129c --rw-rw-rw-   root      system    17070    20696
m      8 0x0d05320c --rw-rw-rw-   root      system    19440    23046
```

The output shows one shared memory segment that is used by the SPMI API library is 0x7804129c (see 41.2, “System Performance Measurement Interface” on page 805 for more details about SPMI API), the process ID of the process that created this shared memory segment is 17070, and the PID that last used it is 20696. To examine the process with process ID 17070, use the **ps** command (see Chapter 8, “The ps command” on page 127 for more details), as shown in Example 22-3 below.

Example 22-3 Using ps

```
# ps -eo comm,pid,user,group|grep 17070
topas    17070    root    system
```

As can be seen from the **ps** output above, it is the **topas** command that uses the 0x7804129c shared memory segment and it is run by the **root** user in the **system** group, which is the same user in the same group that owns the shared memory segment as shown by the **ipcs** command in Example 22-2 on page 368. To identify all users who use the shared memory segment, use the **-S** option with **ipcs** and the **svmon**. Refer to “Removing an unused shared memory segment” on page 370.

The column headings and the meaning of the columns in a **ipcs** report with the **-p** flag are:

- T** The type of facility. There are three facility types:
 - q** Message queue
 - m** Shared memory segment
 - s** Semaphore
- ID** The identifier for the facility entry.
- KEY** The key used as a parameter to the msgget subroutine, the semget subroutine, or the shmget subroutine to make the facility entry.
- MODE** The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows. The first two characters could be:
 - R** If a process is waiting on a msgrcv system call.
 - S** If a process is waiting on a msgsnd system call.
 - D** If the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches from it.
 - C** If the associated shared memory segment is to be cleared when the first attach is run.
 - If the corresponding special flag is not set.

The next nine characters are interpreted as three sets of 3 bits each. The first set refers to the owner’s permissions, the next to

permissions of others in the user group of the facility entry, and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows:

- r** If read permission is granted
- w** If write permission is granted
- a** If alter permission is granted
- If the indicated permission is not granted

OWNER The login name of the owner of the facility entry.

GROUP The name of the group that owns the facility entry.

CPID The PID of the creator of the shared memory entry.

LPID The PID of the last process to attach or detach the shared memory segment.

Removing an unused shared memory segment

If a process that has allocated shared memory does not explicitly detach it before terminating, it can be identified with **ipcs** and then removed by using the **ipcrm** and **slibclean** commands. The **ipcrm** command will detach the specified shared memory identifier. The shared memory segment and data structure associated with it are also removed after the last detach operation. The key of a shared memory segment is changed to `IPC_PRIVATE` when the segment is removed until all processes attached to the segment detach from it. The **slibclean** command will remove any currently unused modules in kernel and library memory.

To look for shared memory segments not used by a no process, use the **ipcs** with the `-mpS` flags as in Example 22-4. Note that the segment ID (SID) is reported after each shared memory line.

Example 22-4 Using ipcs -mpS to view shared memory

```
# ipcs -mpS
IPC status from /dev/mem as of Mon Jun  4 17:42:51 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP  CPID  LPID
Shared Memory:
m          0 0x580508f9 --rw-rw-rw-   root    system  5180  5180

SID :
0x9c1
...(lines omitted)...
m    393226 0x7804129c --rw-rw-rw-   root    system 17048 17048

SID :
0x9d33
```

Then use the **svmon** command to check whether there are any processes that use the shared memory segments shown in the **ipcs** output. Use the **-l** and **-S** flag with the **svmon** command as shown in Example 22-5.

Note: To check all shared memory segments at once, use the command:

```
ipcs -mS|awk '/^0x/{print substr($1,3)}'|xargs -i svmon -lS {}
```

Example 22-5 Using svmon -lS to check processes using segments

```
# svmon -lS 9d33
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
9d33	c	work	shmat/mmap	398	0	0	398
pid(s)= 17048							

If there are process IDs (PIDs) reported on the **pid(s)** line, check if the processes still exist with the **ps** command as Example 22-6 shows.

Example 22-6 Using ps -u to check for active processes

```
# ps -p 17048
```

PID	TTY	TIME	CMD
17048	-	0:04	topas

In this example the PID (17048) still exists. If **ps** only shows the column headers, it is safe to use the **ipcrm** command to remove each unused shared memory segment:

```
ipcrm -M 0x7804129c
```

The **ipcrm** command removes the shared memory segment 0x7804129c. After this has been done, use the **slibclean** command:

```
slibclean
```

Neither the **ipcrm** nor **slibclean** command should display any messages when executed properly.

Using a shared memory segment

For more detailed information about how to program IPC facilities, review the *General Programming Concepts: Writing and Debugging Programs* and especially the section “Creating a Shared Memory Segment with the **shmat** Subroutine” before using shared memory segments in application programs.

Example shared memory program

Example 22-7 shows a sample program that manages a single shared memory segment.

Example 22-7 Example shared memory segment program

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <sys/types.h>
4  #include <sys/ipc.h>
5  #include <sys/shm.h>
6  #define IPCSZ 4096
7  static int    idfile = 0;
8  static char   *idpath = NULL;
9  static key_t  ipckey = 0;
10 static int    ipcid = 0;
11 static char   *ipcdata = NULL;
12 void
13 cleanup(int s)
14 {
15     if (ipcid && ipcdata) {
16         /*
17          * The shmdt subroutine detaches from the data segment of the
18          * calling process the shared memory segment.
19          */
20         if (shmdt(ipcdata) < 0) {
21             perror("shmdt");
22         }
23         /*
24          * Once created, a shared memory segment is deleted only when the
25          * system reboots or by issuing the ipcrm command or using the
26          * shmctl subroutine.
27          */
28         if (shmctl(ipcid,IPC_RMID,(void *)ipcdata) < 0) {
29             perror("shmctl");
30         }
31     }
32     close(idfile);
33     remove(idpath);
34     _cleanup ();
35     _exit (0);
36 }
37 main()
38 {
39     /*
40     * Create a unique shared memory id, this is very important!
41     */
42     if ((idpath = tempnam("/tmp","IPC:")) == NULL) {
43         perror("tempnam");
```

```

44     exit(1);
45 }
46 if ((idfile = creat(idpath,0)) < 0) {
47     perror("creat");
48     exit(2);
49 }
50 if ((ipckey = ftok(idpath,random()%128)) < 0) {
51     perror("ftok");
52     exit(3);
53 }
54 /*
55  * We make sure that we clean up the shared memory that we use
56  * before we terminate the process. atexit() is called when
57  * the process is normally terminated, and we trap signals
58  * that a terminal user, or program malfunction could
59  * generate and cleanup then as well.
60  */
61 atexit(cleanup);
62 signal(SIGINT,cleanup);
63 signal(SIGTERM,cleanup);
64 signal(SIGSEGV,cleanup);
65 signal(SIGQUIT,cleanup);
66 /*
67  * IPC_CREAT Creates the data structure if it does not already exist.
68  * IPC_EXCL Causes the shmget subroutine to be unsuccessful if the
69  * IPC_CREAT flag is also set, and the data structure already exists.
70  */
71 if ((ipcid = shmget(ipckey,IPCSZ,IPC_CREAT|IPC_EXCL|0700)) < 0) {
72     perror("shmget");
73     exit(4);
74 }
75 if ((ipcdata = (char *)shmat(ipcid,0,0)) < 0) {
76     perror("shmat");
77     exit(5);
78 }
79 /*
80  * Work with the shared memory segment...
81  */
82 bzero(ipcdata,IPCSZ);
83 strcpy(ipcdata,"Hello World!");
84 printf("ipcdata\t: %s\n",ipcdata);
85 bzero(ipcdata,IPCSZ);
86 strcpy(ipcdata,"Dude!");
87 printf("ipcdata\t: %s\n",ipcdata);
88 }

```

The program performs in three steps. The first step is the setup part where the unique shared memory key and the shared memory segment are created. This is done from line 42 to line 78. The `ftok` subroutine creates the 32-bit key ID by putting together the file's inode number, the file system device number, and the numeric ID used in the call. Be aware that in the case of two identical file systems where the same numeric ID is used to call `ftok`, `ftok` will return the same number when used in either system.

The second step is the actual data manipulation part. This is between line 82 and 87. The third step is the housekeeping part where all allocated resources from the setup part are removed, released, and freed. This is performed entirely in the `cleanup()` subroutine on lines 15 to 35.

Example 22-8 shows the result of the example program that stores text in the shared memory and then uses the `printf` subroutine to display the stored text.

Example 22-8 Sample program run

```
# shm
ipcdata : Hello World!
ipcdata : Dude!
```

Example 22-9 below shows how the `ipcs -mp` and `ps -p PID` command reports look while our sample program is running.

Example 22-9 Checking our shared memory program while running

```
# ipcs -mp
IPC status from /dev/mem as of Fri May 25 01:41:26 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP      CPID      LPID
Shared Memory:
m      0 0x580508f9 --rw-rw-rw-   root      system     5428     5428
m      1 0xe4663d62 --rw-rw-rw-   imnadm    imnadm    14452    14452
m      2 0x9308e451 --rw-rw-rw-   imnadm    imnadm    14452    14452
m      3 0x52e74b4f --rw-rw-rw-   imnadm    imnadm    14452    14452
m      4 0xc76283cc --rw-rw-rw-   imnadm    imnadm    14452    14452
m      5 0x298ee665 --rw-rw-rw-   imnadm    imnadm    14452    14452
m     131078 0xffffffff D-rw-rw----   root      system     5204     6252
m     262151 0x3d070079 --rw-----   root      system    23734    23734
m      8 0x0d05320c --rw-rw-rw-   root      system    19440    23046

# ps -p 5204,23734
  PID  TTY  TIME CMD
  5204  -   0:00 rncd
23734 pts/4 0:00 shm
```

In the output, the `ps` command checks the shared memory segment's two owner PIDs (5204 and 23734). The PID 23734 was our program's process with ID 262151 and key 0x3d070079. Example 22-10 shows the output of `ipcs -mp` and `ps -p PID` after the sample program has ended.

Example 22-10 Checking our shared memory program

```
# ipcs -mp
IPC status from /dev/mem as of Fri May 25 01:46:50 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP      CPID      LPID
Shared Memory:
m      0 0x580508f9 --rw-rw-rw-   root      system     5428     5428
m      1 0xe4663d62 --rw-rw-rw-  imnadm    imnadm    14452    14452
m      2 0x9308e451 --rw-rw-rw-  imnadm    imnadm    14452    14452
m      3 0x52e74b4f --rw-rw-rw-  imnadm    imnadm    14452    14452
m      4 0xc76283cc --rw-rw-rw-  imnadm    imnadm    14452    14452
m      5 0x298ee665 --rw-rw-rw-  imnadm    imnadm    14452    14452
m     262150 0xffffffff --rw-rw----   root      system     5206     5206
m      8 0x0d05320c --rw-rw-rw-   root      system    19440    23046
# ps -p 23734
  PID  TTY  TIME CMD
```

The output above shows that neither our shared memory segment nor the process that created and used it, exists any more.

Checking processes that use semaphores

Some applications based on a process model use *semaphores* to communicate numeric information between applications, such as status between child and parent processes. That is not using thread programming but the traditional UNIX style using the fork system call to split a process to execute in parallel in an SMP environment. In Example 22-11 we become aware of the fact that there are large amounts of semaphore activity per second by examining a `sar` report.

Example 22-11 sar report

```
# sar -m 5 3
AIX wlmhost 1 5 000BC6AD4C00 05/28/01
17:40:43  msg/s  sema/s
17:40:48    0.00 1352.21
17:40:53    0.00 1359.46
17:40:58    0.00 1353.09
Average    0.00 1354.93
```

We now use the **ipcs** command with the **-tas** flags to check which user(s) are using semaphores. Note that the **-t** flag shows the time when the last semaphore operation was completed. This is why we prefix the **ipcs** report with the current system time by using the date command as shown in Example 22-12.

Example 22-12 ipcs -tas

```
# date;ipcs -tas
Mon May 28 17:47:55 CDT 2001
IPC status from /dev/mem as of Mon May 28 17:43:02 CDT 2001
T      ID      KEY      MODE      OWNER      GROUP      CREATOR      CGROUP      NSEMS      OTIME      CTIME
Semaphores:
s      262144  0x580508f9 --ra-ra-ra-  root      system      root      system      1 17:17:21 17:17:21
...(lines omitted)...
s          13 0x010530ab --ra-----  root      system      root      system      1 17:28:24 17:28:24
s          14 0xffffffff --ra-ra-ra-  baluba    staff      baluba    staff      1 17:29:53 17:29:44
s          15 0xffffffff --ra-ra-ra-  baluba    staff      baluba    staff      1 17:30:51 17:30:42
...(lines omitted)...
s          185 0xffffffff --ra-ra-ra-  baluba    staff      baluba    staff      1 17:54:55 17:54:47
s          186 0xffffffff --ra-ra-ra-  baluba    staff      baluba    staff      1 17:55:04 17:54:55
s          187 0xffffffff --ra-ra-ra-  baluba    staff      baluba    staff      1 17:55:12 17:55:04
```

In the example output above we see that there are almost 200 semaphores on the system, created (the CREATOR column) by the baluba user. Now we can use the **ps** command to identify which programs this user is running, as shown in Example 22-13.

Example 22-13 ps command

```
# ps -fu baluba
      UID  PID  PPID  C   STIME  TTY  TIME CMD
baluba 14830 16412 66 17:55:54 pts/3 0:00 batchsync
baluba 15784 4618 0 17:28:21 pts/3 0:00 -ksh
baluba 16412 15784 66 17:55:54 pts/3 0:00 batchsync
```

The user is only running a command called batchsync, and its start time coincides with semaphore 186 in the previous output. To investigate further what the batchsync application is doing we could use other tools such as **tprof** (see 19.5, “tprof” on page 324) and **truss** (see Chapter 12, “The truss command” on page 191). The final example uses **truss** to monitor what system calls the batchsync application is executing. Note that because the batchsync process is restarted very frequently (the start time shown with the **ps** command is more related to the last semaphores created than the first), we use shell scripting to catch the process ID while it is still active, as shown in Example 22-14.

Example 22-14 Using truss

```
# truss -c -p $(ps -f -opid=,comm= -u baluba|awk '/batchsync/{print $1}')
syscall          seconds  calls  errors
```

_exit	.00	2	
__semop	.24	8677	
kfcntl	.00	4	
	----	---	---
sys totals:	.25	8683	0
usr time:			8.54
elapsed:			8.79

The **ps** command reports the process ID and command name for the user and pipes it to **awk**, which separates the process ID for the user and the batchsync application name. The process IDs are then used by **truss** to monitor and count what system calls the application performs and the number of calls made. As can be seen in the output above, there were 8677 calls made to **semop** during our tracking with **truss**.

To clean up all used semaphores if the application does not, execute the **ipcrm** command, as in Example 22-15, for the specified user.

Example 22-15 ipcrm

```
# ipcs -s|awk '/baluba/{print $2}'|xargs -ti ipcrm -s {}
...(lines omitted)...
ipcrm -s 348
ipcrm -s 349
```

First we use **ipcs** to report all semaphores, then **awk** to only print the specified user's semaphore IDs, and finally the **xargs** command to execute one **ipcrm** for each semaphore ID in the pipe.

The `rmss` command

The `rmss` (Reduced-Memory System Simulator) command is used to ascertain the effects of reducing the amount of available memory on a system without the need to physically remove memory from the system. It is useful for system sizing, as you can install more memory than is required and then use `rmss` to reduce it. Using other performance tools, the effects of the reduced memory can be monitored. The `rmss` command has the ability to run a command multiple times using different simulated memory sizes and produce statistics for all of those memory sizes.

The `rmss` command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

23.1 rmss

The syntax of the **rmss** command is:

```
rmss -c MemSize
rmss -r
rmss -p
rmss [ -d MemSize ][ -f MemSize ][ -n NumIterations ][ -o OutputFile ]
[ -s MemSize ] Command
```

Flags

-c MemSize

Changes the simulated memory size to the MemSize value, which is an integer or decimal fraction in units of megabytes. The MemSize variable must be between 4 MB and the real memory size of the machine. However, it is not recommended to reduce the simulated memory size to under 256 MB on a uniprocessor system. For systems containing larger amounts of memory, such as 16 GB to 32 GB, it is not recommended to reduce the simulated memory size to less than 1 GB due to inherent system structures such as the kernel. There is no default for this flag.

-d MemSize

Specifies the increment or decrement between memory sizes to be simulated. The MemSize value is an integer or decimal fraction in units of megabytes. If the -d flag is omitted, the increment will be 8 MB. Many systems produced have a large amount of memory. Therefore, it is recommended that when testing, you test in increments or decrements of 128 MB.

-f MemSize

Specifies the final memory size. You should finish testing the simulated system by executing the command being tested at a simulated memory size given by the MemSize variable, which is an integer or decimal fraction in units of megabytes. The MemSize variable may be set between 4 MB and the real memory size of the machine. However, for systems containing larger amounts of memory, for example 16 GB to 32 GB, it is not recommended to reduce the simulated memory size to under 1 GB due to inherent system structures such as the kernel. If the -f flag is omitted, the final memory size will be 8 MB.

-n NumIterations

Specifies the number of times to run and measure the command, at each memory size. There is no default for the -n flag. If the -n flag is omitted during **rmss** command initialization, the **rmss** command will determine how many iterations of the command being tested are necessary to

- accumulate a total run time of 10 seconds, and then run the command that many times at each memory size.
- o OutputFile** Specifies the file into which to write the `rmss` report. If the `-o` flag is omitted, then the `rmss` report is written to the file `rmss.out`. In addition, the `rmss` report is always written to standard output.
 - p** Displays the current simulated memory size.
 - r** Resets the simulated memory size to the real memory size of the machine.
 - s MemSize** Specifies the starting memory size. Start by executing the command at a simulated memory size specified by the `MemSize` variable, which is an integer or decimal fraction in units of megabytes. The `MemSize` variable must be between 4 MB and the real memory size of the machine. If the `-s` flag is omitted, the starting memory size will be the real memory size of the machine. It is difficult to start at a simulated memory size of less than 8 MB, because of the size of inherent system structures such as the kernel.

Parameters

Command Specifies the command to be run and measured at each memory size. The `Command` parameter may be an executable or shell script file, with or without command line arguments. There is no default command.

The `rmss` command must be run as the root user or a user who is part of the system group.

Important: Before running `rmss`, note the `schedo` parameters and disable `v_repage_hi`.

23.1.1 Information about measurement and sampling

Using the `rmss` command, you can measure the effects of limiting the amount of memory on the system.

Effective memory is reduced by stealing free page frames from the list of free frames maintained by the Virtual Memory Manager. These frames are kept in a pool of unusable frames and returned to the free list when effective memory is increased by `rmss`. The `rmss` command also adjusts other data structures and system variables that must be maintained at different memory settings.

The reports are generated to a file as specified by the `-o` option of the command line. It is advisable to run any tests at least twice (specify 2 or greater as a parameter for the `-n` option).

Measurements are taken on the completion of each executable or shell script as specified in the command line.

The `rmss` command reports “usable” real memory. `rmss` may report a different size than the size you specify. This is because the system may either have bad memory or `rmss` is unable to steal memory that is already pinned by the operating system such as by device drivers.

23.1.2 Recommendations and precautions

There are no problems with setting the memory size too high as you cannot exceed the maximum installed memory size. However, setting the memory size too low can lead to the following problems:

- ▶ Severe degradation of performance
- ▶ System hang
- ▶ High paging

You can recover from this scenario by following the procedure described in “Resetting the simulated memory size” on page 383

It is recommended that you do not set the simulated memory size of a uniprocessor system to less than 256 MB. For larger systems containing more than 16 GB of memory, the recommendation is that you reduce the simulated memory size to less than 256 MB.

This command is effective immediately and does not require a reboot. Any changes made are not permanent and will be lost upon rebooting.

23.1.3 Examples for `rmss`

This section shows examples of the most important report outputs with a detailed description of the output.

It is important to run the application multiple times for each memory size as this will eliminate the following scenarios:

- ▶ `rmss` can clear a large amount of memory, and the first time you run your application you may experience a longer run time while your application loads files. Also on subsequent runs of the application, as the program is already loaded, shorter run times may be experienced.

- ▶ Due to other factors within a complex UNIX environment, such as AIX, it may not be possible to produce the same run times as the previous program run.

Changing the simulated memory size

Simulated memory size can be changed (between 8 MB and total memory on the system) with the command shown in Example 23-1. In this case the simulated memory size is set to 512 MB.

Example 23-1 Changing simulated memory size

```
# rmss -c 512
Simulated memory size changed to 512 Mb.
```

Displaying the simulated memory size

To display the simulated memory size, use the command shown in Example 23-2.

Example 23-2 Displaying simulated memory size

```
# rmss -p
Simulated memory size is 512 Mb.
```

Resetting the simulated memory size

To reset the simulated memory size to the system's installed memory size, use the command shown in Example 23-3.

Example 23-3 Resetting simulated memory size

```
# rmss -r
Simulated memory size changed to 4096 Mb.
```

Testing an executable run time with rmss

To investigate the performance of the command `cc -O foo.c` with memory sizes 512, 384, and 256 MB, run and measure the command twice at each memory size, then write the report to the `cc.rmss.out` file, and enter:

```
rmss -s 512 -f 256 -d 128 -n 2 -o cc.rmss.out cc -O foo.c
```

To investigate the performance of `shell_script.sh` with different memory sizes from 256 MB to 512 MB, by increments of 64 MB; run and measure `shell_script.sh` twice at each memory size; and write the report to the `rmss.out` file, enter the following:

```
rmss -s 256 -f 512 -d 64 -n 2 -o rmss.out shell_script.sh
```

When any combination of the -s, -f, -d, -n, and -o flags is used, the **rmss** command runs as a driver program, which executes a command multiple times over a range of memory sizes and displays statistics describing the commands performance of the command at each memory size.

The following command sequence was performed to generate the example output shown in Example 23-4.

1. Create a 128 MB file called 128MB_file.
2. Create a shell script called shell_script.sh containing:

```
tar cvf /dev/null 128MB_file > /dev/null 2>&1
```

3. Run the command:

```
rmss -s 256 -f 1024 -d 128 -n 2 -o rmss.out shell_script.sh
```

Example 23-4 Screen output from rmss

```
# cat rmss.out
```

```
Hostname: bolshoi.itso.ibm.com
Real memory size: 4096 Mb
Time of day: Sun May 20 15:57:20 2001
Command: shell_script.sh
```

```
Simulated memory size initialized to 256 Mb.
```

```
Number of iterations per memory size = 1 warmup + 2 measured = 3.
```

Memory size (megabytes)	Avg. Pageins	Avg. Response Time (sec.)	Avg. Pagein Rate (pageins / sec.)
256	9.5	0.4	26.2
384	7.0	0.3	20.4
512	6.0	0.3	17.6
640	5.5	0.3	16.1
768	7.0	0.3	20.4
896	3.0	0.3	9.1
1024	2.5	0.3	7.6

```
Simulated final memory size.
```

The first few lines of the report gives general information, including the name of the machine that the **rmss** command was running on, the real memory size of that machine, the time and date, and the command that was being measured. The next two lines give informational messages that describe the initialization of the **rmss** command. Here, the **rmss** command displays that it has initialized the simulated memory size to 256 MB, which was the starting memory size given

with the `-s` flag. Also, the `rmss` command prints out the number of iterations that the command will be run at each memory size. Here, the command is to be run three times at each memory size; once to warm up and twice when its performance is measured. The number of iterations was specified by the `-n` flag.

The lower part of the report provides the following for each memory size the command was run at:

- ▶ The memory size, along with the average number of page-ins that occurred while the command was run
- ▶ The average response time of the command
- ▶ The average page-in rate that occurred when the command was run

Note: The average page-ins and average page-in rate values include all page-ins that occurred while the command was run, not just those initiated by the command.

The svmon command

The **svmon** command captures a snapshot of virtual memory, so it is useful for determining which processes, user programs, and segments are consuming the most real, virtual, and paging space memory. The **svmon** command can also do tier and class reports on Workload Manager.

The **svmon** command invokes the **svmon_back** command, which does the actual work.

The **svmon** command resides in `/usr/bin` directory and the **svmon_back** command resides in the `/usr/lib/perf` directory. Both are part of the `bos.perf.tools` fileset.

24.1 svmon

The syntax of the **svmon** command is:

```
svmon -G [ -i Interval [ NumIntervals ] ] [ -z ]
svmon -U [ LogName1...LogNameN ] [ -r ] [ -n | -s ] [ -w | -f -c ]
    [ -t Count ] [ -u | -p | -g | -v ] [ -i Interval [ NumIntervals ] ]
    [ -l ] [ -j ] [ -d ] [ -z ] [ -m ] [ -q ]
svmon -C Command1...CommandN [ -r ] [ -n | -s ] [ -w | -f | -c ]
    [ -t Count ] [ -u | -p | -g | -v ] [ -i Interval [ NumIntervals ] ]
    [ -l ] [ -j ] [ -d ] [ -z ] [ -m ] [ -q ]
svmon -W [ ClassName1...ClassNameN ] [-e] [ -r ] [ -n | -s ]
    [ -w | -f | -c ] [-t Count] [ -u | -p | -g | -v ]
    [ -i Interval [ NumIntervals]] [ -l ] [ -j ] [ -d ] [ -z ] [ -m ] [ -q ]
svmon -T [ Tier1...TierN ] [ -a SupClassName ] [ -x ] [ -e ] [ -r ]
    [ -u | -p | -g | -v ] [ -n | -s ] [ -w | -f | -c ] [ -t Count ]
    [ -i Interval [ NumIntervals ] ] [ -l ] [ -z ] [ -m ]
svmon -P [ PID1... PIDN ] [ -r [ -n | -s ] [ -w | -f | -c ] [ -t Count ]
    [ -u | -p | -g | -v ] [ -i Interval [ NumIntervals] ] [ -l ] [ -j ] [ -z ]
    [ -m ] [ -q ]
svmon -S [ SID1...SIDN ] [ -r ] [ -n | -s ] [ -w | -f | -c ]
    [ -t Count ] [ -u | -p | -g | -v ] [ -i Interval [ NumIntervals ] ]
    [ -l ] [ -j ] [ -z ] [ -m ] [ -q ]
svmon -D SID1..SIDN [ -b ] [ -i Interval [ NumIntervals ] ] [ -z ] [ -q ]
svmon -F [ Frame1..FrameN ] [ -i Interval [ NumIntervals ] ] [ -z ] [ -q ]
```

Flags

If no command line flag is given, then the -G flag is the default.

- a SupClassName** Restricts the scope to the subclasses of the SupClassName class parameter (in the Tier report -T). The parameter is a superclass name. No list of class is supported.
- b** Shows the status of the reference and modified bits of all displayed frames (detailed report -D). Once shown, the reference bit of the frame is reset. When used with the -i flag it detects which frames are accessed between each interval. This flag should be used with caution because of its performance impacts.
- c** Indicates that only client segments are to be included in the statistics. By default all segments are analyzed.
- C Command1...CommandN** Displays memory usage statistics for the processes running the command. All commands

- are strings that contain the exact basename of an executable file.
- d** Displays the memory statistics of the processes belonging to a given entity (user name or command name).
 - D SID1...SIDN** Displays memory-usage statistics for segments SID1...SIDN and a detail status of all frames of each segment.
 - e** Displays the memory-usage statistics of the subclasses of the Class parameter in the Workload Class report -W and in the Tier report -T. The class parameter of -W or -a must be a superclass name.
 - f** Indicates that only persistent segments (files) are to be included in the statistics. By default all segments are analyzed.
 - F [Frame1...FrameN]** Displays the status of frames Frame1...FrameN, including the segments that they belong to. If no list of frames is supplied, the percentage of memory used is displayed.
 - g** Indicates that the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space. This flag, together with the segment report, shifts the non-working segment at the end of the sorted list.
 - G** Displays a global report.
 - i Interval [NumIntervals]** Instructs the **svmon** command to display statistics repetitively. Statistics are collected and printed every Interval seconds. NumIntervals is the number of repetitions; if not specified, **svmon** runs until user interruption (Ctrl-C).
 - j** Shows, for each persistent segment, the file path referred. Note: This flag should be used with caution because of its potential performance impacts (especially with **svmon -S**).
 - l** Shows, for each displayed segment, the list of process identifiers that use the segment and, according to the type of report, the entity name (login, command, tier, or class) the process belongs to. For special segments a label is displayed instead of the list of process identifiers.

-m	Displays both source segment and mapping segment information when a segment is mapping a source segment. The default is to display only information about the mapping segment.
-n	Indicates that only non-system segments are to be included in the statistics. By default all segments are analyzed.
-p	Indicates that the information to be displayed is sorted in decreasing order by the total number of pages pinned.
-P [PID1... PIDN]	Displays memory usage statistics for processes PID1...PIDN. PID is a decimal value. If no list of process IDs (PIDs) is supplied, memory usage statistics are displayed for all active processes.
-q	Filters results regarding whether they deal with large pages or not. Additionally, it displays large page metrics.
-r	Displays the range(s) within the segment pages that have been allocated. A working segment may have two ranges because pages are allocated by starting from both ends and moving toward the middle.
-s	Indicates that only system segments are to be included in the statistics. By default all segments are analyzed.
-S [SID1...SIDN]	Displays memory-usage statistics for segments SID1...SIDN. SID is a hexadecimal value. If no list of segment IDs (SIDs) is supplied, memory usage statistics are displayed for all defined segments.
-t Count	Displays memory-usage statistics for the top Count object to be printed.
-T [Tier1...TierN]	Displays memory-usage statistics for all classes of the tier numbers Tier1...TierN. If no list of tiers is supplied, memory usage statistics are displayed for all defined tiers.
-u	Indicates that the information to be displayed is sorted in decreasing order by the total number of pages in real memory. This is the default sorting criteria if none of the following flags is present: -p, -g, and -v.

- U [LogName1...LogNameN]** Displays memory usage statistics for the login names LogName1...LogNameN. LogName is an exact login name string. If no list of login identifiers is supplied, memory usage statistics are displayed for all defined login identifiers.
- v** Indicates that the information to be displayed is sorted in decreasing order by the total number of pages in virtual space. This flag in conjunction with the segment report shifts the non-working segment at the end of the sorted list.
- w** Indicates that only working segments are to be included in the statistics. By default all segments are analyzed.
- W [Clnm1...ClnmN]** Displays memory usage statistics for the workload management class Clnm1...ClnmN. Clnm is the exact name string of a class. For a subclass, the name should have the form superclassname.subclassname. If no list of class names is supplied, memory usage statistics are displayed for all defined class names.
- x** Displays memory usage statistics for segments for every class of a tier in the Tier report -T.
- z** Displays the maximum memory size dynamically allocated by **svmon** during its execution.
- q** Reports only large page segments. In that case, global metrics are only related to these large page segments.

Parameters

- Interval** Statistics are collected and printed every Interval seconds.
- NumIntervals** The number of repetitions; if not specified, **svmon** runs until user interruption, Ctrl-C.

24.1.1 Information about measurement and sampling

When invoked, **svmon** captures a snapshot of the current contents of real, paging, and virtual memory, and summarizes the contents. Note that virtual pages include both real memory and paging space pages except for the -G report. Refer to “Analyzing the global report” on page 398.

The **svmon** command runs in the foreground as a normal user process. Because it can be interrupted while collecting data, it cannot be considered to be a true snapshot of the memory. **svmon** reports are based on virtual counters from the Virtual Memory Manager (VMM) for statistical analysis, and these might not always be current with the actual utilization. For these reasons you should be careful when analyzing the information received from running **svmon** snapshots on very busy systems with many processes because the data might have been updated by VMM while **svmon** is running.

svmon can be started either to take single snapshots or measure information at intervals. However, be aware that **svmon** can take several minutes to complete some functions, depending on the options specified and the system load. Because of this, the observed interval may be longer than what has been specified with the **-i** option.

Because processes and files are managed by VMM, and VMM's view of memory is as a segmented space, almost all of the **svmon** reports will concern segment usage and utilization. To gain the most benefits from the **svmon** reports you should understand what segments are and how they are used.

Segments

When a process is loaded into memory, its different parts (such as stack, heap, and program text) will be loaded into different segments. The same is true for files that are opened through the filesystem or are explicitly mapped.

A segment is a set of pages. It is the basic object used to report the memory consumption. Each segment is 256 MB of memory. The statistics reported by **svmon** are expressed in terms of pages. A page is a 4 KB block of *virtual memory*, while a frame is a 4 KB block of *real memory*. A segment can be used by multiple processes at the same time.

A segment belongs to one of the five following types:

persistent	Used to manipulate Journaled File System (JFS) files and directories
working	Used to implement the data areas of processes and shared memory segments
client	Used to implement some virtual file systems such as Network File System (NFS), the CD-ROM file system, and the Journaled File System 2 (J2)
mapping	Used to implement the mapping of files in memory
real memory map	Used to access the I/O space from the virtual address space

Note that a 64-bit system uses a different segmentation layout than a 32-bit system. Different segments are used for storing specific objects, such as process data and explicitly mapped files.

A dash (-) in the paging space utilization column indicates that the segment does not use paging space. For example, work segments use paging space, but persistent and client segments do not because they are read again from their stored location if the frames they occupied are freed. There are exceptions, such as when a mapped file is opened in a deferred update mode. A working segment will be stored on paging space because it is dynamic data and has no corresponding persistent storage area.

For more information about VMM and segmented memory usage, refer to:

- ▶ 1.3, “Memory performance” on page 12
- ▶ *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*
- ▶ *AIX 5L Version 5.2 System Management Guide: Operating System and Devices*
- ▶ *AIX 5L Version 5.2 Performance Management Guide*
- ▶ The `svmon` command in the *AIX 5L Version 5.2 Commands Reference, Volume 5*

24.1.2 Examples for svmon

Example 24-1 shows the default display when running `svmon`. To monitor on a continual basis, use the `-i` option flag with an interval number and a count number. For example, `svmon -i 5 12` takes a snapshot every five seconds, repeating 12 times. The default report from `svmon`, when run without option flags, shows systemwide memory utilization.

Example 24-1 svmon without options

```
lpar05:/>> svmon
```

	size	inuse	free	pin	virtual
memory	2097152	1155740	941412	134457	710778
pg space	524288	204410			
	work	pers	clnt	lpage	
pin	134457	0	0	0	
in use	526635	368039	261066	0	

In the first part of the output, usually we are most interested in the number of real memory pages that are `inuse` and `free`, as shown on the `memory` line. The number of `pg space` pages that are `inuse` show how many pages are actually in

use on the paging space(s). Other lines display information regarding pinned and used pages related to working, persistent, and client segments.

Determining which processes are using the most real memory

To list the process that is using the most real memory, run **svmon** with the **-P** and **-u** flags as shown in Example 24-2. Use the **-t** flag with *x* to display the top *x* processes. Output is sorted in decreasing order on the number of pages in real memory.

Example 24-2 Output from svmon -uP -t 3

```
# svmon -Pu -t 3 |grep -p Pid|grep '^.*[0-9]'
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
27952	java	247278	2520	4278	253857	N	Y	N
22966	IBM.CSMAgentR	8590	2529	5015	15621	N	Y	N
21718	aixterm	8557	2513	4278	15062	N	N	N

This output shows that the Java process uses the most memory. To calculate the amount of memory, multiply the *in use* value by 4096 (one page). Note that the system supports two virtual page sizes: traditional 4k pages and (since AIX 5.1 with the 5100-02 Recommended Maintenance package) 16MB large pages.

Determining which processes use the most paging space

Run **svmon** with the **-P** and **-g** flags to list the top paging space consumers in decreasing order. Example 24-3 shows the top three processes.

Example 24-3 svmon -gP -t 3

```
# svmon -gP -t 3 |grep -p Pid|grep '^.*[0-9]'
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
18330	rmcd	8241	2517	5020	15433	N	Y	N
22966	IBM.CSMAgentR	8590	2530	5015	15621	N	Y	N
23736	IBM.ERrmd	8066	2523	4886	15321	N	Y	N

The first process, **rmcd**, consumes the most paging space. The **Pgsp** field shows the number of 4 KB pages reserved or used in paging space by this process.

A **Pgsp** number that grows but never decreases may indicate a memory leak in the program. In this example, the **rmcd** process uses $5020 * 4096 = 20561920$ bytes or 20 MB paging space.

Displaying memory used by a WLM class

Use the `-W` flag with `svmon` to find out how much memory is used by processes belonging to a WLM class. Example 24-4 shows information about the shared memory segments.

Example 24-4 svmon -W shared

```
#svmon -W Shared
=====
Superclass          Inuse    Pin    Pgps Virtual
Shared             5164     0     1488   8184
  Vsid      Esid Type Description          LPage  Inuse  Pin  Pgps Virtual
b0036      - work                -    3444  0   26  6019
b8037      - work                -    551  0  954  1436
980f3      - pers /dev/hd2:4132    -    412  0   -   -
c0038      - work                -    177  0  278  416
40268      - pers /dev/hd2:4895 -    171  0   -   -
  60        - work                -    127  0  212  265
30266      - pers /dev/hd2:4916 -    91   0   -   -
800f0      - pers /dev/hd2:6251 -    54   0   -   -
382e7      - work                -    35   0   15   43
c01d8      - pers /dev/hd2:4199 -    33   0   -   -
48109      - pers /dev/hd2:4194 -    27   0   -   -
d81fb      - pers /dev/hd2:4200 -    15   0   -   -
38387      - pers /dev/hd2:338012 -    12   0   -   -
10142      - pers /dev/hd2:4188 -    8    0   -   -
f03fe      - pers /dev/hd2:336117 -    3    0   -   -
d829b      - pers /dev/hd2:4151 -    2    0   -   -
40048      - work                -    2    0   3    5
e01dc      - work                -    0    0   0    0
```

Finding out most utilized segments

With the `-S` option, `svmon` sorts segments by memory usage and displays the statistics for the top memory-usage segments. Example 24-5 shows the top 10 memory users by segment, by using the `-t` flag.

Example 24-5 svmon -S

```
#svmon -S -t 10
Vsid      Esid Type Description          LPage  Inuse  Pin  PgpsVirtual
128a2      - work                -    65536  0   0  65536
5a8ab      - work                -    65536  0   0  65536
30246      - work                -    65529  65529  0  65529
328a6      - work                -    65259  0   0  65259
8a8b1      - work                -    35759  0   0  35759
80010      - work page frame table -    30480  30480  0  30480
90012      - work kernel pinned heap -    25949  8115  54754  60395
15e0      - pers /dev/hd1:6197    -    19671  0   -   -
```

928b2	- work	-	6979	0	0	6979
88011	- work misc kernel tables	-	6684	0	4706	10410

Finding out what files a process or command is using

With **svmon**, persistent segment data (files and directory) display in `device:inode` format. The **ncheck** command maps `device:inode` to a file system and file name. Example 24-6 shows a sample report using **svmon** with the `-p` flag.

Example 24-6 svmon -pP

```
# svmon -pP 30752
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
30752	java	247008	2520	4278	253551	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4327	2511	4252	8303
7a8af	-	work		-	23	7	0	23
a28b4	2	work	process private	-	32	2	0	32
906d2	1	pers	code,/dev/hd2:34861	-	7	0	-	-
328a6	3	work	working storage	-	65259	0	0	65259
4a8a9	8	work	working storage	-	0	0	0	0
a8a1	f	work	shared library data	-	105	0	0	105
528aa	a	work	working storage	-	0	0	0	0
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-
5a8ab	5	work	working storage	-	65536	0	0	65536
128a2	4	work	working storage	-	65536	0	0	65536
828b0	9	work	working storage	-	0	0	0	0
8a8b1	6	work	working storage	-	35759	0	0	35759
b0036	d	work	shared library text	-	3444	0	26	6019
928b2	7	work	working storage	-	6979	0	0	6979
9a8b3	b	mmap	mapped to sid d14fa	-	0	0	-	-

Finding out which segments use paging space

To display information segments sorted by usage on paging space, use the **svmon** command with the `-S` and `-g` flags, as shown Example 24-7.

Example 24-7 svmon -gS

```
# svmon -gS
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
90012	-	work	kernel pinned heap	-	25949	8115	54754	60395
88011	-	work	misc kernel tables	-	6701	0	4706	10423
0	-	work	kernel seg	-	4327	2511	4252	8303
5004a	-	work		-	5605	5605	3309	8914
b8037	-	work		-	592	0	954	1477
582cb	-	work		-	0	0	653	653
b8437	-	work		-	287	2	573	730

7800f	- work page table area	-	513	2	524	526
88391	- work	-	82	2	524	536
420	- work	-	8	2	482	487
c0478	- work	-	14	2	393	398
904f2	- work	-	9	2	381	387
c8479	- work	-	48	2	374	418
10442	- work	-	46	2	367	381
90052	- work	-	49	2	363	385
40348	- work	-	9	2	352	354
20544	- work	-	9	2	345	351
20264	- work	-	71	2	345	390
28305	- work	-	111	2	342	420
40188	- work	-	47	2	337	348
c8339	- work	-	23	2	336	345
48309	- work	-	32	2	334	364

We can use the `-D` option to display frame information about each segment. Example 24-8 displays segment `Vsid 420`.

Example 24-8 svmon -D sid

```
lpar05:/>> svmon -D 12862
```

```
Segid: 12862
Type: persistent
LPage: N
Address Range: 0..1828
```

Page	Frame	Pin	ExtSegid	ExtPage
1828	1631094	N	-	-
1705	961115	N	-	-
1706	1631368	N	-	-
1707	1869959	N	-	-
1708	1631392	N	-	-
1709	1116454	N	-	-
1710	1436745	N	-	-
1711	1116458	N	-	-
1712	580493	N	-	-
1713	1789220	N	-	-
1714	897149	N	-	-
1715	1493724	N	-	-
1716	311427	N	-	-
1717	1675108	N	-	-

....(lines omitted).....

The output shows that the segment is a persistent segment. To compare the `-D` output with `-S` and `-r`, as is shown in Example 24-9 on page 398, we view a similar report of the frame address range (0..1828).

Example 24-9 svmon -rS sid

```
lpar05:/>> svmon -rS 12862
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	virtual
12862	-	pers	/dev/hd2:319685	-	913	0	-	-

Addr Range: 0..1828

stdin: END

If we use the -F option to look at the frame itself (as in Example 24-10), we monitor whether it is referenced or modified, but only over a very short interval.

Example 24-10 svmon -F

```
# svmon -F 93319 -i 1 5
```

Frame	Segid	Ref	Mod	Pincount	State	Swbits
93319	3ba7	Y	Y	0/0	In-Use	88000004
93319	3ba7	Y	N	0/0	In-Use	88000004
93319	3ba7	N	N	0/0	In-Use	88000004
93319	3ba7	N	N	0/0	In-Use	88000004
93319	3ba7	N	N	0/0	In-Use	88000004

Analyzing the global report

To monitor system memory utilization with **svmon**, use the -G flag. Example 24-11 shows the used and free sizes of real and virtual memory in the system.

Example 24-11 svmon -G

```
# svmon -G
```

memory	size	inuse	free	pin	virtual
	131047	26602	104445	13786	38574
pg space	262144	14964			
	work	pers	clnt		
pin	13786	0	0		
in use	20589	444	5569		

The column headings in a global report are:

memory	Specifies statistics describing the use of real memory, including:
size	Number of real memory frames (size of real memory). This includes any free frames that have been made unusable by the memory sizing tool, the <code>rmss</code> command.
inuse	Number of frames containing pages.
free	Number of frames free of all memory pools.
pin	Number of frames containing pinned pages.
virtual	Number of pages allocated in the system virtual space for working segments only (not all segment types).
stolen	Number of frames stolen by <code>rmss</code> and marked unusable by the VMM.
pg space	Specifies statistics describing the use of paging space.
size	Size of paging space.
inuse	Number of paging space pages used.
pin	Specifies statistics on the subset of real memory containing pinned pages, including:
work	Number of frames containing working segment pinned pages.
pers	Number of frames containing persistent segment pinned pages.
c1nt	Number of frames containing client segment pinned pages.
in use	Specifies statistics on the subset of real memory in use, including:
work	Number of frames containing working segment pages.
pers	Number of frames containing persistent segment pages.
c1nt	Number of frames containing client segment pages.

To show systemwide memory utilization, run `svmon` without any flags or only with the `-G` flag as shown in Example 24-12.

Example 24-12 svmon with the -G flag

```
# svmon -G
      size      inuse      free      pin      virtual
memory 131047    41502    89545    16749    62082
pg space 262144    29622

      work      pers      c1nt
pin    16749         0         0
in use 39004    2498         0
```

In the first part of the output we usually are most interested in the number of real memory pages that are `inuse` and `free`, as shown on the memory line. The number of `pg space` pages that are `inuse` is pages that are actually in use on the paging space(s). The last line, `in use`, shows the utilization of different memory segment types (`work`, `pers`, and `clnt`). Note that `clnt` indicates both NFS and JFS2 cached file pages (this actually includes CD-ROM filesystems) while the `pers` column shows cached JFS file pages.

Example 24-13 illustrates how the report looks when using the `rmss` command (see Chapter 23, “The `rmss` command” on page 379) to limit available memory for test purposes in the system. Note the additional column `stolen`.

Example 24-13 svmon report when using rmss

```
# rmss -s 128 $(whence svmon) -G

Hostname: wlmhost
Real memory size: 512 Mb
Time of day: Thu May 24 22:27:13 2001
Command: /usr/bin/svmon-G

Simulated memory size initialized to 128 Mb.

```

	size	inuse	free	pin	virtual	stolen
memory	131047	117619	13428	13784	134214	95584
pg space	262144	14964				

	work	pers	clnt
pin	13784	0	0
in use	116211	445	963

Analyzing memory utilization per user

The `-U` flag can be used with `svmon` to monitor users' memory utilization. The following series of examples shows how `svmon` reports the memory usage for a process by using the different optional flags with the `-U` flag. Without any user specification, the `-U` option reports on all users.

The column headings in a user report are:

User	Indicates the user name.
Inuse	Indicates the total number of pages in real memory in segments that are used by the user.
Pin	Indicates the total number of pages pinned in segments that are used by the user.
Pgsp	Indicates the total number of pages reserved or used on paging space by segments that are used by the user.

Virtual	Indicates the total number of pages allocated in the process virtual space.
Vsid	Indicates the virtual segment ID, which identifies a unique segment in the VMM.
Esid	Indicates the effective segment ID. The Esid is only valid when the segment belongs to the address space of the process. When provided, it indicates how the segment is used by the process. If the Vsid segment is mapped by several processes but with different Esid values, then this field contains '-'. In that case, the exact Esid values can be obtained through the -P flag applied on each of the process identifiers using the segment. A '-' also displays for segments used to manage open files or multi-threaded structures because these segments are not part of the user address space of the process.
Type	Identifies the type of the segment: pers indicates a persistent segment, work means a working segment, clnt means a client segment, map means a mapped segment, and rmap means a real memory mapping segment.
Description	<p>Gives a textual description of the segment. The content of this column depends on the segment type and usage:</p> <p>persistent JFS files in the format <device>:<inode>, such as /dev/hd1:123.</p> <p>working Data areas of processes and shared memory segments, dependent on the role of the segment based on the VSID and ESID.</p> <p>mapping Mapped to source segment IDs.</p> <p>client NFS, CD-ROM, and JFS2 files, dependent on the role of the segment based on the VSID and ESID.</p> <p>rmapping I/O space mapping dependent on the role of the segment based on the VSID and ESID.</p>
Inuse	Indicates the number of pages in real memory in this segment.
Pin	Indicates the number of pages pinned in this segment.
Pgsp	Indicates the number of pages used on paging space by this segment. This field is relevant only for working segments.
Virtual	Indicates the number of pages allocated for the virtual space of the segment.

The segments used by the processes are separated into three categories:

- SYSTEM Segments shared by all processes.
- EXCLUSIVE Segments used by the set of processes belonging to the specified user.
- SHARED Segments shared by several users.

The global statistics for the specified user is the sum of each of the following fields: Inuse, Pin, Pgps, and Virtual of the segment categories SYSTEM, EXCLUSIVE, and SHARED.

Source segment and mapping segment

The optional -m flag displays source segment and mapping segment information, as shown in Example 24-14.

Example 24-14 svmon -U user -m

```
# svmon -U hennie -m
```

```
=====
User                               Inuse   Pin    Pgps  Virtual  LPageCap
hennie                             8164   2515   4278  14645    N

.....
SYSTEM segments                    Inuse   Pin    Pgps  Virtual
                                   4327   2511   4252   8303

Vsid   Esid Type Description                LPage  Inuse   Pin Pgps Virtual
0      0 work kernel seg                    -    4327  2511 4252  8303

.....
EXCLUSIVE segments                Inuse   Pin    Pgps  Virtual
                                   184     4     0     171

Vsid   Esid Type Description                LPage  Inuse   Pin Pgps Virtual
a2a54  2 work process private                -     95     2  0    95
aaa55  2 work process private                -     35     2  0    35
6aa4d  f work shared library data           -     22     0  0    22
9aa53  f work shared library data           -     19     0  0    19
b8057  1 pers code,/dev/hd2:6230             -      8     0  -    -
ea0dd  - pers /dev/hd2:204925                 -      1     0  -    -
7aa4f  - pers /dev/hd1:4164                   -      1     0  -    -
1a0e3  - pers /dev/hd2:206940                 -      1     0  -    -
f20de  - pers /dev/hd2:204932                 -      1     0  -    -
ba0d7  - pers /dev/hd2:204911                 -      1     0  -    -
```



```

.....
SHARED segments
      Inuse      Pin      Pgps Virtual
      3653       0       26   6171

Vsid      Esid Type Description      LPage Inuse  Pin Pgps Virtual
b0036     d work shared library text      -   3597  0  26  6171
800f0     1 pers code,/dev/hd2:6251      -    54   0   -   -
f80df     - pers /dev/hd4:2              -    1   0   -   -
700ee     - pers /dev/hd2:2              -    1   0   -   -

```

Displaying segments for all processes belonging to a user

The optional `-d` flag displays, for a given entity, the memory statistics of the processes belonging to the specified user. When the `-d` flag is specified, the statistics are followed by the information about all processes run by the specified user. The `svmon` command displays information about the segments used by these processes. This set of segments is separated into three categories; segments that are flagged *system* by the Virtual Memory Manager (VMM), segments that are *only* used by the set of processes belonging to the specified user, and segments that are *shared* among several users (Example 24-15).

Example 24-15 svmon -U user -d

```

lpar05:/hennie/svmon>> svmon -U hennie -d

=====
User      Inuse      Pin      Pgps Virtual LPageCap
hennie    8196      2516     4278   14682     N

-----
      Pid Command      Inuse      Pin      Pgps Virtual 64-bit Mthrd LPage
      20058 ksh          8124      2514     4278   14619     N   N   N
      31962 find          8024      2514     4278   14565     N   N   N

.....
SYSTEM segments
      Inuse      Pin      Pgps Virtual
      4327      2512     4252   8303

Vsid      Esid Type Description      LPage Inuse  Pin Pgps Virtual
0         0 work kernel seg          -   4327  2512 4252  8303

.....
EXCLUSIVE segments
      Inuse      Pin      Pgps Virtual
      189       4       0       180

Vsid      Esid Type Description      LPage Inuse  Pin Pgps Virtual
a2a54     2 work process private      -    95   2   0   95
aaa55     2 work process private      -    44   2   0   44
6aa4d     f work shared library data    -    22   0   0   22

```

```

9aa53      f work shared library data      -   19   0   0   19
b8057      1 pers code,/dev/hd2:6230        -    8   0   -   -
7aa4f      - pers /dev/hd1:4164             -    1   0   -   -

```

```

.....
SHARED segments                Inuse   Pin   Pgps Virtual
                                3680    0     26   6199

```

```

Vsid      Esid Type Description                LPage  Inuse   Pin Pgps Virtual
b0036     d work shared library text          -   3625    0  26  6199
800f0     1 pers code,/dev/hd2:6251          -    54     0   -   -
f80df     - pers /dev/hd4:2                  -     1     0   -   -

```

Showing other processes that also use segments

The optional `-l` flag shows, for each displayed segment, the list of process identifiers that use the segment and the user the process belongs to. For special segments a label is displayed instead of the list of process identifiers. With the `-l` flag specified, each shared segment is followed by the list of process identifiers that use the segment. Besides the process identifier, the user who started it is also displayed, as shown in Example 24-16.

Example 24-16 `svmon -U user -l`

```

lpar05:/hennie/svmon>> svmon -U hennie -l
=====
User                Inuse   Pin   Pgps Virtual LPageCap
hennie              8147    2513  4278  14642      N
.....
SYSTEM segments    Inuse   Pin   Pgps Virtual
                    4327    2511  4252   8303
.....
Vsid      Esid Type Description                LPage  Inuse   Pin Pgps Virtual
0         0 work kernel seg                    -   4327    2511 4252  8303
.....
EXCLUSIVE segments Inuse   Pin   Pgps Virtual
                    118     2     0     117
.....
Vsid      Esid Type Description                LPage  Inuse   Pin Pgps Virtual
a2a54     2 work process private                -    95     2   0   95
6aa4d     f work shared library data            -    22     0   0   22
7aa4f     - pers /dev/hd1:4164                  -     1     0   -   -
.....
SHARED segments    Inuse   Pin   Pgps Virtual
                    3702    0     26   6222

```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3648	0	26	6222
			Shared library text segment					
800f0	1	pers	code,/dev/hd2:6251	-	54	0	-	-
			pid:36880 user: root					
			pid:35244 user: res1					
			pid:32664 user: res1					
			pid:29512 user: root					
			pid:28316 user: root					
			pid:25240 user: root					
			pid:22690 user: root					
			pid:22350 user: root					
			pid:20058 user: hennie					
			pid:15422 user: root					

Displaying the total number of virtual pages

The optional `-v` flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in virtual space (virtual pages include real memory and paging space pages). It is shown in Example 24-17 for the user called hennie.

Example 24-17 `svmon -U user -v`

```
lpar05:/hennie/svmon>> svmon -U hennie -v
```

```
=====
```

User	Inuse	Pin	Pgsp	Virtual	LPageCap
hennie	8147	2513	4278	14642	N

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	4327	2511	4252	8303

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4327	2511	4252	8303

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	118	2	0	117

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
a2a54	2	work	process private	-	95	2	0	95
6aa4d	f	work	shared library data	-	22	0	0	22
7aa4f	-	pers	/dev/hd1:4164	-	1	0	-	-

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	3702	0	26	6222

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3648	0	26	6222
800f0	1	pers	code,/dev/hd2:6251	-	54	0	-	-

Displaying total number of reserved paging space pages

The optional -g flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space, as shown in Example 24-18.

Example 24-18 `svmon -U user -g`

```
lpar05:/hennie/svmon>> svmon -U hennie -g
```

```
=====
```

User	Inuse	Pin	Pgsp	Virtual	LPageCap
hennie	8224	2515	4278	14699	N

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	4327	2511	4252	8303

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4327	2511	4252	8303

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	193	4	0	174

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2a56	2	work	process private	-	35	2	0	35
8aa51	f	work	shared library data	-	19	0	0	19
a2a54	2	work	process private	-	98	2	0	98
6aa4d	f	work	shared library data	-	22	0	0	22
7aa4f	-	pers	/dev/hd1:4164	-	1	0	-	-
90212	-	pers	/dev/hd2:8231	-	1	0	-	-
f1dfe	-	pers	/dev/hd2:401971	-	4	0	-	-
48349	-	pers	/dev/hd2:331860	-	1	0	-	-
9de1	-	pers	/dev/hd2:372747	-	1	0	-	-
11de2	-	pers	/dev/hd2:372748	-	1	0	-	-
19de3	-	pers	/dev/hd2:372749	-	1	0	-	-
21de4	-	pers	/dev/hd2:372750	-	1	0	-	-
b8057	1	pers	code,/dev/hd2:6230	-	8	0	-	-

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	3704	0	26	6222

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3648	0	26	6222
700ee	-	pers	/dev/hd2:2	-	1	0	-	-
800f0	1	pers	code,/dev/hd2:6251	-	54	0	-	-
f80df	-	pers	/dev/hd4:2	-	1	0	-	-

Displaying by total number of pinned pages

The optional `-p` flag indicates that the displayed information is sorted in decreasing order by the total number of pages pinned, as Example 24-19 shows.

Example 24-19 `svmon -U user -p`

```
lpar05:/hennie/svmon>> svmon -U hennie -p
```

```
=====
```

User	Inuse	Pin	Pgsp	Virtual	LPageCap
hennie	8215	2515	4278	14694	N

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	4327	2511	4252	8303

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4327	2511	4252	8303

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	184	4	0	169

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
a2a54	2	work	process private	-	98	2	0	98
8aa51	2	work	process private	-	30	2	0	30
696ed	-	pers	/dev/hd2:149613	-	1	0	-	-
f807f	-	pers	/dev/hd2:30737	-	1	0	-	-
7aa4f	-	pers	/dev/hd1:4164	-	1	0	-	-
b2a56	f	work	shared library data	-	19	0	0	19
6aa4d	f	work	shared library data	-	22	0	0	22
d96db	-	pers	/dev/hd2:92324	-	1	0	-	-
305a6	-	pers	/dev/hd2:61464	-	1	0	-	-
980b3	-	pers	/dev/hd2:2048	-	2	0	-	-
b8057	1	pers	code,/dev/hd2:6230	-	8	0	-	-

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	3704	0	26	6222

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
f80df	-	pers	/dev/hd4:2	-	1	0	-	-
700ee	-	pers	/dev/hd2:2	-	1	0	-	-

b0036	d work shared library text	-	3648	0	26	6222
800f0	1 pers code,/dev/hd2:6251	-	54	0	-	-

Displaying by total number of real memory pages

The optional -u flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in real memory, as shown in Example 24-20.

Example 24-20 `svmon -U user -u`

```
lpar05:/hennie/svmon>> svmon -U hennie -u
```

```
=====
User                               Inuse    Pin    Pgps  Virtual  LPageCap
hennie                             207777   2527   4278  214251   N
.....
SYSTEM segments                    Inuse    Pin    Pgps  Virtual
                                   4350    2519   4252   8326
.....
Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
0       0 work kernel seg             -    4327  2512 4252 8303
5a82b   - work                          -     23   7    0    23
.....
```

```
lpar05:/hennie/svmon>> svmon -U hennie -u
```

```
=====
User                               Inuse    Pin    Pgps  Virtual  LPageCap
hennie                             247424   2527   4278  253903   N
.....
SYSTEM segments                    Inuse    Pin    Pgps  Virtual
                                   4350    2519   4252   8326
.....
Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
0       0 work kernel seg             -    4327  2512 4252 8303
5a82b   - work                          -     23   7    0    23
.....
EXCLUSIVE segments                 Inuse    Pin    Pgps  Virtual
                                   239364   8     0    239355
.....
Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
b2856   4 work working storage        -    65536  0   0 65536
c2a58   5 work working storage        -    65536  0   0 65536
6284c   3 work working storage        -    65259  0   0 65259
105c2   6 work working storage        -    35759  0   0 35759
caa59   7 work working storage        -     6979  0   0 6979
```

```

2a60          f work shared library data      - 105  0  0 105
d15ba         2 work process private         -  44  2  0  44
a2a54         2 work process private         -  34  2  0  34
....(line omitted)....

```

```

.....
SHARED segments
                Inuse   Pin   Pgps Virtual
                3710    0    26   6222

```

```

Vsid   Esid Type Description          LPage Inuse   Pin Pgps Virtual
b0036   d work shared library text      - 3648    0  26 6222
800f0   1 pers code,/dev/hd2:6251       -  54     0  -  -
906d2   1 pers code,/dev/hd2:34861      -  7      0  -  -
f80df   - pers /dev/hd4:2                -  1      0  -  -

```

Displaying only client segments

The optional `-c` flag indicates that only client segments are to be included in the statistics. Note that Example 24-21 shows that the specified user does not use any client segments.

Example 24-21 `svmon -U user -c`

```

lpar05:/hennie/svmon>> svmon -U hennie -c
=====
User                Inuse   Pin   Pgps Virtual LPageCap
hennie              10      0     0     0       N
=====
EXCLUSIVE segments
                Inuse   Pin   Pgps Virtual
                10      0     0     0
=====
Vsid   Esid Type Description          LPage Inuse   Pin Pgps Virtual
32a86   - clnt
lpar05:/hennie/svmon>>

```

Example 24-22 shows a user reading files in a JFS2 filesystem.

Example 24-22 `svmon -U user -c`

```

lpar05:/>> svmon -U pieter -c
=====
User                Inuse   Pin   Pgps Virtual LPageCap
pieter             201024  0     0     0       N
=====
EXCLUSIVE segments
                Inuse   Pin   Pgps Virtual
                201024  0     0     0

```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
814f0	-	clnt		-	201024	0	-	-

Displaying only persistent segments

The optional `-f` flag indicates that only persistent segments (files) are to be included in the statistics, as shown in Example 24-23.

Example 24-23 `svmon -U user -f`

```
lpar05:/>> svmon -U gerda -f
```

```
=====
User                                Inuse    Pin    Pgsp  Virtual  LPageCap
gerda                                69       0      0      0         N
.....
EXCLUSIVE segments                 Inuse    Pin    Pgsp  Virtual
                                   13       0      0      0
.....
Vsid    Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
b8057   1 pers code,/dev/hd2:6230    -      8     0  -  -
a80f5   - pers /dev/hd2:4112         -      4     0  -  -
5856b   - pers /dev/hd1:2111         -      1     0  -  -
.....
SHARED segments                    Inuse    Pin    Pgsp  Virtual
                                   56       0      0      0
.....
Vsid    Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
800f0   1 pers code,/dev/hd2:6251    -     54     0  -  -
700ee   - pers /dev/hd2:2            -      1     0  -  -
f80df   - pers /dev/hd4:2            -      1     0  -  -
=====
```

Displaying only working segments

The optional `-w` flag indicates that only working segments are to be included in the statistics, as shown in Example 24-24.

Example 24-24 `svmon -U user -w`

```
lpar05:/>> svmon -U myra -w
```

```
=====
User                                Inuse    Pin    Pgsp  Virtual  LPageCap
myra                                7852    2514   4278   14395    N
.....
SYSTEM segments                    Inuse    Pin    Pgsp  Virtual
                                   4327    2510   4252    8303
=====
```



```

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
  0         0 work kernel seg              -   4327  2510 4252  8303

.....
EXCLUSIVE segments                Inuse   Pin    Pgsp  Virtual
                                171     4      0     171

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
82a90     2 work process private            -    95   2  0   95
b85b7     2 work process private            -    35   2  0   35
62a8c     f work shared library data        -    22   0  0   22
6c14d     f work shared library data        -    19   0  0   19

.....
SHARED segments                   Inuse   Pin    Pgsp  Virtual
                                3354    0     26   5921

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
b0036     d work shared library text        -  3354   0  26  5921
lpar05:/>>

```

Displaying only system segments

The optional `-s` flag indicates that only system segments are displayed in the output, as shown in Example 24-25.

Example 24-25 `svmon -U user -s`

```

lpar05:/>> svmon -U myra -s

=====
User      Inuse   Pin    Pgsp  Virtual  LPageCap
myra     4327   2510   4252  8303      N

.....
SYSTEM segments                Inuse   Pin    Pgsp  Virtual
                                4327   2510   4252  8303

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
  0         0 work kernel seg              -   4327  2510 4252  8303

```

Displaying only non-system segments

The optional `-n` flag indicates that only non-system segments are to be included in the statistics, as shown in Example 24-26 on page 412.

Example 24-26 svmon -U user -n

```
lpar05:/>> svmon -U bettie -n
```

```
=====
User                               Inuse    Pin    Pgps  Virtual  LPageCap
bettie                             3588     4      26    6088     N

.....
EXCLUSIVE segments                Inuse    Pin    Pgps  Virtual
                                   178      4      0     167

Vsid    Esid Type Description                LPage  Inuse  Pin Pgps Virtual
82a90   2 work process private                -      95    2   0   95
6c14d   2 work process private                -      31    2   0   31
62a8c   f work shared library data           -      22    0   0   22
e14dc   f work shared library data           -      19    0   0   19
b8057   1 pers code,/dev/hd2:6230            -       8     0   -   -
980b3   - pers /dev/hd2:2048                 -       2     0   -   -
b85b7   - pers /dev/hd1:2118                 -       1     0   -   -

.....
SHARED segments                   Inuse    Pin    Pgps  Virtual
                                   3410     0      26    5921

Vsid    Esid Type Description                LPage  Inuse  Pin Pgps Virtual
b0036   d work shared library text           -    3354   0   26  5921
800f0   1 pers code,/dev/hd2:6251            -      54    0   -   -
f80df   - pers /dev/hd4:2                    -       1     0   -   -
700ee   - pers /dev/hd2:2                    -       1     0   -   -
=====
```

Displaying allocated page ranges within segments

The optional `-r` flag displays the range(s) within the segment pages that have been allocated. A working segment may have two ranges because pages are allocated by starting from both ends and moving toward the middle. With the `-r` flag specified, each segment is followed by the range(s) within the segment where pages have been allocated, as shown in Example 24-27.

Example 24-27 svmon -U user -r

```
lpar05:/>> svmon -U bettie -r
```

```
=====
User                               Inuse    Pin    Pgps  Virtual  LPageCap
bettie                             7926    2514   4278  14395     N

.....
SYSTEM segments                   Inuse    Pin    Pgps  Virtual
                                   4327    2510   4252   8303
=====
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4327	2510	4252	8303
Addr Range: 0..27833								
.....								
EXCLUSIVE segments					Inuse	Pin	Pgsp	Virtual
					189	4	0	171
Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
82a90	2	work	process private	-	95	2	0	95
Addr Range: 0..151 : 65306..65535								
e14dc	2	work	process private	-	35	2	0	35
Addr Range: 0..431 : 65304..65535								
62a8c	f	work	shared library data	-	22	0	0	22
Addr Range: 0..3296								
6c14d	f	work	shared library data	-	19	0	0	19
Addr Range: 0..3296								
b8057	1	pers	code,/dev/hd2:6230	-	8	0	-	-
Addr Range: 0..7								
....(lines omitted)....								
.....								
SHARED segments					Inuse	Pin	Pgsp	Virtual
					3410	0	26	5921
Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3354	0	26	5921
Addr Range: 0..60123								
800f0	1	pers	code,/dev/hd2:6251	-	54	0	-	-
Addr Range: 0..55								
f80df	-	pers	/dev/hd4:2	-	1	0	-	-
Addr Range: 0..0								
700ee	-	pers	/dev/hd2:2	-	1	0	-	-
Addr Range: 0..0								

Analyzing processes reports

The `-P` flag can be used with `svmon` to monitor process memory utilization. In the following series of examples, `svmon` reports the memory usage for a process by using the different optional flags with the `-P` flag. Without any process specified, the `-P` option reports on all processes.

The column headings in a process report are:

- Pid Indicates the process ID.
- Command Indicates the command the process is running.

Inuse	Indicates the total number of pages in real memory in segments that are used by the process.
Pin	Indicates the total number of pages pinned in segments that are used by the process.
Pgsp	Indicates the total number of pages reserved or used on paging space by segments that are used by the process.
Virtual	Indicates the total number of pages allocated in the process virtual space.
64-bit	Indicates whether the process is a 64-bit process (Y) or a 32-bit process (N).
Mthrd	Indicates whether the process is multi-threaded (Y) or not (N).
Vsid	Indicates the virtual segment ID. Identifies a unique segment in the VMM.
Esid	Indicates the effective segment ID. The Esid is only valid when the segment belongs to the address space of the process. When provided, it indicates how the segment is used by the process. If the Vsid segment is mapped by several processes but with different Esid values, then this field contains '-'. In that case, the exact Esid values can be obtained through the -P flag applied on each of the process identifiers using the segment. A '-' also displays for segments used to manage open files or multi-threaded structures because these segments are not part of the user address space of the process.
Type	Identifies the type of the segment; pers indicates a persistent segment, work indicates a working segment, clnt indicates a client segment, map indicates a mapped segment and rmap indicates a real memory mapping segment.
Description	<p>Gives a textual description of the segment. The content of this column depends on the segment type and usage:</p> <p>persistent JFS files in the format <device>:<inode>, such as /dev/hd1:123.</p> <p>working Data areas of processes and shared memory segments dependent on the role of the segment based on the VSID and ESID.</p> <p>mapping Mapped to source segment IDs.</p> <p>client NFS, CD-ROM, and J2 files, dependent on the role of the segment based on the VSID and ESID.</p>

rmapping I/O space mapping dependent on the role of the segment based on the VSID and ESID.

Inuse	Indicates the number of pages in real memory in this segment.
Pin	Indicates the number of pages pinned in this segment.
Pgsp	Indicates the number of pages used on paging space by this segment. This field is relevant only for working segments.
Virtual	Indicates the number of pages allocated for the virtual space of the segment.

When process information is displayed, **svmon** displays information about all segments used by the process.

Displaying source segment and mapping segment information

The optional **-m** flag displays source segment and mapping segment information, as shown in Example 24-28.

Example 24-28 svmon -P pid -m

```
lpar05:/>> svmon -P 24832 -m
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	247169	2519	4278	253459	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	65536	0	0	65536
c2a58	5	work	working storage	-	65536	0	0	65536
6284c	3	work	working storage	-	65259	0	0	65259
105c2	6	work	working storage	-	35759	0	0	35759
caa59	7	work	working storage	-	6979	0	0	6979
0	0	work	kernel seg	-	4327	2510	4252	8303
b0036	d	work	shared library text	-	3360	0	26	5927
d14fa	-	pers	/dev/hd2:41157	-	246	0	-	-
2a60	f	work	shared library data	-	105	0	0	105
b14d6	2	work	process private	-	32	2	0	32
5a82b	-	work		-	23	7	0	23
906d2	1	pers	code,/dev/hd2:34861	-	7	0	-	-
10862	a	work	working storage	-	0	0	0	0
6158c	9	work	working storage	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
ba837	8	work	working storage	-	0	0	0	0

```
lpar05:/>>
```

Showing other processes that use segments

The optional -l flag shows, for each displayed segment, the list of process identifiers that use the segment. For special segments a label is displayed instead of the list of process identifiers, as shown in Example 24-29.

Example 24-29 `svmon -P pid -l`

```
lpar05:/>> svmon -P 24832 -l
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	246923	2519	4278	253459	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage pid(s)=24832	-	65536	0	0	65536
c2a58	5	work	working storage pid(s)=24832	-	65536	0	0	65536
6284c	3	work	working storage pid(s)=24832	-	65259	0	0	65259
105c2	6	work	working storage pid(s)=24832	-	35759	0	0	35759
caa59	7	work	working storage pid(s)=24832	-	6979	0	0	6979
0	0	work	kernel seg System segment	-	4327	2510	4252	8303
b0036	d	work	shared library text Shared library text segment	-	3360	0	26	5927
2a60	f	work	shared library data pid(s)=24832	-	105	0	0	105
b14d6	2	work	process private pid(s)=24832	-	32	2	0	32
5a82b	-	work	System segment	-	23	7	0	23
906d2	1	pers	code,/dev/hd2:34861 pid(s)=30752, 24832	-	7	0	-	-
6158c	9	work	working storage pid(s)=24832	-	0	0	0	0
10862	a	work	working storage pid(s)=24832	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa pid(s)=24832	-	0	0	-	-
ba837	8	work	working storage pid(s)=24832	-	0	0	0	0

Displaying by total number of virtual pages

The optional -v flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in virtual space (virtual pages include real memory and paging space pages), as shown in Example 24-30.

Example 24-30 `svmon -P 22674 -v`

```
lpar05:/>> svmon -P 24832 -v
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage	
24832	java	246923	2519	4278	253459	N	Y	N	

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual	
b2856	4	work	working storage	-	65536	0	0	65536	
c2a58	5	work	working storage	-	65536	0	0	65536	
6284c	3	work	working storage	-	65259	0	0	65259	
105c2	6	work	working storage	-	35759	0	0	35759	
caa59	7	work	working storage	-	6979	0	0	6979	
b0036	d	work	shared library text	-	3360	0	26	5927	
2a60	f	work	shared library data	-	105	0	0	105	
b14d6	2	work	process private	-	32	2	0	32	
5a82b	-	work		-	23	7	0	23	
10862	a	work	working storage	-	0	0	0	0	
6158c	9	work	working storage	-	0	0	0	0	
ba837	8	work	working storage	-	0	0	0	0	
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-	
906d2	1	pers	code,/dev/hd2:34861	-	7	0	-	-	

Displaying by total number of reserved paging space pages

The optional -g flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space, as shown in Example 24-31.

Example 24-31 `svmon -P 22674 -g`

```
lpar05:/>> svmon -P 24832 -g
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage	
24832	java	199600	2521	51504	253553	N	Y	N	

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual	
c2a58	5	work	working storage	-	51222	0	14328	65536	
6284c	3	work	working storage	-	51654	0	13632	65259	
b2856	4	work	working storage	-	52054	0	13491	65536	
105c2	6	work	working storage	-	31353	0	4407	35759	
caa59	7	work	working storage	-	5844	0	1135	6979	
b0036	d	work	shared library text	-	3227	0	28	6021	

2a60	f work shared library data	-	90	0	13	105
b14d6	2 work process private	-	30	2	2	32
5a82b	- work	-	21	7	2	23
10862	a work working storage	-	0	0	0	0
6158c	9 work working storage	-	0	0	0	0
ba837	8 work working storage	-	0	0	0	0
82810	b mmap mapped to sid d14fa	-	0	0	-	-
906d2	1 pers code,/dev/hd2:34861	-	6	0	-	-

Displaying by total number of pinned pages

The optional -p flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages pinned, shown in Example 24-32.

Example 24-32 svmon -P pid -p

```
lpar05:/>> svmon -P 24832 -p
```

```
-----
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	183009	2519	67854	253560	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
5a82b	-	work		-	21	7	2	23
b14d6	2	work	process private	-	29	2	3	32
caa59	7	work	working storage	-	5360	0	1619	6979
906d2	1	pers	code,/dev/hd2:34861	-	5	0	-	-
6284c	3	work	working storage	-	47001	0	18258	65259
10862	a	work	working storage	-	0	0	0	0
105c2	6	work	working storage	-	29581	0	6178	35759
c2a58	5	work	working storage	-	46573	0	18963	65536
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
2a60	f	work	shared library data	-	90	0	13	105
b0036	d	work	shared library text	-	3050	0	28	6028
6158c	9	work	working storage	-	0	0	0	0
b2856	4	work	working storage	-	47237	0	18299	65536
ba837	8	work	working storage	-	0	0	0	0

Displaying by total number of real memory pages

The optional -u flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in real memory, as shown in Example 24-33 on page 419.

Example 24-33 svmon -P pid -u

```
lpar05:/>> svmon -P 24832 -u
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	183020	2519	67854	253560	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
caa59	7	work	working storage	-	5360	0	1619	6979
b0036	d	work	shared library text	-	3055	0	28	6028
2a60	f	work	shared library data	-	90	0	13	105
b14d6	2	work	process private	-	29	2	3	32
5a82b	-	work		-	21	7	2	23
906d2	1	pers	code, /dev/hd2:34861	-	5	0	-	-
6158c	9	work	working storage	-	0	0	0	0
10862	a	work	working storage	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
ba837	8	work	working storage	-	0	0	0	0

Displaying only client segments

The optional `-c` flag indicates that only client segments are to be included in the statistics. Note that Example 24-34 shows that the specified process does not use any client segments:

Example 24-34 svmon -P pid -c

```
lpar05:/>> svmon -P 24832 -c
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	0	0	0	0	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
------	------	------	-------------	-------	-------	-----	------	---------

Displaying only persistent segments

The optional `-f` flag indicates that only persistent segments (files) are to be included in the statistics, as shown in Example 24-35.

Example 24-35 svmon -P pid -f

```
lpar05:/>> svmon -P 24832 -f
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	5	0	0	0	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
906d2	1	pers	code, /dev/hd2:34861	-	5	0	-	-

Displaying only working segments

The optional `-w` flag indicates that only working segments are to be included in the statistics, as shown in Example 24-36.

Example 24-36 `svmon -P pid -w`

```
lpar05:/>> svmon -P 24832 -w
```

```
-----
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	183017	2519	67854	253560	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
caa59	7	work	working storage	-	5360	0	1619	6979
b0036	d	work	shared library text	-	3057	0	28	6028
2a60	f	work	shared library data	-	90	0	13	105
b14d6	2	work	process private	-	29	2	3	32
5a82b	-	work		-	21	7	2	23
10862	a	work	working storage	-	0	0	0	0
6158c	9	work	working storage	-	0	0	0	0
ba837	8	work	working storage	-	0	0	0	0

Displaying only system segments

The optional `-s` flag indicates that only system segments are to be included in the statistics, as shown in Example 24-37.

Example 24-37 `svmon -P pid -s`

```
lpar05:/>> svmon -P 24832 -s
```

```
-----
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	4089	2517	4493	8326	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4068	2510	4491	8303
5a82b	-	work		-	21	7	2	23

Displaying only non-system segments

The optional `-n` flag indicates that only non-system segments are to be included in the statistics, as shown in Example 24-38.

Example 24-38 `svmon -P pid -n`

```
lpar05:/>> svmon -P 24832 -n
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	178933	2	63361	245234	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
caa59	7	work	working storage	-	5360	0	1619	6979
b0036	d	work	shared library text	-	3057	0	28	6028
2a60	f	work	shared library data	-	90	0	13	105
b14d6	2	work	process private	-	29	2	3	32
906d2	1	pers	code,/dev/hd2:34861	-	5	0	-	-
10862	a	work	working storage	-	0	0	0	0
6158c	9	work	working storage	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
ba837	8	work	working storage	-	0	0	0	0

Showing allocated page ranges within segments

The optional `-r` flag displays the range(s) within the segment pages that have been allocated (shown in Example 24-39). A working segment may have two ranges because pages are allocated by starting from both ends and moving toward the middle.

Example 24-39 `svmon -P pid -r`

```
lpar05:/>> svmon -P 24832 -r
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
24832	java	183022	2519	67854	253560	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
			Addr Range: 0..65375					
caa59	7	work	working storage	-	5360	0	1619	6979
			Addr Range: 0..12922					
b0036	d	work	shared library text	-	3057	0	28	6028

		Addr Range: 0..60123					
2a60	f	work shared library data	-	90	0	13	105
		Addr Range: 0..4572					
b14d6	2	work process private	-	29	2	3	32
		Addr Range: 65303..65535					
5a82b	-	work	-	21	7	2	23
		Addr Range: 0..49377					
906d2	1	pers code,/dev/hd2:34861	-	5	0	-	-
		Addr Range: 0..8					
6158c	9	work working storage	-	0	0	0	0
10862	a	work working storage	-	0	0	0	0
82810	b	mmap mapped to sid d14fa	-	0	0	-	-
ba837	8	work working storage	-	0	0	0	0

Analyzing the command reports

The `-C` flag can be used with `svmon` to monitor a command's memory utilization. The following series of examples shows how `svmon` reports the memory usage for commands by using the different optional flags with the `-C` flag.

The column headings in a command report are:

Command	Indicates the command name.
Inuse	Indicates the total number of pages in real memory in segments that are used by the command (for all processes running the command).
Pin	Indicates the total number of pages pinned in segments that are used by the command (for all processes running the command).
Pgsp	Indicates the total number of pages reserved or used on paging space by segments that are used by the command.
Virtual	Indicates the total number of pages allocated in the virtual space of the command.
Vsid	Indicates the virtual segment ID. Identifies a unique segment in the VMM.
Esid	Indicates the effective segment ID. The <code>Esid</code> is only valid when the segment belongs to the address space of the process. When provided, it indicates how the segment is used by the process. If the <code>Vsid</code> segment is mapped by several processes but with different <code>Esid</code> values, then this field contains '-'. In that case, the exact <code>Esid</code> values can be obtained through the <code>-P</code> flag applied on each of the process identifiers using the segment. A '-' also displays

	for segments used to manage open files or multi-threaded structures because these segments are not part of the user address space of the process.
Type	Identifies the type of the segment; pers indicates a persistent segment, work indicates a working segment, clnt indicates a client segment, map indicates a mapped segment, and rmap indicates a real memory mapping segment.
Description	<p>Gives a textual description of the segment. The content of this column depends on the segment type and usage:</p> <p>persistent JFS files in the format <device>:<inode>, such as /dev/hd1:123.</p> <p>working Data areas of processes and shared memory segments, dependent on the role of the segment based on the VSID and ESID.</p> <p>mapping Mapped to source segment IDs.</p> <p>client NFS, CD-ROM, and J2 files, dependent on the role of the segment based on the VSID and ESID.</p> <p>rmapping I/O space mapping dependent on the role of the segment based on the VSID and ESID.</p>
Inuse	Indicates the number of pages in real memory in this segment.
Pin	Indicates the number of pages pinned in this segment.
Pgsp	Indicates the number of pages used on paging space by this segment. This field is relevant only for working segments.
Virtual	Indicates the number of pages allocated for the virtual space of the segment.
The segments used by the command are separated into three categories:	
SYSTEM	Segments shared by all processes.
EXCLUSIVE	Segments used by the specified command (process).
SHARED	Segments shared by several commands (processes).
The global statistics for the specified command is the sum of each of the following fields; Inuse, Pin, Pgsp, and Virtual of the segment categories SYSTEM, EXCLUSIVE, and SHARED.	

Source segment and mapping segment

The optional -m flag displays source segment and mapping segment information when a segment is mapping a source segment, as shown in Example 24-40.

Example 24-40 `svmon -C command -m`

```
lpar05:/>> svmon -C java -m
```

```
=====
Command                               Inuse    Pin    Pgps  Virtual
java                                   362720  2528  127576 492789

.....
SYSTEM segments                       Inuse    Pin    Pgps  Virtual
                                       4112    2524  4493   8349

Vsid   Esid Type Description                LPage  Inuse  Pin Pgps Virtual
7a8af   - work -                -      23    7  0   23
5a82b   - work -                -      21    7  2   23

.....
EXCLUSIVE segments                   Inuse    Pin    Pgps  Virtual
                                       355550  4    123055 478412

Vsid   Esid Type Description                LPage  Inuse  Pin Pgps Virtual
328a6   3 work working storage                -    51472  0 13787 65259
128a2   4 work working storage                -    49778  0 15758 65536
b2856   4 work working storage                -    47237  0 18299 65536
6284c   3 work working storage                -    47001  0 18258 65259
5a8ab   5 work working storage                -    46866  0 18670 65536
c2a58   5 work working storage                -    46573  0 18963 65536
105c2   6 work working storage                -    29581  0 6178 35759
8a8b1   6 work working storage                -    25218  0 10541 35759
928b2   7 work working storage                -    6028  0 951 6979
caa59   7 work working storage                -    5360  0 1619 6979
d14fa   - pers /dev/hd2:41157                -    195   0  -  -
a8a1    f work shared library data            -    90   0 10 105
2a60    f work shared library data            -    90   0 13 105
b14d6   2 work process private                -    29   2 3 32
a28b4   2 work process private                -    27   2 5 32
906d2   1 pers code,/dev/hd2:34861            -    5   0  -  -
10862   a work working storage                -    0   0 0 0
82810   b mmap mapped to sid d14fa            -    0   0  -  -
528aa   a work working storage                -    0   0 0 0
6158c   9 work working storage                -    0   0 0 0
9a8b3   b mmap mapped to sid d14fa            -    0   0  -  -
4a8a9   8 work working storage                -    0   0 0 0
ba837   8 work working storage                -    0   0 0 0
828b0   9 work working storage                -    0   0 0 0
```

```

.....
SHARED segments
      Inuse      Pin      Pgps Virtual
      3058        0        28    6028

Vsid      Esid Type Description      LPage Inuse      Pin Pgps Virtual
b0036     d work shared library text      -   3057      0  28  6028
c85b9     - pers /dev/hd2:12302           -    1        0  -   -

```

All processes running a command

The `-d` flag reports information about all processes running the specified command, then `svmon` displays information about the segments used by those processes. This set of segments is separated into three categories: segments flagged *system* by the VMM, segments *only* used by the set of processes running the command, and segments *shared* between several command names, as shown in Example 24-41.

Example 24-41 svmon -C command -d

```
lpar05:/>> svmon -C java -d
```

```

=====
Command      Inuse      Pin      Pgps Virtual
java         362525     2528     127576  492789

```

```

-----
      Pid Command      Inuse      Pin      Pgps Virtual 64-bit Mthrd LPage
      30752 java        186633     2519     64241  253560      N      Y      N
      24832 java        183022     2519     67854  253560      N      Y      N

```

```

.....
SYSTEM segments
      Inuse      Pin      Pgps Virtual
      4112        2524     4493     8349

```

```

Vsid      Esid Type Description      LPage Inuse      Pin Pgps Virtual
7a8af     - work                    -    23        7  0  23
5a82b     - work                    -    21        7  2  23

```

```

.....
EXCLUSIVE segments
      Inuse      Pin      Pgps Virtual
      355355      4     123055  478412

```

```

Vsid      Esid Type Description      LPage Inuse      Pin Pgps Virtual
328a6     3 work working storage      -   51472      0 13787 65259
128a2     4 work working storage      -   49778      0 15758 65536
b2856     4 work working storage      -   47237      0 18299 65536
6284c     3 work working storage      -   47001      0 18258 65259
5a8ab     5 work working storage      -   46866      0 18670 65536
c2a58     5 work working storage      -   46573      0 18963 65536

```

105c2	6 work	working storage	-	29581	0	6178	35759
8a8b1	6 work	working storage	-	25218	0	10541	35759
928b2	7 work	working storage	-	6028	0	951	6979
caa59	7 work	working storage	-	5360	0	1619	6979
a8a1	f work	shared library data	-	90	0	10	105
2a60	f work	shared library data	-	90	0	13	105
b14d6	2 work	process private	-	29	2	3	32
a28b4	2 work	process private	-	27	2	5	32
906d2	1 pers	code,/dev/hd2:34861	-	5	0	-	-
4a8a9	8 work	working storage	-	0	0	0	0
82810	b mmap	mapped to sid d14fa	-	0	0	-	-
6158c	9 work	working storage	-	0	0	0	0
828b0	9 work	working storage	-	0	0	0	0
9a8b3	b mmap	mapped to sid d14fa	-	0	0	-	-
10862	a work	working storage	-	0	0	0	0
ba837	8 work	working storage	-	0	0	0	0
528aa	a work	working storage	-	0	0	0	0

```
.....
SHARED segments                Inuse   Pin   Pgsp  Virtual
                               3058   0     28   6028
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3057	0	28	6028
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-

Other processes also using segments

The optional `-l` flag shows, for each displayed segment, the list of process identifiers that use the segment and the command the process belongs to as shown in Example 24-42. For special segments, a label is displayed instead of the list of process identifiers. When the `-l` flag is specified, each segment in the last category is followed by the list of process identifiers that use the segment. Besides the process identifier, the command name it runs is also displayed.

Example 24-42 `svmon -C command -l`

```
lpar05:/>> svmon -C java -l
```

```
=====
Command                Inuse   Pin   Pgsp  Virtual
java                   362525  2528  127576 492789
```

```
.....
SYSTEM segments       Inuse   Pin   Pgsp  Virtual
                       4112   2524  4493  8349
```


Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
7a8af	-	work		-	23	7	0	23
5a82b	-	work		-	21	7	2	23

.....

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	355355	4	123055	478412

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
328a6	3	work	working storage	-	51472	0	13787	65259
128a2	4	work	working storage	-	49778	0	15758	65536
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
5a8ab	5	work	working storage	-	46866	0	18670	65536
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
8a8b1	6	work	working storage	-	25218	0	10541	35759
928b2	7	work	working storage	-	6028	0	951	6979
caa59	7	work	working storage	-	5360	0	1619	6979
a8a1	f	work	shared library data	-	90	0	10	105
2a60	f	work	shared library data	-	90	0	13	105
b14d6	2	work	process private	-	29	2	3	32
a28b4	2	work	process private	-	27	2	5	32
906d2	1	pers	code, /dev/hd2:34861	-	5	0	-	-
4a8a9	8	work	working storage	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
6158c	9	work	working storage	-	0	0	0	0
828b0	9	work	working storage	-	0	0	0	0
9a8b3	b	mmap	mapped to sid d14fa	-	0	0	-	-
10862	a	work	working storage	-	0	0	0	0
ba837	8	work	working storage	-	0	0	0	0
528aa	a	work	working storage	-	0	0	0	0

.....

SHARED segments	Inuse	Pin	Pgsp	Virtual
	3058	0	28	6028

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3057	0	28	6028
			Shared library text segment					
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-
			pid:35052 cmd: ksh					
			pid:30752 cmd: java					
			pid:15750 cmd: ksh					
			pid:15186 cmd: sleep					

Total number of virtual pages

The optional -v flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in virtual space (virtual pages include real memory and paging space pages), as shown in Example 24-43.

Example 24-43 `svmon -C command -v`

```
lpar05:/>> svmon -C java -v
```

```
=====
```

Command	Inuse	Pin	Pgsp	Virtual
java	362526	2528	127576	492789

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	4112	2524	4493	8349

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4068	2510	4491	8303
5a82b	-	work		-	21	7	2	23
7a8af	-	work		-	23	7	0	23

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	355355	4	123055	478412

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b2856	4	work	working storage	-	47237	0	18299	65536
128a2	4	work	working storage	-	49778	0	15758	65536
5a8ab	5	work	working storage	-	46866	0	18670	65536
c2a58	5	work	working storage	-	46573	0	18963	65536
328a6	3	work	working storage	-	51472	0	13787	65259
6284c	3	work	working storage	-	47001	0	18258	65259
8a8b1	6	work	working storage	-	25218	0	10541	35759
105c2	6	work	working storage	-	29581	0	6178	35759
caa59	7	work	working storage	-	5360	0	1619	6979
928b2	7	work	working storage	-	6028	0	951	6979
a8a1	f	work	shared library data	-	90	0	10	105
2a60	f	work	shared library data	-	90	0	13	105
b14d6	2	work	process private	-	29	2	3	32
a28b4	2	work	process private	-	27	2	5	32
4a8a9	8	work	working storage	-	0	0	0	0
10862	a	work	working storage	-	0	0	0	0
528aa	a	work	working storage	-	0	0	0	0
6158c	9	work	working storage	-	0	0	0	0
828b0	9	work	working storage	-	0	0	0	0
ba837	8	work	working storage	-	0	0	0	0
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
9a8b3	b	mmap	mapped to sid d14fa	-	0	0	-	-

```
906d2          1 pers code,/dev/hd2:34861          -      5      0      -      -
```

```
.....
SHARED segments          Inuse      Pin      Pgps  Virtual
                          3059      0        28    6028

Vsid      Esid Type Description          LPage  Inuse  Pin Pgps Virtual
b0036     d work shared library text      -    3058  0  28  6028
c85b9     - pers /dev/hd2:12302           -      1     0  -   -
lpar05:/>>
```

Total number of reserved paging space pages

The optional -g flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space, as shown in Example 24-44.

Example 24-44 svmon -C command -g

```
lpar05:/>> svmon -C java -g
```

```
=====
Command          Inuse      Pin      Pgps  Virtual
java             362526     2528    127576 492789

.....
SYSTEM segments          Inuse      Pin      Pgps  Virtual
                          4112     2524    4493   8349

Vsid      Esid Type Description          LPage  Inuse  Pin Pgps Virtual
0         0 work kernel seg              -    4068  2510 4491 8303
5a82b     - work                          -      21     7   2   23
7a8af     - work                          -      23     7   0   23

.....
EXCLUSIVE segments          Inuse      Pin      Pgps  Virtual
                          355355     4    123055 478412

Vsid      Esid Type Description          LPage  Inuse  Pin Pgps Virtual
c2a58     5 work working storage         -    46573  0 18963 65536
5a8ab     5 work working storage         -    46866  0 18670 65536
b2856     4 work working storage         -    47237  0 18299 65536
6284c     3 work working storage         -    47001  0 18258 65259
128a2     4 work working storage         -    49778  0 15758 65536
328a6     3 work working storage         -    51472  0 13787 65259
8a8b1     6 work working storage         -    25218  0 10541 35759
105c2     6 work working storage         -    29581  0  6178 35759
caa59     7 work working storage         -     5360  0  1619 6979
928b2     7 work working storage         -     6028  0   951 6979
```

2a60	f work shared library data	-	90	0	13	105
a8a1	f work shared library data	-	90	0	10	105
a28b4	2 work process private	-	27	2	5	32
b14d6	2 work process private	-	29	2	3	32
528aa	a work working storage	-	0	0	0	0
10862	a work working storage	-	0	0	0	0
ba837	8 work working storage	-	0	0	0	0
4a8a9	8 work working storage	-	0	0	0	0
6158c	9 work working storage	-	0	0	0	0
828b0	9 work working storage	-	0	0	0	0
82810	b mmap mapped to sid d14fa	-	0	0	-	-
9a8b3	b mmap mapped to sid d14fa	-	0	0	-	-
906d2	1 pers code,/dev/hd2:34861	-	5	0	-	-

```

.....
SHARED segments                Inuse      Pin      Pgps  Virtual
                               3059      0        28    6028

Vsid      Esid Type Description          LPage  Inuse   Pin Pgps Virtual
b0036    d work shared library text      -    3058   0  28  6028
c85b9    - pers /dev/hd2:12302          -      1     0   -   -
lpar05:/>>

```

Total number of pinned pages

The optional -p flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages pinned, as Example 24-45 shows.

Example 24-45 svmon -C command -p

```

lpar05:/>> svmon -C java -p

=====
Command                Inuse      Pin      Pgps  Virtual
java                   362570    2528    127576  492795

.....
SYSTEM segments                Inuse      Pin      Pgps  Virtual
                               4112    2524    4493    8349

Vsid      Esid Type Description          LPage  Inuse   Pin Pgps Virtual
0         0 work kernel seg              -    4068   2510 4491 8303
5a82b    - work                          -     21     7   2   23
7a8af    - work                          -     23     7   0   23

.....
EXCLUSIVE segments                Inuse      Pin      Pgps  Virtual
                               355355      4    123055  478412

```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b14d6	2	work	process private	-	29	2	3	32
a28b4	2	work	process private	-	27	2	5	32
906d2	1	pers	code,/dev/hd2:34861	-	5	0	-	-
c2a58	5	work	working storage	-	46573	0	18963	65536
328a6	3	work	working storage	-	51472	0	13787	65259
caa59	7	work	working storage	-	5360	0	1619	6979
4a8a9	8	work	working storage	-	0	0	0	0
6284c	3	work	working storage	-	47001	0	18258	65259
528aa	a	work	working storage	-	0	0	0	0
105c2	6	work	working storage	-	29581	0	6178	35759
10862	a	work	working storage	-	0	0	0	0
5a8ab	5	work	working storage	-	46866	0	18670	65536
a8a1	f	work	shared library data	-	90	0	10	105
82810	b	mmap	mapped to sid d14fa	-	0	0	-	-
2a60	f	work	shared library data	-	90	0	13	105
828b0	9	work	working storage	-	0	0	0	0
8a8b1	6	work	working storage	-	25218	0	10541	35759
928b2	7	work	working storage	-	6028	0	951	6979
6158c	9	work	working storage	-	0	0	0	0
9a8b3	b	mmap	mapped to sid d14fa	-	0	0	-	-
b2856	4	work	working storage	-	47237	0	18299	65536
ba837	8	work	working storage	-	0	0	0	0
128a2	4	work	working storage	-	49778	0	15758	65536

```
.....
SHARED segments                Inuse    Pin    Pgsp  Virtual
                               3103      0      28    6034
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-
b0036	d	work	shared library text	-	3102	0	28	6034

Total number of real memory pages

The optional -u flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in real memory, as shown in Example 24-46.

Example 24-46 svmon -C command -u

```
lpar05:/>> svmon -C java -u
```

```
=====
```

Command	Inuse	Pin	Pgsp	Virtual
java	362617	2528	127576	492795

```
.....
```

SYSTEM segments		Inuse	Pin	Pgsp	Virtual
		4113	2524	4493	8349

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	-	4069	2510	4491	8303
7a8af	-	work		-	23	7	0	23
5a82b	-	work		-	21	7	2	23

EXCLUSIVE segments		Inuse	Pin	Pgsp	Virtual
		355355	4	123055	478412

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
328a6	3	work	working storage	-	51472	0	13787	65259
128a2	4	work	working storage	-	49778	0	15758	65536
b2856	4	work	working storage	-	47237	0	18299	65536
6284c	3	work	working storage	-	47001	0	18258	65259
5a8ab	5	work	working storage	-	46866	0	18670	65536
c2a58	5	work	working storage	-	46573	0	18963	65536
105c2	6	work	working storage	-	29581	0	6178	35759
8a8b1	6	work	working storage	-	25218	0	10541	35759
928b2	7	work	working storage	-	6028	0	951	6979
caa59	7	work	working storage	-	5360	0	1619	6979
a8a1	f	work	shared library data	-	90	0	105	ba837
8	work	working storage		-	0	0	0	0
.....(lines omitted).....								
528aa	a	work	working storage	-	0	0	0	0

SHARED segments		Inuse	Pin	Pgsp	Virtual
		3149	0	28	6034

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d	work	shared library text	-	3148	0	28	6034
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-

Client segments only

The optional `-c` flag indicates that only client segments are to be included in the statistics. Example 24-47 shows that the specified process does not use any client segments.

Example 24-47 svmon -C command -c

```
lpar05:/>> svmon -C java -c
```

```
=====
```

Command	Inuse	Pin	Pgsp	Virtual
java	0	0	0	0

```
lpar05:/>>
```

Example 24-48 shows that a command is using client segments. From the following output we cannot know what kind of virtual file system it uses except that it is only used by this command at the time of the snapshot.

Example 24-48 svmon -C command -c

```
# svmon -c dd
```

```
=====
```

Command	Inuse	Pin	Pgsp	Virtual
dd	22808	0	0	0

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	22808	0	0	0

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
ce59	-	clnt		22808	0	-	-

Persistent segments only

The optional `-f` flag indicates that only persistent segments (files) are to be included in the statistics, as shown in Example 24-49.

Example 24-49 svmon -C command -f

```
lpar05:/>> svmon -C java -f
```

```
=====
```

Command	Inuse	Pin	Pgsp	Virtual
java	6	0	0	0

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	5	0	0	0

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
906d2	1	pers	code,/dev/hd2:34861	-	5	0	-	-

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	1	0	0	0

```
.....
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
c85b9	-	pers	/dev/hd2:12302	-	1	0	-	-

Working segments only

The optional `-w` flag indicates that only working segments are to be included in the statistics, as shown in Example 24-50 on page 434.

Example 24-50 svmon -C command -w

lpar05:/>> svmon -C java -w

```

=====
Command                               Inuse    Pin    Pgps  Virtual
java                                   362611  2528  127576 492795

.....
SYSTEM segments                       Inuse    Pin    Pgps  Virtual
                                       4113    2524  4493   8349

Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
   0      0 work kernel seg             -    4069  2510 4491 8303
  7a8af   - work                       -     23    7    0   23
  5a82b   - work                       -     21    7    2   23

.....
EXCLUSIVE segments                   Inuse    Pin    Pgps  Virtual
                                       355350   4  123055 478412

Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
328a6    3 work working storage           -  51472   0 13787 65259
128a2    4 work working storage           -  49778   0 15758 65536
b2856    4 work working storage           -  47237   0 18299 65536
6284c    3 work working storage           -  47001   0 18258 65259
5a8ab    5 work working storage           -  46866   0 18670 65536
c2a58    5 work working storage           -  46573   0 18963 65536
105c2    6 work working storage           -  29581   0  6178 35759
8a8b1    6 work working storage           -  25218   0 10541 35759
928b2    7 work working storage           -   6028   0   951 6979
caa59    7 work working storage           -   5360   0   1619 6979
a8a1     f work shared library data       -    90    0   10  105
2a60     f work shared library data       -    90    0   13  105
b14d6    2 work process private           -    29    2    3   32
a28b4    2 work process private           -    27    2    5   32
828b0    9 work working storage           -     0    0    0    0
528aa    a work working storage           -     0    0    0    0
6158c    9 work working storage           -     0    0    0    0
10862    a work working storage           -     0    0    0    0
ba837    8 work working storage           -     0    0    0    0
4a8a9    8 work working storage           -     0    0    0    0

.....
SHARED segments                      Inuse    Pin    Pgps  Virtual
                                       3148     0     28   6034

Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
b0036    d work shared library text       -  3148   0   28  6034
lpar05:/>>

```


System segments only

The optional -s flag indicates that only system segments are to be included in the statistics, as shown in Example 24-51.

Example 24-51 `svmon -C command -s`

```
lpar05:/>> svmon -C java -s
=====
Command                               Inuse      Pin      Pgps  Virtual
java                                   4113      2524    4493   8349

.....
SYSTEM segments                       Inuse      Pin      Pgps  Virtual
                                   4113      2524    4493   8349

Vsid      Esid Type Description          LPage  Inuse  Pin Pgps Virtual
    0          0 work kernel seg          -    4069  2510 4491  8303
7a8af      - work                               -     23    7    0    23
5a82b      - work                               -     21    7    2    23
lpar05:/>>
```

Non-system segments only

The optional -n flag indicates that only non-system segments are to be included in the statistics, as shown in Example 24-52.

Example 24-52 `svmon -C command -n`

```
lpar05:/>> svmon -C java -n
=====
Command                               Inuse      Pin      Pgps  Virtual
java                                   358504    4    123083  484446

.....
EXCLUSIVE segments                   Inuse      Pin      Pgps  Virtual
                                   355355    4    123055  478412

Vsid      Esid Type Description          LPage  Inuse  Pin Pgps Virtual
328a6      3 work working storage        -    51472  0  13787  65259
128a2      4 work working storage        -    49778  0  15758  65536
b2856      4 work working storage        -    47237  0  18299  65536
6284c      3 work working storage        -    47001  0  18258  65259
5a8ab      5 work working storage        -    46866  0  18670  65536
c2a58      5 work working storage        -    46573  0  18963  65536
105c2      6 work working storage        -    29581  0   6178  35759
8a8b1      6 work working storage        -    25218  0  10541  35759
928b2      7 work working storage        -     6028  0    951  6979
caa59      7 work working storage        -     5360  0   1619  6979
```

a8a1	f work shared library data	-	90	0	10	105
2a60	f work shared library data	-	90	0	13	105
b14d6	2 work process private	-	29	2	3	32
a28b4	2 work process private	-	27	2	5	32
906d2	1 pers code,/dev/hd2:34861	-	5	0	-	-
4a8a9	8 work working storage	-	0	0	0	0
82810	b mmap mapped to sid d14fa	-	0	0	-	-
6158c	9 work working storage	-	0	0	0	0
828b0	9 work working storage	-	0	0	0	0
9a8b3	b mmap mapped to sid d14fa	-	0	0	-	-
10862	a work working storage	-	0	0	0	0
ba837	8 work working storage	-	0	0	0	0
528aa	a work working storage	-	0	0	0	0

SHARED segments						
		Inuse	Pin	Pgsp	Virtual	
		3149	0	28	6034	
Vsid	Esid	Type	Description	LPage	Inuse	Pin Pgsp Virtual
b0036	d	work	shared library text	-	3148	0 28 6034
c85b9	-	pers	/dev/hd2:12302	-	1	0 - -

Allocated page ranges within segments

The optional `-r` flag displays the range(s) within the segment pages that have been allocated. A working segment may have two ranges because pages are allocated by starting from both ends and moving toward the middle. When the `-r` flag is specified, each segment is followed by the range(s) within the segment where the pages have been allocated, as shown in Example 24-53.

Example 24-53 svmon -C command -r

```
lpar05:/>> svmon -C java -r
```

Command						
java		Inuse	Pin	Pgsp	Virtual	
		362617	2528	127576	492795	
SYSTEM segments						
		Inuse	Pin	Pgsp	Virtual	
		4113	2524	4493	8349	
Vsid	Esid	Type	Description	LPage	Inuse	Pin Pgsp Virtual
	0	0	work kernel seg	-	4069	2510 4491 8303
			Addr Range: 0..27833			
7a8af	-	work		-	23	7 0 23
			Addr Range: 0..49377			
5a82b	-	work		-	21	7 2 23
			Addr Range: 0..49377			

.....						
EXCLUSIVE segments		Inuse	Pin	Pgsp	Virtual	
		355355	4	123055	478412	
Vsid	Esid Type Description	LPage	Inuse	Pin	Pgsp	Virtual
328a6	3 work working storage	-	51472	0	13787	65259
128a2	4 work working storage	-	49778	0	15758	65536
b2856	4 work working storage	-	47237	0	18299	65536
6284c	3 work working storage	-	47001	0	18258	65259
5a8ab	5 work working storage	-	46866	0	18670	65536
c2a58	5 work working storage	-	46573	0	18963	65536
105c2	6 work working storage	-	29581	0	6178	35759
	Addr Range: 0..65375					
8a8b1	6 work working storage	-	25218	0	10541	35759
	Addr Range: 0..65375					
928b2	7 work working storage	-	6028	0	951	6979
	Addr Range: 0..12922					
caa59	7 work working storage	-	5360	0	1619	6979
	Addr Range: 0..12922					
a8a1	f work shared library data	-	90	0	10	105
	Addr Range: 0..4572					
2a60	f work shared library data	-	90	0	13	105
	Addr Range: 0..4572					
b14d6	2 work process private	-	29	2	3	32
	Addr Range: 65303..65535					
a28b4	2 work process private	-	27	2	5	32
	Addr Range: 65303..65535					
906d2	1 pers code,/dev/hd2:34861	-	5	0	-	-
	Addr Range: 0..8					
4a8a9	8 work working storage	-	0	0	0	0
82810	b mmap mapped to sid d14fa	-	0	0	-	-
6158c	9 work working storage	-	0	0	0	0
828b0	9 work working storage	-	0	0	0	0
9a8b3	b mmap mapped to sid d14fa	-	0	0	-	-
10862	a work working storage	-	0	0	0	0
ba837	8 work working storage	-	0	0	0	0
528aa	a work working storage	-	0	0	0	0
.....						
SHARED segments		Inuse	Pin	Pgsp	Virtual	
		3149	0	28	6034	
Vsid	Esid Type Description	LPage	Inuse	Pin	Pgsp	Virtual
b0036	d work shared library text	-	3148	0	28	6034
	Addr Range: 0..60123					
c85b9	- pers /dev/hd2:12302	-	1	0	-	-
	Addr Range: 0..0					

Analyzing segment utilization

The `-S` flag can be used with `svmon` to monitor segment utilization. The following series of examples shows how `svmon` reports the memory usage for a process by using the different optional flags with the `-S` flag.

The column headings in a segment report are:

Vsid	Indicates the virtual segment ID. Identifies a unique segment in the VMM.										
Esid	Indicates the effective segment ID. The Esid is only valid when the segment belongs to the address space of the process. When provided, it indicates how the segment is used by the process. If the Vsid segment is mapped by several processes but with different Esid values, then this field contains '-'. In that case, the exact Esid values can be obtained through the <code>-P</code> flag applied on each of the process identifiers using the segment. A '-' also displays for segments used to manage open files or multi-threaded structures because these segments are not part of the user address space of the process.										
Type	Identifies the type of the segment: <code>pers</code> indicates a persistent segment, <code>work</code> indicates a working segment, <code>clnt</code> indicates a client segment, <code>map</code> indicates a mapped segment, and <code>rmap</code> indicates a real memory mapping segment.										
Description	Gives a textual description of the segment. The content of this column depends on the segment type and usage: <table><tr><td><code>persistent</code></td><td>JFS files in the format <code><device>:<inode></code>, such as <code>/dev/hd1:123</code>.</td></tr><tr><td><code>working</code></td><td>Data areas of processes and shared memory segments, dependent on the role of the segment based on the VSID and ESID.</td></tr><tr><td><code>mapping</code></td><td>Mapped to source segment IDs.</td></tr><tr><td><code>client</code></td><td>NFS, CD-ROM, and J2 files, dependent on the role of the segment based on the VSID and ESID.</td></tr><tr><td><code>rmapping</code></td><td>I/O space mapping dependent on the role of the segment based on the VSID and ESID.</td></tr></table>	<code>persistent</code>	JFS files in the format <code><device>:<inode></code> , such as <code>/dev/hd1:123</code> .	<code>working</code>	Data areas of processes and shared memory segments, dependent on the role of the segment based on the VSID and ESID.	<code>mapping</code>	Mapped to source segment IDs.	<code>client</code>	NFS, CD-ROM, and J2 files, dependent on the role of the segment based on the VSID and ESID.	<code>rmapping</code>	I/O space mapping dependent on the role of the segment based on the VSID and ESID.
<code>persistent</code>	JFS files in the format <code><device>:<inode></code> , such as <code>/dev/hd1:123</code> .										
<code>working</code>	Data areas of processes and shared memory segments, dependent on the role of the segment based on the VSID and ESID.										
<code>mapping</code>	Mapped to source segment IDs.										
<code>client</code>	NFS, CD-ROM, and J2 files, dependent on the role of the segment based on the VSID and ESID.										
<code>rmapping</code>	I/O space mapping dependent on the role of the segment based on the VSID and ESID.										
Inuse	Indicates the number of pages in real memory in this segment.										
Pin	Indicates the number of pages pinned in this segment.										
Pgsp	Indicates the number of pages used on paging space by this segment. This field is relevant only for working segments.										

Virtual Indicates the number of pages allocated for the virtual space of the segment.

Without any segment specification the -S option reports on all segments as shown in Example 24-54.

Example 24-54 svmon -S

```
lpar05:/>> svmon -S
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
6aa8d	-	pers	/dev/lv01:17	-	166408	0	-	-
814f0	-	clnt		-	150780	0	-	-
30246	-	work		-	65529	65529	0	65529
4aa49	-	clnt		-	60354	0	-	-
328a6	-	work		-	51472	0	13787	65259
128a2	-	work		-	49778	0	15758	65536
b2856	-	work		-	47237	0	18299	65536
6284c	-	work		-	47001	0	18258	65259
5a8ab	-	work		-	46866	0	18670	65536
c2a58	-	work		-	46573	0	18963	65536
80010	-	work	page frame table	-	30481	30480	0	30481
105c2	-	work		-	29581	0	6178	35759
90012	-	work	kernel pinned heap	-	25409	9001	53920	60396
8a8b1	-	work		-	25218	0	10541	35759
82b50	-	pers	/dev/lv01:305233	-	21413	0	-	-

.....(Lines omitted).....

Source segment and mapping segment

The optional -m flag displays source segment and mapping segment information, as shown in Example 24-55.

Example 24-55 svmon -S -m

```
# svmon -Sm
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
6aa8d	-	pers	/dev/lv01:17	-	166408	0	-	-
814f0	-	clnt		-	150780	0	-	-
30246	-	work		-	65529	65529	0	65529
4aa49	-	clnt		-	60354	0	-	-
328a6	-	work		-	51472	0	13787	65259
128a2	-	work		-	49778	0	15758	65536
b2856	-	work		-	47237	0	18299	65536
6284c	-	work		-	47001	0	18258	65259
5a8ab	-	work		-	46866	0	18670	65536
c2a58	-	work		-	46573	0	18963	65536
80010	-	work	page frame table	-	30481	30480	0	30481
105c2	-	work		-	29581	0	6178	35759
90012	-	work	kernel pinned heap	-	25434	9001	53920	60396
8a8b1	-	work		-	25218	0	10541	35759

```

82b50      - pers /dev/lv01:305233      - 21413  0  -  -
8ab51      - clnt                               - 20577  0  -  -
92b32      - pers /dev/lv01:305218          - 11704  0  -  -
9ab33      - clnt                               - 10887  0  -  -
cb6b9      - clnt                               -  8862  0  -  -
2b20      - pers /dev/lv01:305209          -  8241  0  -  -
ab21      - clnt                               -  7750  0  -  -
..(lines omitted)...

```

Other processes also using segments

The optional `-l` flag shows, for each displayed segment, the list of process identifiers that use the segment, as shown in Example 24-56. For special segments a label is displayed instead of the list of process identifiers.

Example 24-56 `svmon -S -l`

```

# svmon -S1

Vsid      Esid Type Description                LPage  Inuse  Pin Pgsp Virtual
6aa8d     - pers /dev/lv01:17
          Unused segment
814f0     - clnt
          Unused segment
30246     - work
          Unused segment
4aa49     - clnt
          Unused segment
328a6     3 work working storage
          pid(s)=30752
128a2     4 work working storage
          pid(s)=30752
b2856     4 work working storage
          pid(s)=24832
6284c     3 work working storage
          pid(s)=24832
5a8ab     5 work working storage
          pid(s)=30752
c2a58     5 work working storage
          pid(s)=24832
80010     - work page frame table
          System segment

```

Total number of virtual pages

The optional `-v` flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in virtual space (virtual pages include real memory and paging space pages), as shown in Example 24-57 on page 441.

Example 24-57 svmon -S -v

```
# svmon -Sv
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
5a8ab	-	work		-	46866	0	18670	65536
b2856	-	work		-	47237	0	18299	65536
128a2	-	work		-	49778	0	15758	65536
c2a58	-	work		-	46573	0	18963	65536
30246	-	work		-	65529	65529	0	65529
6284c	-	work		-	47001	0	18258	65259
328a6	-	work		-	51472	0	13787	65259
90012	-	work	kernel pinned heap	-	25446	9001	53920	60396
105c2	-	work		-	29581	0	6178	35759
8a8b1	-	work		-	25218	0	10541	35759
80010	-	work	page frame table	-	30481	30480	0	30481
88011	-	work	misc kernel tables	-	7238	0	5568	10839
5004a	-	work		-	5605	5605	3309	8914

.....(lines omitted).....

Total number of reserved paging space pages

The optional -g flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space, as shown in Example 24-58.

Example 24-58 svmon -S -g

```
lpar05:/>> svmon -Sg
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
90012	-	work	kernel pinned heap	-	25452	9005	53920	60396
c2a58	-	work		-	46573	0	18963	65536
5a8ab	-	work		-	46866	0	18670	65536
b2856	-	work		-	47237	0	18299	65536
6284c	-	work		-	47001	0	18258	65259
128a2	-	work		-	49778	0	15758	65536
328a6	-	work		-	51472	0	13787	65259
8a8b1	-	work		-	25218	0	10541	35759
105c2	-	work		-	29581	0	6178	35759
88011	-	work	misc kernel tables	-	7271	0	5568	10871
5004a	-	work		-	5605	5605	3309	8914
caa59	-	work		-	5360	0	1619	6979
b8037	-	work		-	472	0	1014	1430
928b2	-	work		-	6028	0	951	6979
7800f	-	work	page table area	-	772	2	784	786
582cb	-	work		-	0	0	653	653
b8437	-	work		-	266	2	614	748
88391	-	work		-	81	2	525	536
420	-	work		-	8	2	482	487

```

c0478      - work      -      14      2 393  398
904f2      - work      -       9      2 381  387
....(lines omitted).....

```

Total number of pinned pages

The optional -p flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages pinned, as Example 24-59 shows.

Example 24-59 svmon -S -p

```

# svmon -Sp

Vsid      Esid Type Description      LPage  Inuse  Pin Pgps Virtual
30246     - work      - 65529 65529  0 65529
80010     - work page frame table - 30481 30480  0 30481
90012     - work kernel pinned heap - 25457 9009 53920 60396
5004a     - work      - 5605 5605 3309 8914
28005     - work software hat - 1024 1024  0 1024
8001      - work segment table - 602 600  0 602
20004     - work kernel ext seg - 40 39  0 40
18003     - work page space disk map - 37 37  0 37
28365     - work      - 16 8 15 25
104e2     - work      - 11 7 13 22
...(lines omitted)...

```

Total number of real memory pages

The optional -u flag indicates that the information to be displayed is sorted in decreasing order by the total number of pages in real memory, as shown in Example 24-60.

Example 24-60 svmon -S -u

```

# svmon -Su

Vsid      Esid Type Description      LPage  Inuse  Pin Pgps Virtual
6aa8d     - pers /dev/lv01:17 - 166408 0 - -
814f0     - clnt      - 150780 0 - -
30246     - work      - 65529 65529  0 65529
4aa49     - clnt      - 60354 0 - -
328a6     - work      - 51472 0 13787 65259
128a2     - work      - 49778 0 15758 65536
b2856     - work      - 47237 0 18299 65536
6284c     - work      - 47001 0 18258 65259
5a8ab     - work      - 46866 0 18670 65536
c2a58     - work      - 46573 0 18963 65536
80010     - work page frame table - 30481 30480  0 30481
105c2     - work      - 29581 0 6178 35759
90012     - work kernel pinned heap - 25457 9001 53920 60396
8a8b1     - work      - 25218 0 10541 35759

```



```
82b50          - pers /dev/lv01:305233          - 21413    0    -    -
.....(lines omitted).....
```

Client segments only

The optional `-c` flag indicates that only client segments are to be included in the statistics. Note that client segments are not paged to paging space when the frames they occupy are needed for another use, hence the dash (-) in the `Pgsp` and `Virtual` columns, as shown in Example 24-61.

Example 24-61 `svmon -S -c`

```
# svmon -Sc
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
814f0	-	clnt		-	150780	0	-	-
4aa49	-	clnt		-	60354	0	-	-
8ab51	-	clnt		-	20577	0	-	-
9ab33	-	clnt		-	10887	0	-	-
cb6b9	-	clnt		-	8862	0	-	-

Persistent segments only

The optional `-f` flag indicates that only persistent segments (files) are to be included in the statistics. Note that persistent segments are not paged to paging space when the frames they occupy are needed for another use, hence the dash (-) in the `Pgsp` and `Virtual` columns as shown in Example 24-62.

Example 24-62 `svmon -S -f`

```
# svmon -Sf
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
6aa8d	-	pers	/dev/lv01:17	-	166408	0	-	-
82b50	-	pers	/dev/lv01:305233	-	21413	0	-	-
92b32	-	pers	/dev/lv01:305218	-	11704	0	-	-
2b20	-	pers	/dev/lv01:305209	-	8241	0	-	-
b2b36	-	pers	/dev/lv01:305220	-	6483	0	-	-
52b2a	-	pers	/dev/lv01:305214	-	5958	0	-	-
42e48	-	pers	/dev/lv01:188427	-	5668	0	-	-
a2b34	-	pers	/dev/lv01:305219	-	5636	0	-	-
12b42	-	pers	/dev/lv01:305226	-	5396	0	-	-

```
.....(lines omitted).....
```

Working segments only

The optional `-w` flag indicates that only working segments are to be included in the statistics. Note that working segments *are* paged to paging space when the frames they occupy are needed for another use, as shown in Example 24-63 on page 444.

Example 24-63 svmon -S -w

```
# svmon -Sw
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
30246		- work		-	65529	65529	0	65529
328a6		- work		-	51472	0	13787	65259
128a2		- work		-	49778	0	15758	65536
b2856		- work		-	47237	0	18299	65536
6284c		- work		-	47001	0	18258	65259
5a8ab		- work		-	46866	0	18670	65536
c2a58		- work		-	46573	0	18963	65536
80010		- work	page frame table	-	30481	30480	0	30481
105c2		- work		-	29581	0	6178	35759
90012		- work	kernel pinned heap	-	25462	9009	53920	60396
8a8b1		- work		-	25218	0	10541	35759
88011		- work	misc kernel tables	-	7316	0	5568	10903
928b2		- work		-	6028	0	951	6979
5004a		- work		-	5605	5605	3309	8914
caa59		- work		-	5360	0	1619	6979

...(lines omitted)....

System segments only

The optional `-s` flag indicates that only system segments are to be included in the statistics. Note that system segments can be paged to paging space when the frames they occupy are needed for another use, but the part that is pinned (Pin) will not. In Example 24-64 you see that the kernel page frame table cannot be paged out because its frame usage equals the pinned size (1792 and 1792).

Example 24-64 svmon -S -s

```
# svmon -Ss
```

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
80010		- work	page frame table	-	30481	30480	0	30481
90012		- work	kernel pinned heap	-	25468	9001	53920	60396
88011		- work	misc kernel tables	-	7318	0	5568	10903
5004a		- work		-	5605	5605	3309	8914
	0	- work	kernel seg	-	4069	2510	4491	8303
400e8		- pers	/dev/hd2:3	-	1731	0	-	-
28005		- work	software hat	-	1024	1024	0	1024
7800f		- work	page table area	-	772	2	784	786
ca839		- pers	/dev/lv01:4	-	721	0	-	-

.....(lines omitted).....

Non-system segments only

The optional `-n` flag indicates that only non-system segments are to be included in the statistics as shown in Example 24-65 on page 445.

Example 24-65 svmon -S -n

```
# svmon -Sn

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
6aa8d     - pers /dev/lv01:17          - 166408  0 - -
814f0     - clnt                               - 150780  0 - -
30246     - work                               - 65529 65529  0 65529
4aa49     - clnt                               - 60354  0 - -
328a6     - work                               - 51472  0 13787 65259
128a2     - work                               - 49778  0 15758 65536
b2856     - work                               - 47237  0 18299 65536
6284c     - work                               - 47001  0 18258 65259
5a8ab     - work                               - 46866  0 18670 65536
c2a58     - work                               - 46573  0 18963 65536
105c2     - work                               - 29581  0 6178 35759
8a8b1     - work                               - 25218  0 10541 35759
82b50     - pers /dev/lv01:305233      - 21413  0 - -
.....(lines omitted).....
```

Allocated page ranges within segments

The optional `-r` flag displays the range(s) within the segment pages that have been allocated. A working segment may have two ranges because pages are allocated by starting from both ends and moving toward the middle, as shown in Example 24-66.

Example 24-66 svmon -S segment -r

```
# svmon -Sr

Vsid      Esid Type Description          LPage  Inuse  Pin Pgsp Virtual
6aa8d     - pers /dev/lv01:17          - 166408  0 - -
          Addr Range: 0..229232
814f0     - clnt                               - 150780  0 - -
          Addr Range: 0..229232
30246     - work                               - 65529 65529  0 65529
          Addr Range: 0..65528
4aa49     - clnt                               - 60354  0 - -
          Addr Range: 0..86655
328a6     - work                               - 51472  0 13787 65259
128a2     - work                               - 49778  0 15758 65536
b2856     - work                               - 47237  0 18299 65536
6284c     - work                               - 47001  0 18258 65259
5a8ab     - work                               - 46866  0 18670 65536
c2a58     - work                               - 46573  0 18963 65536
80010     - work page frame table          - 30481 30480  0 30481
          Addr Range: 0..30719
105c2     - work                               - 29581  0 6178 35759
          Addr Range: 0..65375
```

```

90012      - work kernel pinned heap      - 25480  9009 53920 60396
8a8b1      - work                               - 25218   0 10541 35759
           Addr Range: 0..65375
82b50      - pers /dev/lv01:305233        - 21413   0   -
.....(lines omitted).....

```

Analyzing detailed reports

The -D flag can be used with **svmon** to monitor detailed utilization. The following series of examples shows how **svmon** reports the memory usage for a process by using the different optional flags with the -D flag. Because the detailed report shows all frames used by a segment, the output usually will be quite extensive. The information shown for each frame comes from examining the Page Frame Table (PFT) for the segment frames and reporting the same information that the *lrud* kernel process would use when the number of free pages is lower than the kernel minfree value. See Chapter 14, “The vmo, ioo, and vmtune commands” on page 229 for more detail.

The column headings in a detailed report are:

Segid Indicates the virtual segment ID. Identifies a unique segment in the VMM.

Type Identifies the type of the segment: *pers* indicates a persistent segment, *work* indicates a working segment, *clnt* indicates a client segment, *map* indicates a mapped segment, and *rmap* indicates a real memory mapping segment.

LPage Indicates whether the segment uses large pages.

Size of page space allocation

Indicates the number of pages used on paging space by this segment. This field is relevant only for working segments.

Virtual Indicates the number of pages allocated for the virtual space of the segment.

Inuse Indicates the number of pages in real memory in this segment.

Page Page number relative to the virtual space. This page number can be higher than the number of frames in a segment (65535) if the virtual space is larger than a single segment (large file).

Frame Frame number in the real memory.

Pin Indicates whether the frame is pinned.

Ref Indicates whether the frame has been referenced by a process (-b flag only).

Mod Indicates whether the frame has been modified by a process (-b flag only).

ExtSegid Extended segment identifier: This field is set only when the page number is higher than the maximum number of frames in a segment.

ExtPage Extended page number: This field is set only when the page number is higher than the maximum number of frames in a segment and indicates the page number within the extended segment.

Segment details

Example 24-67 monitors details of segment 6aa8d showing the status of the reference and modified bits of all displayed frames.

Example 24-67 svmon -D segment

```
#svmon -D 6aa8d
```

```
Segid: 6aa8d
```

```
Type: persistent
```

```
LPage: N
```

```
Address Range: 0..229232
```

Page	Frame	Pin	ExtSegid	ExtPage
0	859753	N	-	-
1	822083	N	-	-
2	375590	N	-	-
3	903339	N	-	-
4	859755	N	-	-
5	919616	N	-	-
6	859761	N	-	-
7	586516	N	-	-
8	859769	N	-	-
9	586460	N	-	-
10	859771	N	-	-
12	859775	N	-	-
13	472106	N	-	-
14	859779	N	-	-
15	2025902	N	-	-
16	859793	N	-	-
17	466674	N	-	-

```
....(pages omitted)....
```

Monitoring segment details during a time interval

Example 24-68 on page 448 monitors details of segment 9012 showing the status of the reference and modified bits of all of the displayed frames that are accessed between each interval. Once shown, the reference bit of the frame is reset.

Example 24-68 svmon -D segment -b -i

```
# svmon -D 9012 -b -i 5 3
```

```
Segid: 9012
Type: working
LPage: N
Address Range: 65338..65535
Size of page space allocation: 0 pages ( 0.0 Mb)
Virtual: 3 frames ( 0.0 Mb)
Inuse: 3 frames ( 0.0 Mb)
```

Page	Frame	Pin	Ref	Mod	ExtSegid	ExtPage
65339	771	Y	Y	Y	-	-
65340	770	Y	Y	Y	-	-
65338	4438	N	Y	Y	-	-

```
Segid: 9012
Type: working
LPage: N
Address Range: 65338..65535
Size of page space allocation: 0 pages ( 0.0 Mb)
Virtual: 3 frames ( 0.0 Mb)
Inuse: 3 frames ( 0.0 Mb)
```

Page	Frame	Pin	Ref	Mod	ExtSegid	ExtPage
65339	771	Y	Y	Y	-	-
65340	770	Y	Y	Y	-	-
65338	4438	N	Y	Y	-	-

```
Segid: 9012
Type: working
LPage: N
Address Range: 65338..65535
Size of page space allocation: 0 pages ( 0.0 Mb)
Virtual: 3 frames ( 0.0 Mb)
Inuse: 3 frames ( 0.0 Mb)
```

Page	Frame	Pin	Ref	Mod	ExtSegid	ExtPage
65339	771	Y	Y	Y	-	-
65340	770	Y	Y	Y	-	-
65338	4438	N	Y	Y	-	-

Analyzing frame reports

The `-F` flag can be used with `svmon` to monitor frame utilization. Example 24-69 on page 449 shows the use of the `-F` flag with no argument specified. The frame report returns the percentage of real memory used in the system with the reference flag set.

Example 24-69 svmon -F

```
# svmon -F
Processing.. 100%
percentage of memory used: 85.82%
```

Example 24-70 shows a comparison of the -F to -G output.

Example 24-70 Comparing -F and -G output

```
# svmon -G
      size      inuse      free      pin      virtual
memory  131047   27675   103372   13734    38228
pg space 262144     14452
      work      pers      clnt
pin      13734        0         0
in use   21252        815       5608

# svmon -F
Processing.. 100%
percentage of memory used: 88.85%
```

The used memory percentage from the -G output is 21.18 percent used (27675 / 131047) because the percentage of memory used in the -F report refers only to frames with the reference flag set; that is, the sum of pages that are not eligible to be released if the page stealer (*lrud* kproc) needs to allocate the frames for other use. The -G report shows all memory that processes have allocated, either by themselves or by the VMM for their use (such as shared libraries that needed to be loaded and linked dynamically to the process in order for the process to be loaded properly).

Specifying frame numbers

When frame numbers are specified with the -F option, the column headings in the report are:

Frame	Frame number in real memory.
Segid	Indicates the virtual segment ID that the frame belongs to.
Ref	Indicates whether the frame has been referenced by a process.
Mod	Indicates whether the frame has been modified by a process.
Pincount	Indicates the long-term pincount and the short-term pincount for the frame.
State	Indicates the state of the frame (Bad, In-Use, Free, I/O, PgAhead, Hidden).
Swbits	Indicates the status of the frame in the Software Page Frame Table.

ExtSegid Extended segment ID: This field will only be set when it belongs to an extended segment.

LPage Indicates whether this frame belongs to a large page segment.

The information shown for the frame(s) comes from examining the Page Frame Table (PFT) and reporting the same information that the lru kernel process would use when the number of free pages is lower than the kernel minfree value. (See Chapter 14, “The vmo, ioo, and vmtune commands” on page 229 for more about minfree.) In Example 24-71 we specify a frame to monitor, in this case frame 815.

Example 24-71 svmon -F frame

```
lpar05:/>> svmon -F 2096915
```

Frame	Segid	Ref	Mod	Pincount	State	Swbits	ExtSegid	LPage
2096915	7804f	Y	Y	1/0	Hidden	88000000	-	N

```
lpar05:/>>
```

Monitoring frames during a time interval

To monitor a frame over a specified interval, use the `-i` flag. In Example 24-72, we monitor a frame on five-second intervals repeated three times.

Example 24-72 svmon -F frame -i

```
lpar05:/>> svmon -F 2096915 -i 2 3
```

Frame	Segid	Ref	Mod	Pincount	State	Swbits	ExtSegid	LPage
2096915	7804f	Y	Y	1/0	Hidden	88000000	-	N

Frame	Segid	Ref	Mod	Pincount	State	Swbits	ExtSegid	LPage
2096915	7804f	Y	Y	1/0	Hidden	88000000	-	N

Frame	Segid	Ref	Mod	Pincount	State	Swbits	ExtSegid	LPage
2096915	7804f	Y	Y	1/0	Hidden	88000000	-	N

```
lpar05:/>>
```

This example shows that frame 2096915 is both referenced and modified during the time the trace was run.

Monitoring frame reuse between processes

The following sample starts by using the `dd` command to read the same file from the JFS2 file system three times in a row. First we use the command `report (-C)` to find out what segments the `dd` command is using, as shown in Example 24-73 on page 451.

Example 24-73 svmon -cC dd

```
lpar05:/>> svmon -cC dd

=====
Command                Inuse    Pin    Pgps  Virtual
dd                      106638   0      0     0

.....
EXCLUSIVE segments      Inuse    Pin    Pgps  Virtual
                        106638   0      0     0

Vsid    Esid Type Description          LPage  Inuse  Pin Pgps Virtual
4aa49   -  clnt                    -   69618  0  -  -
d417a   -  clnt                    -   37020  0  -  -
lpar05:/>>
```

The output shows that segment 4aa49 is a client segment (used for the JFS2 file system file pages). Example 24-74 shows how we use the virtual segment ID (Vsid) to see what frames this segment has allocated with the detailed report using the -D flag.

Example 24-74 svmon -D d65a

```
lpar05:/>> svmon -D 4aa49

Segid: 4aa49
Type: client
LPage: N
Address Range: 0..86655

Page    Frame    Pin  ExtSegid  ExtPage
  0     135766   N    -         -
  2     859765   N    -         -
  3     586502   N    -         -
  4     859783   N    -         -
  6     859789   N    -         -
  7     465940   N    -         -
  8     859825   N    -         -
  9     434850   N    -         -
 10     859829   N    -         -
 11     436416   N    -         -
 12     859833   N    -         -
 14     859845   N    -         -
 15     470036   N    -         -
 16     859401   N    -         -
 17     641112   N    -         -
 18     859411   N    -         -
 19     641096   N    -         -
```

```
20      859409      N      -      -  
...(lines omitted)...
```

This output shows that frame 135766 is one of the frames that is used by this segment. Now we can run the frame report (-F) continuously to monitor this frame, as shown in Example 24-75.

Example 24-75 svmon -F -i 5

```
lpar05:/>> svmon -F 135766 -i 5  
  
      Frame Segid Ref Mod   Pincount   State   Swbits ExtSegid LPage  
135766 4aa49  Y  N     0/0      In-Use 88000004      -   N  
  
Frame Segid Ref Mod   Pincount   State   Swbits ExtSegid LPage  
135766 4aa49  N  N     0/0      In-Use 88000004      -   N  
  
      Frame Segid Ref Mod   Pincount   State   Swbits ExtSegid LPage  
135766 4aa49  N  N     0/0      In-Use 88000004      -   N  
  
      Frame Segid Ref Mod   Pincount   State   Swbits ExtSegid LPage  
135766 4aa49  Y  N     0/0      In-Use 88000004      -   N  
  
      Frame Segid Ref Mod   Pincount   State   Swbits ExtSegid LPage  
135766 4aa49  N  N     0/0      In-Use 88000004      -   N  
...(lines omitted)...
```

The first report line shows that the frame is referenced by the **dd** command that causes VMM to page in the JFS2 file system file into a frame. The next two report lines show that the page scanning has removed the reference flag (can be freed by the page stealer); this is shown as an N in the Ref column . We restart the **dd** command, and the frame containing the JFS2 filesystem file data is reused by the second **dd** command. (The files pages were already loaded in real memory.) The next two lines show that the reference flag has been removed by the page scanner again.

Note that the detailed segment report gives a similar output when both -D and -b flags are used, as shown in Example 24-76.

Example 24-76 svmon -bD segment

```
lpar05:/>> svmon -bD 4aa49  
  
Segid: 4aa49  
Type: client  
LPage: N  
Address Range: 0..86655
```

Page	Frame	Pin	Ref	Mod	ExtSegid	ExtPage
0	135766	N	N	N	-	-
2	859765	N	N	N	-	-
3	586502	N	N	N	-	-
4	859783	N	N	N	-	-
6	859789	N	N	N	-	-
7	465940	N	N	N	-	-
8	859825	N	N	N	-	-
9	434850	N	N	N	-	-
10	859829	N	N	N	-	-
11	436416	N	N	N	-	-
12	859833	N	N	N	-	-
14	859845	N	N	N	-	-
15	470036	N	N	N	-	-
16	859401	N	N	N	-	-
17	641112	N	N	N	-	-
18	859411	N	N	N	-	-
19	641096	N	N	N	-	-

...(lines omitted)...

Disk I/O–related performance tools

This part describes the tools to monitor the performance-relevant data and statistics for disk I/O.

- ▶ The **filemon** command described in Chapter 25, “The filemon command” on page 457 monitors a trace of file system and I/O system events, and reports performance statistics for files, virtual memory segments, logical volumes, and physical volumes.
- ▶ The **fileplace** command described in Chapter 26, “The fileplace command” on page 479 displays the placement of a file’s logical or physical blocks within a Journaled File System (JFS).

- ▶ Chapter 27, “The `lslv`, `lspv`, and `lsvg` commands” on page 501 describes the `lslv` command, which displays the characteristics and status of the logical volume; the `lspv` command, which is useful for displaying information about the physical volume, its logical volume content, and logical volume allocation layout; and the `lsvg` command, which displays information about volume groups.
- ▶ The `lvmstat` command described in Chapter 28, “The `lvmstat` command” on page 519 reports input and output statistics for logical partitions, logical volumes, and volume groups.

The filemon command

The **filemon** command monitors a trace of file-system and I/O-system events and reports performance statistics for files, virtual memory segments, logical volumes, and physical volumes. **filemon** is useful to those whose applications are believed to be disk-bound and want to know where and why. For file-specific layout and distribution, refer to Chapter 26, “The fileplace command” on page 479.

Monitoring disk I/O with the **filemon** command is usually done when there is a known performance issue regarding the I/O. The **filemon** command shows the load on different disks, logical volumes, and files in great detail.

The **filemon** command resides in /usr/sbin and is part of the bos.perf.tools filesset, which is installable from the AIX base installation media.

25.1 filemon

The syntax of the **filemon** command is:

```
filemon [ -d ] [ -i Trace_File -n Gennames_File ] [ -o File ] [ -O Levels ] [ -P ]  
[ -T n ] [ -u ] [ -v ]
```

Flags

- i Trace_File** Reads the I/O trace data from the specified *Trace_File*, instead of from the real-time trace process. The **filemon** report summarizes the I/O activity for the system and period represented by the trace file. The **-n** option must also be specified.
- n Gennames_File** Specifies a *Gennames_File* for offline trace processing. This file is created by running the **gennames** command and redirecting the output to a file as follows. (The **-i** option must also be specified.)
- o File** Writes the I/O activity report to the specified *File* instead of to the *stdout* file.
- d** Starts the **filemon** command, but defers tracing until the **trcon** command has been executed by the user. By default, tracing is started immediately.
- T n** Sets the kernel's trace buffer size to *n* bytes. The default size is 32,000 bytes. The buffer size can be increased to accommodate larger bursts of events. (A typical event record size is 30 bytes.).
- P** Pins monitor process in memory. The **-P** flag causes the **filemon** command's text and data pages to be pinned in memory for the duration of the monitoring period. This flag can be used to ensure that the real-time **filemon** process is not paged out when running in a memory-constrained environment.
- v** Prints extra information in the report. The most significant effect of the **-v** flag is that all logical files and all segments that were accessed are included in the I/O activity report instead of only the 20 most active files and segments.
- O Levels** Monitors only the specified file-system levels. Valid level identifiers are:
- lf** Logical file level
 - vm** Virtual memory level

- lv** Logical volume level
- pv** Physical volume level
- all** Short for lf, vm, lv, and pv

The vm, lv, and pv levels are implied by default.

- u** Reports on files that were opened prior to the start of the trace daemon. The process ID (PID) and the file descriptor (FD) are substituted for the file name.

25.1.1 Information about measurement and sampling

To provide a more complete understanding of file system performance for an application, the **filemon** command monitors file and I/O activity at four levels:

Logical file system The **filemon** command monitors logical I/O operations on logical files. The monitored operations include all read, write, open, and lseek system calls, which may or may not result in actual physical I/O depending on whether the files are already buffered in memory. I/O statistics are kept on a per-file basis.

Virtual memory system The **filemon** command monitors physical I/O operations (that is, paging) between segments and their images on disk. I/O statistics are kept on a per-segment basis.

Logical volumes The **filemon** command monitors I/O operations on logical volumes. I/O statistics are kept on a per-logical-volume basis.

Physical volumes The **filemon** command monitors I/O operations on physical volumes. At this level, physical resource utilizations are obtained. I/O statistics are kept on a per-physical-volume basis.

Any combination of the four levels can be monitored, as specified by the command line flags. By default, the **filemon** command only monitors I/O operations at the virtual memory, logical volume, and physical volume levels. These levels are all concerned with requests for real disk I/O.

The **filemon** command monitors a trace of a specific number of trace hooks, such as for file system and disk I/O . (See Chapter 40, “The trace, trcnm, and trcrpt commands” on page 759 for more about the **trace** command and tracehooks). You can list the trace hooks used by **filemon** by using the **trcevgrp** command as in Example 25-1.

Example 25-1 Using trcevgrp

```
# trcevgrp -l filemon
filemon - Hooks for FILEMON performance tool (reserved)
101,102,104,106,107,10B,10C,10D,12E,130,139,154,15B,163,19C,1BA,1BE,1BC,1C9,221
,222,232,3D3,45B
```

The **filemon** tracing of I/O is usually stopped by issuing the **trcstop** command; it is when this is done that **filemon** writes the output. **filemon** tracing can be paused by using the **trcoff** command and restarted by using the **trcon** command. By default, **filemon** starts tracing immediately, but tracing may be deferred until a **trcon** command is issued if the **-d** flag is used.

The **filemon** command can also process a trace file that has been previously recorded by the trace facility. The file and I/O activity report will be based on the events recorded in that file. In order to include all trace hooks that are needed for **filemon**, use the **-J filemon** option when running the **trace** command.

General notes on interpreting the reports

Check for most active segments, logical volumes, and physical volumes in this report. Check for reads and writes to paging space to determine if the disk activity is true application I/O or is due to paging activity. Check for files and logical volumes that are particularly active.

Value ranges

In some **filemon** reports there are different value ranges such as **min**, **max**, **avg**, and **sdev**. The **min** represents the minimum value, the **max** represents the maximum value, **avg** is the average, and **sdev** is the standard deviation, which shows how much the individual response times deviated from the average. If the distribution of response times is scattered over a large range, the standard deviation will be large compared to the average response time.

Access pattern analysis

As the read sequences count approaches the reads count, file access is more random. On the other hand, if the read sequence count is significantly smaller than the reads count and the read sequence length is a high value, the file access is more sequential. The same applies to the writes and write sequences. Sequences are strings of pages that are read (paged in) or written (paged out) consecutively. The **seq. lengths** is the length, in pages, of the sequences.

Fragmentation analysis

The amount of fragmentation in a logical volume or a file (blocks) cannot be obtained directly from the **filemon** output.

The amount of fragmentation and sequentiality of a file can be obtained by using the **fileplace** command on that file. (See Chapter 26, “The fileplace command” on page 479.) However, if seek times are larger than the number of reads and writes, there is more fragmentation and less sequentiality.

It is more difficult for a logical volume, which can be viewed as having two parts. The first part is the logical partitions that constitute the logical volume. To determine fragmentation on the logical volume, use the **lslv** command to determine sequentiality and space efficiency. (Refer to Chapter 27, “The lslv, lspv, and lsvg commands” on page 501.) The second part is the file system. This part is more complex because a file system contains meta data areas such as inode and data block maps, and, in the case of J2, it can also contain an inline journaling log, and of course the data blocks that contain the actual file data. Note that the output from **filemon** cannot be used to determine whether a file system has many files that are fragmented.

Segments

The Most Active Segments report lists the most active files by file system and inode. This report is useful in determining whether the activity is to a file system (segtype is persistent), the JFS log (segtype is log), or to paging space (segtype is working).

Unknown files

In some cases you will find references to unknown files. The mount point of the file system and inode of the file can be used with the **ncheck** command to identify these files:

```
ncheck -i <inode> <mount point>
```

Example 25-2 shows how this works.

Example 25-2 Checking filenames by using ncheck and inode number

```
# ncheck -i 36910 /home  
/home:  
36910 /dude/out/bigfile
```

When using the **ncheck** command, both the mount point and the file path within that mount point must be concatenated. In the example above this would be `/home/dude/out/bigfile`.

25.1.2 Examples for filemon

The output from **filemon** can be quite extensive. To quickly find out if anything needs attention, we filtered it with the **awk** command for most of our examples below to extract specific summary tables from the **filemon** output file.

Starting the monitoring

Example 25-3 shows how to run **filemon**. To have **filemon** monitor I/O during a time interval just run the **sleep** program with the specified amount of seconds and then the **trcstop** program. Below we have used the **all** option, and then the **awk** command to extract relevant parts of the complete report. Note that the output will be put in the **filemon.out** file in Example 25-3.

Example 25-3 Using filemon

```
# filemon -uo filemon.out -o all; sleep 60; trcstop
```

Enter the "trcstop" command to complete filemon processing

```
[filemon command: Reporting started]
```

```
[filemon command: Reporting completed]
```

```
[filemon command: 96.371 secs in measured interval]
```

Using different reports

The following is one way to use the analysis of one report as input to another report to pinpoint possible bottlenecks and performance issues. In Example 25-4, we start at the bottom and look at disk I/O, and extract a part of the report generated by **filemon**, which is the Most Active Physical Volumes.

Example 25-4 Most Active Physical Volumes report

```
# awk '/Most Active Physical Volumes/,/^\$/' filemon.out
```

```
Most Active Physical Volumes
```

util	#rblk	#wblk	KB/s	volume	description	

0.24	16	50383	171.9	/dev/hdisk0	N/A	
0.08	68608	36160	357.4	/dev/hdisk1	N/A	

This shows us that **hdisk1** is more utilized than **hdisk0**, with almost twice the amount of transferred data (KB/s). However **hdisk0** is more utilized with 24 percent compared to eight percent for **hdisk1** but this is mostly for writing whereas **hdisk1** has twice the amount of reading as it has writing. At this point we could also examine the disks, volume groups, and logical volumes with static

reporting commands such as **lspv**, **lsvg**, and **lslv** (Chapter 27, “The lslv, lspv, and lsvg commands” on page 501). To get more detailed realtime information about the usage of the logical volumes, extract the Most Active Logical Volumes part from our previously created output file as shown in Example 25-5.

Example 25-5 Most Active Logical Volumes report

```
# awk '/Most Active Logical Volumes/,/^$/' filemon.out
Most Active Logical Volumes
```

util	#rblk	#wblk	KB/s	volume	description
0.22	0	37256	127.1	/dev/hd8	jfslog
0.08	68608	36160	357.4	/dev/lv0hd0	N/A
0.04	0	11968	40.8	/dev/hd3	/tmp
0.01	0	312	1.1	/dev/hd4	/
0.01	16	536	1.9	/dev/hd2	/usr
0.00	0	311	1.1	/dev/hd9var	/var Frag_Sz.= 512

The logical volume **lv0hd0** is the most utilized for both reading and writing (but still only at 8 percent utilization), so now we extract information about this particular logical volume from the output file. Example 25-6 shows the report with the summary part and a detailed section as well.

Example 25-6 Detailed output for a logical volume

```
# awk '/VOLUME: \\dev\\lv0hd0/,/^$/' filemon.out
VOLUME: /dev/lv0hd0 description: N/A
reads: 1072 (0 errs)
  read sizes (blks): avg 64.0 min 64 max 64 sdev 0.0
  read times (msec): avg 7.112 min 2.763 max 29.334 sdev 2.476
  read sequences: 1072
  read seq. lengths: avg 64.0 min 64 max 64 sdev 0.0
writes: 565 (0 errs)
  write sizes (blks): avg 64.0 min 64 max 64 sdev 0.0
  write times (msec): avg 7.378 min 2.755 max 13.760 sdev 2.339
  write sequences: 565
  write seq. lengths: avg 64.0 min 64 max 64 sdev 0.0
seeks: 1074 (65.6%)
  seek dist (blks): init 60288,
  avg 123.6 min 64 max 64000 sdev 1950.9
time to next req(msec): avg 89.512 min 3.135 max 1062.120 sdev 117.073
throughput: 357.4 KB/sec
utilization: 0.08
```

In this example note that the I/O is random because both the reads (1072) equal the read sequences (1072), as does the writes and write sequences. To determine which files were most utilized during our monitoring, the Most Active Files report in Example 25-7 can be used.

Example 25-7 Most Active Files report

```
# awk '/Most Active Files/,/^\$/' filemon.out
Most Active Files
-----
```

#MBs	#opns	#rds	#wrs	file	volume:inode
337.3	2059	86358	0	fma.data	/dev/hd2:342737
176.7	2057	45244	0	fms.data	/dev/hd2:342738
45.6	1	1010	450	r1v0hd0	
9.6	2	2458	0	unix	/dev/hd2:30988
6.8	12	66140	0	errlog	/dev/hd9var:2065

```
...(lines omitted)...
```

We now find the fma.data file, and by running the `lsfs` command we learn that hd2 is the /usr filesystem, as shown in Example 25-8.

Example 25-8 Determining which filesystem uses a known logical volume

```
# lsfs|awk '/\/dev\/hd2/{print $3}'
/usr
```

Then we can search for the file within the /usr filesystem as shown in Example 25-9.

Example 25-9 Finding a file in a filesystem

```
# find /usr -name fma.data
/usr/lpp/htx/rules/reg/hxef1p/fma.data
```

We now have both the filename and the path, so we can now check how the file is allocated on the logical volume by using the `fileplace` command. (See Chapter 26, “The fileplace command” on page 479.)

Analyzing the physical volume reports

The physical volume report is divided into three parts; the header, the physical volume summary, and the detailed physical volume report. The header shows when and where the report was created and the CPU utilization during the monitoring period. To create only a physical volume report, issue the `filemon` command as follows (we are using a six-second measurement period):

```
filemon -uo filemon.pv -0 pv;sleep 6;trcstop
```

Example 25-10 shows the full physical volume report. In the report, the disks are presented in descending order of utilization. The disk with the highest utilization is shown first.

Example 25-10 Physical volume report

Mon Jun 4 08:21:16 2001

System: AIX wlmhost Node: 5 Machine: 000BC6AD4C00

Cpu utilization: 12.8%

Most Active Physical Volumes

```
-----
util #rblk #wblk KB/s volume description
-----
0.77 10888 1864 811.5 /dev/hdisk3 N/A
0.36 7352 2248 610.9 /dev/hdisk2 N/A
..(lines omitted)...
```

Detailed Physical Volume Stats (512 byte blocks)

```
-----
VOLUME: /dev/hdisk3 description: N/A
reads: 717 (0 errs)
 read sizes (blks): avg 15.2 min 8 max 64 sdev 11.9
 read times (msec): avg 12.798 min 0.026 max 156.093 sdev 19.411
 read sequences: 645
 read seq. lengths: avg 16.9 min 8 max 128 sdev 15.0
writes: 142 (0 errs)
 write sizes (blks): avg 13.1 min 8 max 56 sdev 9.2
 write times (msec): avg 16.444 min 0.853 max 50.547 sdev 8.826
 write sequences: 142
 write seq. lengths: avg 13.1 min 8 max 56 sdev 9.2
seeks: 786 (91.5%)
 seek dist (blks): init 0,
                  avg 525847.9 min 8 max 4284696 sdev 515636.2
 seek dist (%tot blks):init 0.00000,
                  avg 2.95850 min 0.00005 max 24.10632 sdev 2.90104
time to next req(msec): avg 14.069 min 0.151 max 75.270 sdev 14.015
throughput: 811.5 KB/sec
utilization: 0.77
```

```
VOLUME: /dev/hdisk2 description: N/A
reads: 387 (0 errs)
 read sizes (blks): avg 19.0 min 8 max 72 sdev 18.5
 read times (msec): avg 5.016 min 0.007 max 14.633 sdev 4.157
 read sequences: 235
 read seq. lengths: avg 31.3 min 8 max 384 sdev 58.1
writes: 109 (0 errs)
```

```

write sizes (blks):  avg   20.6 min      8 max    64 sdev   16.7
write times (msec): avg  13.558 min   4.569 max  26.689 sdev   5.596
write sequences:    109
write seq. lengths: avg   20.6 min      8 max    64 sdev   16.7
seeks:              344   (69.4%)
seek dist (blks):   init 4340200,
                   avg 515940.3 min      8 max 1961736 sdev 486107.7
seek dist (%tot blks):init 24.41859,
                   avg 2.90276 min 0.00005 max 11.03701 sdev 2.73491
time to next req(msec): avg  15.813 min  0.134 max 189.876 sdev  27.143
throughput:         610.9 KB/sec
utilization:        0.36

```

In Example 25-11, we only extract the physical volume summary section.

Example 25-11 Most Active Physical Volumes section

```

# awk '/Most Active Physical Volumes/,/^\$/' filemon.pv
Most Active Physical Volumes

```

util	#rblk	#wblk	KB/s	volume	description
0.24	16	50383	171.9	/dev/hdisk0	N/A
0.84	370680	372028	3853.4	/dev/hdisk1	N/A
0.08	68608	36160	357.4	/dev/hdisk2	N/A

The disk with the highest transfer rate and utilization is hdisk3, which is 84 percent utilized (0.84) at a 3.8 MB transfer rate.

The fields, in the Most Active Physical Volumes report of the **filemon** command, are interpreted as follows:

util	Utilization of the volume (fraction of time busy). The rows are sorted by this field in decreasing order.
#rblk	Number of 512-byte blocks read from the volume.
#wblk	Number of 512-byte blocks written to the volume.
KB/sec	Total volume throughput, in kilobytes per second.
volume	Name of volume.
description	Type of volume.

To find out more detail about a specific disk, look further in the report generated by **filemon**, as shown in Example 25-12.

Example 25-12 Detailed physical volume report section

```

# grep -p "VOLUME:.*hdisk3" filemon.pv
VOLUME: /dev/hdisk3 description: N/A

```



```

reads:                914      (0 errs)
  read sizes (blks):   avg   24.0 min      8 max      64 sdev   21.8
  read times (msec):   avg  4.633 min    0.275 max   14.679 sdev  4.079
  read sequences:      489
  read seq. lengths:   avg   44.8 min      8 max     384 sdev   81.2
writes:             218      (0 errs)
  write sizes (blks):  avg   21.3 min      8 max      64 sdev   17.6
  write times (msec):  avg 15.552 min    4.625 max  32.366 sdev  5.686
  write sequences:  218
  write seq. lengths:  avg   21.3 min      8 max      64 sdev   17.6
seeks:             707      (62.5%)
  seek dist (blks):    init 4671584,
                      avg 574394.4 min      8 max 2004640 sdev 484350.2
  seek dist (%tot blks):init 26.28301,
                      avg 3.23163 min 0.00005 max 11.27840 sdev 2.72502
time to next req(msec): avg 10.279 min 0.191 max 175.833 sdev 15.355
throughput:           1137.4 KB/sec
utilization:       0.52

```

The output shows that the disk has had a 52% utilization during the measuring interval, and that it is mostly random read and writes; 62.5% seeks for reads and writes. You can also see that the read I/O is mixed between random and sequential, but the writing is random.

The fields, in the Detailed Physical Volume report of the `filemon` command, are interpreted as follows:

VOLUME	Name of the volume.
description	Description of the volume (describes contents, if discussing a logical volume; and type, if dealing with a physical volume).
reads	Number of read requests made against the volume.
read sizes (blks)	The read transfer-size statistics (avg/min/max/sdev) in units of 512-byte blocks.
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.
read sequences	Number of read sequences. A sequence is a string of 512-byte blocks that are read consecutively and indicate the amount of sequential access.
read seq. lengths	Statistics describing the lengths of the read sequences in blocks.
writes	Number of write requests made against the volume.
write sizes (blks)	The write transfer-size statistics.

write times (msec)	The write-response time statistics.
write sequences	Number of write sequences. A sequence is a string of 512-byte blocks that are written consecutively.
write seq. lengths	Statistics describing the lengths of the write sequences, in blocks.
seeks	Number of seeks that preceded a read or write request, also expressed as a percentage of the total reads and writes that required seeks.
seek dist (blks)	Seek distance statistics, in units of 512-byte blocks. In addition to the usual statistics (avg/min/max/sdev), the distance of the initial seek operation (assuming block 0 was the starting position) is reported separately. This seek distance is sometimes very large, so it is reported separately to avoid skewing the other statistics.
seek dist (cyls)	Seek distance statistics, in units of disk cylinders.
time to next req	Statistics (avg/min/max/sdev) describing the length of time, in milliseconds, between consecutive read or write requests to the volume. This column indicates the rate at which the volume is being accessed.
throughput	Total volume throughput, in Kilobytes per second.
utilization	Fraction of time the volume was busy. The entries in this report are sorted by this field, in decreasing order.

Analyzing the file report

The logical file report is divided into three parts: the header, the file summary, and the detailed file report. The header shows when and where the report was created and the CPU utilization during the monitoring period. To create only a logical file report, issue the `filemon` command as follows (in this case using a six-second measurement period):

```
filemon -uo filemon.lf -0 lf;sleep 6;trcstop
```

Example 25-13 shows the full file report. In the report the file with the highest utilization is in the beginning and then listed in descending order.

Example 25-13 File report

```
Mon Jun  4 09:06:27 2001
System: AIX wlmhost Node: 5 Machine: 000BC6AD4C00
```

```
TRACEBUFFER 2 WRAPAROUND, 18782 missed entries
Cpu utilization: 24.8%
```

```
18782 events were lost. Reported data may have inconsistencies or errors.
```

Most Active Files

```
-----  
#MBs #opns #rds #wrs file volume:inode  
-----  
53.5 33 5478 0 file.tar /dev/data1v:17  
1.3 324 324 0 group /dev/hd4:4110  
1.2 0 150 0 pid=0_fd=15820  
0.6 163 163 0 passwd /dev/hd4:4149  
0.4 33 99 0 methods.cfg /dev/hd2:8492  
0.3 0 32 0 pid=0_fd=21706  
...(lines omitted)...
```

Detailed File Stats

```
-----  
FILE: /data/file.tar volume: /dev/data1v (/data) inode: 17  
opens: 33  
total bytes xfrd: 56094720  
reads: 5478 (0 errs)  
read sizes (bytes): avg 10240.0 min 10240 max 10240 sdev 0.0  
read times (msec): avg 0.090 min 0.080 max 0.382 sdev 0.017  
...(lines omitted)...
```

In Example 25-14 we only extract the file summary section.

Example 25-14 Most Active Files report section

```
# awk '/Most Active Files/,/^\$/' filemon.out  
Most Active Files
```

```
-----  
#MBs #opns #rds #wrs file volume:inode  
-----  
180.8 1 0 46277 index.db /dev/hd3:107  
53.5 33 5478 0 file.tar /dev/data1v:17  
1.3 324 324 0 group /dev/hd4:4110  
1.2 0 150 0 pid=0_fd=15820  
0.6 163 163 0 passwd /dev/hd4:4149  
...(lines omitted)...
```

We notice heavy reading (#rds) of the file.tar file and writing (#wrs) of the index.db. The fields, in the Most Active Files report of the **filemon** command, are interpreted as follows:

#MBS Total number of megabytes transferred to/from file. The rows are sorted by this field in decreasing order.

#opns	Number of times the file was opened during measurement period.
#rds	Number of read system calls made against file.
#wrs	Number of write system calls made against file.
file	Name of file (full path name is in detailed report).
volume:inode	Name of volume that contains the file, and the files inode number. This field can be used to associate a file with its corresponding persistent segment, shown in the virtual memory I/O reports. This field may be blank; for example, for temporary files created and deleted during execution.

To find out more detail about a specific file, look further in the report generated by **filemon** as shown in Example 25-15.

Example 25-15 Detailed File report section

```
# grep -p "FILE:.*file.tar" filemon4.1f
FILE: /data/file.tar volume: /dev/data1v (/data) inode: 17
opens: 33
total bytes xfrd: 56094720
reads: 5478 (0 errs)
  read sizes (bytes): avg 10240.0 min 10240 max 10240 sdev 0.0
  read times (msec): avg 0.090 min 0.080 max 0.382 sdev 0.017
```

The fields in the detailed file report of the **filemon** command are interpreted as follows:

FILE	Name of the file. The full path name is given, if possible.
volume	Name of the logical volume/file system containing the file.
inode	Inode number for the file within its file system.
opens	Number of times the file was opened while monitored.
total bytes xfrd	Total number of bytes read/written to/from the file.
reads	Number of read calls against the file.
read sizes (bytes)	The read transfer-size statistics (avg/min/max/sdev) in bytes.
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.
writes	Number of write calls against the file.
write sizes (bytes)	The write transfer-size statistics.
write times (msec)	The write response-time statistics.
seeks	Number of lseek subroutine calls.

Analyzing the logical volume report

The logical volume report has three parts; the header, the logical volume summary, and the detailed logical volume report. The header shows when and where the report was created and the CPU utilization during the monitoring period. To create only a logical volume report, issue the **filemon** command as follows (in this case using a six-second measurement period):

```
filemon -uo filemon.lv -0 lv;sleep 6;trcstop
```

Example 25-16 shows the full logical volume report. The logical volume with the highest utilization is at the top, and the others are listed in descending order.

Example 25-16 Logical volume report

```
Mon Jun  4 09:17:45 2001
```

```
System: AIX wlmhost Node: 5 Machine: 000BC6AD4C00
```

```
Cpu utilization: 13.9%
```

Most Active Logical Volumes

util	#rblk	#wblk	KB/s	volume	description
0.78	10104	2024	761.1	/dev/lv05	jfs2
0.39	10832	2400	830.4	/dev/lv04	jfs2
0.04	0	128	8.0	/dev/hd2	/usr

...(lines omitted)...

Detailed Logical Volume Stats (512 byte blocks)

```
VOLUME: /dev/lv05 description: jfs2
```

```
reads: 727 (0 errs)
```

```
read sizes (blks): avg 13.9 min 8 max 64 sdev 10.5
```

```
read times (msec): avg 19.255 min 0.369 max 72.025 sdev 15.779
```

```
read sequences: 587
```

```
read seq. lengths: avg 17.2 min 8 max 136 sdev 16.7
```

```
writes: 162 (0 errs)
```

```
write sizes (blks): avg 12.5 min 8 max 56 sdev 7.1
```

```
write times (msec): avg 12.911 min 3.088 max 57.502 sdev 7.814
```

```
write sequences: 161
```

```
write seq. lengths: avg 12.6 min 8 max 56 sdev 7.1
```

```
seeks: 747 (84.0%)
```

```
seek dist (blks): init 246576,
```

```
avg 526933.0 min 8 max 1994240 sdev 479435.6
```

```
time to next req(msec): avg 8.956 min 0.001 max 101.086 sdev 13.560
```

```
throughput: 761.1 KB/sec
```

```
utilization: 0.78
```

```
...(lines omitted)...
```

In Example 25-17, we extract only the logical volume section.

Example 25-17 Most Active Logical Volumes report

```
# awk '/Most Active Logical Volumes/,/^$/' filemon.out
```

Most Active Logical Volumes

util	#rblk	#wblk	KB/s	volume	description
1.00	370664	370768	3846.8	/dev/hd3	/tmp
0.02	0	568	2.9	/dev/hd8	jfslog
0.01	0	291	1.5	/dev/hd9var	/var Frag_Sz.= 512
0.00	0	224	1.2	/dev/hd4	/
0.00	0	25	0.1	/dev/hd1	/home Frag_Sz.= 512
0.00	16	152	0.9	/dev/hd2	/usr

The logical volume hd3 with filesystem /tmp is fully utilized (100 percent) with a 3.8 MB transfer rate per second.

The fields in the Most Active Logical Volumes report of the **filemon** command are:

util	Utilization of the volume (fraction of time busy). The rows are sorted by this field, in decreasing order. The first number, 1.00, means 100 percent.
#rblk	Number of 512-byte blocks read from the volume.
#wblk	Number of 512-byte blocks written to the volume.
KB/sec	Total transfer throughput in kilobytes per second.
volume	Name of volume.
description	Contents of volume; either a filesystem name or logical volume type (jfs, jfs2, paging, jfslog, jfs2log, boot, or sysdump). Also, indicates if the file system is fragmented or compressed.

To check the details of the highest utilized logical volumes, create a script as shown in Example 25-18 (here we call it `filemon.lvdetail`) and then run it using the **filemon** output file as input.

Example 25-18 Script filemon.lvdetail

```
#!/bin/ksh
file=${1:-filemon.out}
switch=${2:-0.20} # 20%

# extract the summary table...
awk '/Most Active Logical Volumes/,/^$/' $file|
# select logical volumes starting from line 5 and no empty lines...
```

```

awk 'NR>4&&$0!~/^$/{if ($1 >= switch)print $5}' switch=$switch|
  while read lv;do
# strip the /dev/ stuff and select the detail section.
  awk '/VOLUME: \/dev\/'$\{lv##*\}'/,/^$/' $file
  done

```

For our continuing example the result would appear as shown in Example 25-19.

Example 25-19 Logical volume detailed selection report

```

# filemon.lvdetail filemon.out
VOLUME: /dev/lv05 description: jfs2
reads:          727      (0 errs)
  read sizes (blks):  avg   13.9 min      8 max    64 sdev   10.5
  read times (msec):  avg  19.255 min   0.369 max  72.025 sdev  15.779
  read sequences:    587
  read seq. lengths:  avg   17.2 min      8 max   136 sdev   16.7
writes:         162      (0 errs)
  write sizes (blks): avg   12.5 min      8 max    56 sdev    7.1
  write times (msec): avg  12.911 min   3.088 max  57.502 sdev   7.814
  write sequences:   161
  write seq. lengths: avg   12.6 min      8 max    56 sdev    7.1
seeks:          747      (84.0%)
  seek dist (blks):  init 246576,
                    avg 526933.0 min      8 max 1994240 sdev 479435.6
time to next req(msec): avg   8.956 min   0.001 max  101.086 sdev  13.560
throughput:      761.1 KB/sec
utilization:     0.78

VOLUME: /dev/lv04 description: jfs2
reads:          510      (0 errs)
  read sizes (blks):  avg   21.2 min      8 max    72 sdev   18.6
  read times (msec):  avg   5.503 min   0.368 max  25.989 sdev   5.790
  read sequences:    265
  read seq. lengths:  avg   40.9 min      8 max   384 sdev   73.4
writes:         110      (0 errs)
  write sizes (blks): avg   21.8 min      8 max    64 sdev   16.8
  write times (msec): avg   9.994 min   4.440 max  18.378 sdev   2.752
  write sequences:   101
  write seq. lengths: avg   23.8 min      8 max    64 sdev   18.6
seeks:          366      (59.0%)
  seek dist (blks):  init 127264,
                    avg 538451.3 min      8 max 2009448 sdev 504054.2
time to next req(msec): avg  12.842 min   0.003 max  187.120 sdev  23.317
throughput:      830.4 KB/sec
utilization:     0.39

```

The descriptions for the detailed output shown in the example are:

VOLUME	Name of the volume.
description	Description of the volume. Describes contents, if discussing a logical volume, and type if dealing with a physical volume.
reads	Number of read requests made against the volume.
read sizes (blks)	The read transfer-size statistics (avg/min/max/sdev) in units of 512-byte blocks.
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.
read sequences	Number of read sequences. A sequence is a string of 512-byte blocks that are read consecutively and indicate the amount of sequential access.
read seq. lengths	Statistics describing the lengths of the read sequences in blocks.
writes	Number of write requests made against the volume.
write sizes (blks)	The write transfer-size statistics.
write times (msec)	The write-response time statistics.
write sequences	Number of write sequences. A sequence is a string of 512-byte blocks that are written consecutively.
write seq. lengths	Statistics describing the lengths of the write sequences in blocks.
seeks	Number of seeks that preceded a read or write request, also expressed as a percentage of the total reads and writes that required seeks.
seek dist (blks)	Seek distance statistics in units of 512-byte blocks. In addition to the usual statistics (avg/min/max/sdev), the distance of the initial seek operation (assuming block 0 was the starting position) is reported separately. This seek distance is sometimes very large, so it is reported separately to avoid skewing the other statistics.
seek dist (cyls)	(Hard files only) Seek distance statistics, in units of disk cylinders.
time to next req	Statistics (avg/min/max/sdev) describing the length of time, in milliseconds, between consecutive read or write requests to the volume. This column indicates the rate at which the volume is being accessed.
throughput	Total volume throughput in kilobytes per second.

utilization Fraction of time the volume was busy. The entries in this report are sorted by this field, in decreasing order.

Analyzing the virtual memory segments report

The virtual memory report has three parts: the header, the segment summary, and the detailed segment report. The header shows when and where the report was created and the CPU utilization during the monitoring period. To create only a virtual memory report, issue the `filemon` command as follows (in this case using a six-second measurement period):

```
filemon -uo filemon.vm -0 vm;sleep 6;trcstop
```

Example 25-20 shows the full virtual memory report, in which the segment with the highest utilization is at the top, and the others are listed in descending order.

Example 25-20 Virtual memory report

```
Mon Jun  4 09:34:17 2001
System: AIX wlmhost Node: 5 Machine: 000BC6AD4C00

Cpu utilization:  7.0%

Most Active Segments
-----
```

#MBs	#rpgs	#wpgs	segid	segtype	volume:inode
1.8	416	50	2058ab	page table	
1.4	301	57	8c91	page table	
1.3	286	52	4c89	page table	
1.3	311	23	2040a8	page table	
1.1	236	47	2068ad	page table	
1.0	201	54	2050aa	page table	
1.0	184	67	2048a9	page table	
0.7	123	46	2060ac	page table	
0.0	0	7	2084	log	
0.0	3	0	ec9d	???	

```
...(lines omitted)...
-----

Detailed VM Segment Stats  (4096 byte pages)
-----

SEGMENT: 2058ab  segtype: page table
segment flags:   pgtbl
reads:          416 (0 errs)
  read times (msec):  avg  3.596 min  0.387 max  24.262 sdev  3.500
  read sequences:    55
  read seq. lengths:  avg   7.6 min           1 max           48 sdev   13.6
writes:         50 (0 errs)
```

```

write times (msec):  avg  9.924 min  2.900 max  14.530 sdev  2.235
write sequences:  25
write seq. lengths:  avg   2.0 min    1 max    8 sdev   1.5

...(lines omitted)...

SEGMENT: 2084  segtype: log
segment flags:  log
writes:      7  (0 errs)
  write times (msec):  avg 12.259 min  7.381 max  15.250 sdev  2.499
  write sequences:  5
  write seq. lengths:  avg   1.4 min    1 max    2 sdev   0.5

SEGMENT: ec9d  segtype: ???
segment flags:
reads:      3  (0 errs)
  read times (msec):  avg  0.964 min  0.944 max  0.981 sdev  0.015
  read sequences:  1
  read seq. lengths:  avg   3.0 min    3 max    3 sdev   0.0

...(lines omitted)...

```

In Example 25-21 we only extract the segment section.

Example 25-21 Most Active Segments report

```

# awk '/Most Active Segments/,/^$/' filemon.out
Most Active Segments

```

```

-----
#MBs #rpgs #wpgs segid segtype volume:inode
-----
15.1 2382 1484 2070ae page table
14.3 2123 1526 2058ab page table
14.1 1800 1802 672d page table
13.9 2209 1353 6f2c page table
13.9 2287 1261 2060ac page table
13.4 2054 1383 2068ad page table
12.2 1874 1242 2050aa page table
11.6 1985 983 2048a9 page table
...(lines omitted)...

```

The fields in the Most Active Segments report of the **filemon** command are interpreted as follows:

#MBS	Total number of megabytes transferred to/from segment. The rows are sorted by this field in decreasing order.
#rpgs	Number of 4096-byte pages read into segment from disk.
#wpgs	Number of 4096-byte pages written from segment to disk.

segid	Internal ID of segment.
segtype	Type of segment: working segment, persistent segment, client segment, page table segment, system segment, or special persistent segments containing file system data (log, root directory, .inode, .inodemap, .inodex, .inodexmap, .indirect, .diskmap).
volume:inode	For persistent segments, name of volume that contains the associated file, and the files inode number. This field can be used to associate a persistent segment with its corresponding file, shown in the file I/O reports. This field is blank for non-persistent segments.

A detailed segment report is shown in Example 25-22.

Example 25-22 Detailed segment report

```
# grep -p "SEGMENT:.*\?\?\?" filemon.vm
SEGMENT: ec9d segtype: ???
segment flags:
reads:          3          (0 errs)
  read times (msec):  avg  0.964 min  0.944 max  0.981 sdev  0.015
  read sequences:    1
  read seq. lengths: avg   3.0 min    3 max    3 sdev   0.0
```

The fields, in the Detailed VM Segment Stats report of the `filemon` command, are interpreted as follows:

SEGMENT	Internal segment ID.
segtype	Type of segment contents.
segment flags	Various segment attributes.
volume	For persistent segments, the name of the logical volume containing the corresponding file.
inode	For persistent segments, the inode number for the corresponding file.
reads	Number of 4096-byte pages read into the segment (that is, paged in).
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.
read sequences	Number of read sequences. A sequence is a string of pages that are read (paged in) consecutively. The number of read sequences is an indicator of the amount of sequential access.

read seq. lengths	Statistics describing the lengths of the read sequences in pages.
writes	Number of pages written from the segment (paged out).
write times (msec)	Write response time statistics.
write sequences	Number of write sequences. A sequence is a string of pages that are written (paged out) consecutively.
write seq.lengths	Statistics describing the lengths of the write sequences in pages.

In this example, **filemon** only shows a segment ID and does not indicate whether it is a file, logical volume, or physical volume. To find out more about the segment we use the **svmon** command (refer to Chapter 24, “The svmon command” on page 387) as shown in Example 25-23.

Example 25-23 Using svmon to show segment information

```
# svmon -S ec9d
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
ec9d	-	pers	/dev/lv00:17	4	0	-	-

In Example 25-23, the **svmon** command with the **-S** flag shows that segment **ec9d** is a persistent segment, which means it is some kind of JFS file and it uses 4 * 4096 bytes of real memory (**Inuse**). To map the <device>:<inode>, shown above, into a file system path name, use the **ncheck** command as in Example 25-24.

Example 25-24 Using ncheck

```
# ncheck -i 17 /dev/lv00
/dev/lv00:
17      /read_write
```

The **ncheck** command shows the path name of a specified inode number within the specified file system (logical volume). To obtain the full path name to the file **read_write** (in the output above) we need the file system mount point, which can be obtained by using the **lsfs** command as shown in Example 25-25.

Example 25-25 Using lsfs

```
# lsfs /dev/lv00
```

Name	Nodename	Mount Pt	VFS	Size	Options	Auto	Accounting
/dev/lv00	--	/tools	jfs	32768	rw	yes	no

The absolute path to the **read_write** file is **/tools/read_write**.

The fileplace command

The **fileplace** command displays the placement of a file's logical or physical blocks within a Journaled File System (JFS), not Network File System (NFS) or Enhanced Journaled File System (J2). Logically contiguous files in the file system may be both logically and physically fragmented on the logical and physical volume level, depending on the available free space at the time the file and logical volume (file system) were created.

The **fileplace** command can be used to examine and assess the efficiency of a file's placement on disk and help identify those files that will benefit from reorganization.

The **fileplace** command resides in /usr/bin and is part of the bos.perf.tools filesset, which is installable from the AIX base installation media.

26.1 fileplace

The syntax of the **fileplace** command is:

```
fileplace [ { -l | -p } [ -i ] [ -v ] ] File  
fileplace [-m lvname ]
```

Flags

- i** Displays the indirect blocks for the file, if any. The indirect blocks are displayed in terms of either their logical or physical volume block addresses, depending on whether the **-l** or **-p** flag is specified.
- l** Displays file placement in terms of logical volume fragments for the logical volume containing the file. The **-l** and **-p** flags are mutually exclusive.
- p** Displays file placement in terms of underlying physical volume for the physical volumes that contain the file. If the logical volume containing the file is mirrored, the physical placement is displayed for each mirror copy. The **-l** and **-p** flags are mutually exclusive.
- v** Displays more information about the file and its placement, including statistics on how widely the file is spread across the volume and the degree of fragmentation in the volume. The statistics are expressed in terms of either the logical or physical volume fragment numbers, depending on whether the **-l** or **-p** flag is specified.
- m lvname** Displays logical to physical map for a logical volume.

Note: If neither the **-l** flag nor the **-p** flag is specified, the **-l** flag is implied by default. If both flags are specified, the **-p** flag is used.

Parameters

File The file to display information about.

26.1.1 Information about measurement and sampling

The **fileplace** command extracts information about a file's physical and logical disposition from the JFS logical volume *superblock* and *inode* tables directly from disk and displays this information in a readable form. If the file is newly created, extended, or truncated, the file system information may not yet be on the disk

when the **fileplace** command is run. In this case use the **sync** command to flush the file information to the logical volume.

Data on logical volumes (file systems) appears to be contiguous to the user but can be discontinuous on the physical volume. File and file system fragmentation can severely hurt I/O performance because it causes more disk arm movement. To access data from a disk, the disk controller must first be directed to the specified block by the LVM through the device driver. Then the disk arm must seek the correct cylinder. After that the read/write heads must wait until the correct block rotates under them. Finally the data must be transmitted to the controller over the I/O bus to memory before it can be made available to the application program. Of course some adapters and I/O architectures support both multiple outstanding I/O requests and reordering of those requests, which in some cases will be sufficient, but in most cases will not.

To assess the performance effect of file fragmentation, an understanding of how the file is used by the application is required:

- ▶ If the application is primarily accessing this file sequentially, the *logical fragmentation* is more important. At the end of each fragment read ahead stops. The fragment size is therefore very important.
- ▶ If the application is accessing this file randomly, the *physical fragmentation* is more important. The closer the information is in the file, the less latency there is when accessing the physical data blocks.

Attention: Avoid using fragmentation sizes smaller than 4096 bytes. Even though it is allowed, it will increase the need for system administration and can cause performance degradation in the I/O system. Fragmentation sizes smaller than 4096 are only useful when a file system is used for files smaller than the fragmentation size (<512, 1024, or 2048 bytes). If needed these filesystems should be created separately and defragmented regularly by using the **defragfs** command. If no other job control system is used in the system, use **cron** to execute the command on a regular basis. One scenario in which it could be appropriate is when an application creates many Simultaneous Peripheral Operation Off Line (SPOOL) files, for example printer files that are written once and read mainly one time (by the qdaemon).

26.1.2 Examples for fileplace

In Example 26-1 on page 482, the **fileplace** command lists to standard output the ranges of logical volume fragments allocated to the specified file. The order in which the logical volume fragments are listed corresponds directly to their order in the file.

Example 26-1 Using fileplace

fileplace index.db

File: index.db Size: 1812480 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no

Logical Fragment

0000016-0000023	8 frags	32768 Bytes,	1.8%
0000025-0000028	4 frags	16384 Bytes,	0.9%
0000544-0000974	431 frags	1765376 Bytes,	97.3%

The report shows that the majority of the file occupies a consecutive range of blocks starting from 544 and ending at 974 (97.3%).

Analyzing the logical report

The logical report that the **fileplace** command creates with the **-l** flag (default) displays the file placement in terms of logical volume fragments for the logical volume containing the file. It is shown in Example 26-2.

Example 26-2 Using fileplace -l

fileplace -l index.db

File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no

Logical Fragment

0000016-0000023	8 frags	32768 Bytes,	1.8%
0000025-0000028	4 frags	16384 Bytes,	0.9%
0000544-0000974	431 frags	1765376 Bytes,	97.3%

The fields, in the logical report of the **fileplace** command, are interpreted as:

File	The name of the file being examined
Size	The file size in bytes
Vol	The name of the logical volume where the file is placed
Blk Size	The block size in bytes for that logical volume
Frag Size	The fragment size in bytes
Nfrags	The number of fragments
Compress	Whether the file system is compressed
Logical Fragments	The logical block numbers where the file resides

The Logical Fragments part of the report is interpreted as, from left to right:

Start	The start of a consecutive block range
Stop	The end of the consecutive block range
Nfrags	Number of contiguous fragments in the block range
Size	The number of bytes in the contiguous fragments
Percent	Percentage of the block range compared with total file size

Portions of a file may not be mapped to any logical blocks in the volume. These areas are implicitly filled with null (0x00) by the file system when they are read. These areas show as *unallocated* logical blocks. A file that has these holes will show the file size to be a larger number of bytes than it actually occupies. Refer to “Sparsely allocated files” on page 492.

26.1.3 Analyzing the physical report

The physical report that the **fileplace** command creates with the **-p** flag displays the file placement in terms of underlying physical volume (or the physical volumes that contain the file). If the logical volume containing the file is mirrored, the physical placement is displayed for each mirror copy. The physical report is shown in Example 26-3.

Example 26-3 Using fileplace -p

```
# fileplace -p index.db
```

```
File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v  
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no
```

Physical Addresses (mirror copy 1)	Logical Fragment
-----	-----
0537136-0537143 hdisk1 8 frags 32768 Bytes, 1.8%	0000016-0000023
0537145-0537148 hdisk1 4 frags 16384 Bytes, 0.9%	0000025-0000028
0537664-0538094 hdisk1 431 frags 1765376 Bytes, 97.3%	0000544-0000974

The fields, in the physical report of the **fileplace** command, are interpreted as:

File	The name of the file being examined
Size	The file size in bytes
Vol	The name of the logical volume where the file is placed
Blk Size	The block size in bytes for that logical volume
Frag Size	The fragment size in bytes
Nfrags	The number of fragments

Compress	Whether the file system is compressed
Physical Address	The physical block numbers where the file resides for each mirror copy
The Physical Address part of the report are interpreted as, from left to right:	
Start	The start of a consecutive block range
Stop	The end of the consecutive block range
PVol	Physical volume where the block is stored
Nfrags	Number of contiguous fragments in the block range
Size	The number of bytes in the contiguous fragments
Percent	Percentage of block range compared with total file size
Logical Fragment	The logical block addresses corresponding to the physical block addresses

Portions of a file may not be mapped to any physical blocks in the volume. These areas are implicitly filled with null (0x00) by the file system when they are read. These areas show as *unallocated* physical blocks. A file that has these holes will show the file size to be a larger number of bytes than it actually occupies. Refer to “Sparsely allocated files” on page 492.

Analyzing the physical address

The Logical Volume Device Driver (LVDD) requires that all disks are partitioned in 512 bytes blocks. This is the physical disk block size, and is the basis for the block addressing reported by the `fileplace` command. Refer to “Interface to Physical Disk Device Drivers” in *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts* for more details.

The `XLATE ioctl` operation translates a logical address (logical block number and mirror number) to a physical address (physical device and physical block number on that device). Refer to the “XLATE ioctl Operation” in *AIX 5L Version 5.2 Files Reference* for more details.

Whatever the fragment size, a full block is considered to be 4096 bytes. In a file system with a fragment size less than 4096 bytes, however, a need for a full block can be satisfied by any contiguous sequence of fragments totalling 4096 bytes. It does not need to begin on a multiple-of-4096-byte boundary. For more information, refer to the *AIX 5L Version 5.2 Performance Management Guide*.

The primary performance hazard for file systems with small fragment sizes is space fragmentation. The existence of small files scattered across the logical volume can make it impossible to allocate contiguous or closely spaced blocks for a large file. Performance can suffer when accessing large files. Carried to an

extreme, space fragmentation can make it impossible to allocate space for a file, even though there are many individual free fragments.

Another adverse effect on disk I/O activity is the number of I/O operations. For a file with a size of 4 KB stored in a single fragment of 4 KB, only one disk I/O operation would be required to either read or write the file. If the choice of the fragment size was 512 bytes, eight fragments would be allocated to this file, and for a read or write to complete, several additional disk I/O operations (disk seeks, data transfers, and allocation activity) would be required. Therefore, for file systems that use a fragment size of 4 KB, the number of disk I/O operations might be far less than for file systems that employ a smaller fragment size.

Example 26-4 illustrates how the 512-byte physical disk block is reported by the `fileplace` command.

Example 26-4 Using fileplace -p

```
# fileplace -p file.log
```

```
File: file.log Size: 148549 bytes Vol: /dev/hd1
Blk Size: 4096 Frag Size: 512 Nfrags: 296 Compress: no
```

Physical Addresses (mirror copy 1)		Logical Fragment			
-----		-----			
4693063	hdisk0	8 frags	4096 Bytes,	2.7%	0052039
4693079	hdisk0	8 frags	4096 Bytes,	2.7%	0052055
4693106	hdisk0	8 frags	4096 Bytes,	2.7%	0052082
4693120	hdisk0	8 frags	4096 Bytes,	2.7%	0052096
0829504-0829528	hdisk0	32 frags	16384 Bytes,	10.8%	1562432-1562456
0825064-0825080	hdisk0	24 frags	12288 Bytes,	8.1%	1557992-1558008
0825120	hdisk0	8 frags	4096 Bytes,	2.7%	1558048
0825008-0825016	hdisk0	16 frags	8192 Bytes,	5.4%	1557936-1557944
0824182	hdisk0	8 frags	4096 Bytes,	5.4%	1557110-1557118
0829569-0829593	hdisk0	32 frags	16384 Bytes,	10.8%	1562497-1562521
0829632-0829656	hdisk0	32 frags	16384 Bytes,	10.8%	1562560-1562584
0829696-0829712	hdisk0	24 frags	12288 Bytes,	8.1%	1562624-1562640
0829792-0829864	hdisk0	80 frags	40960 Bytes,	27.0%	1562720-1562792

In the following explanation we use the following line from the previous example:

```
0825008-0825016 hdisk0 16 frags 8192 Bytes, 5.4% 1557936-1557944
```

As the fragment size is less than 4096 bytes in this case, the start range is the starting address of the 4096/FragSize contiguous blocks, and the end range is nothing but the starting address of the 4096/FragSize contiguous blocks.

Hence from 0825008 to 08250015 is the first 4096-byte block, which is occupied by the file (8 frags in this case), and from 08250016 to 08250023 is the next 4096-byte

block that is occupied by the file (8 frags, totals up to 16 frags now). Note that the actual range is 0825008–0850023, but instead 0825008–08250016 is displayed.

The reason why **fileplace** does not display the proper end physical address is that AIX always tries to allocate the specified block size contiguously on the disk. Hence, for a 4 KB block size, AIX will always look for eight contiguous 512-byte blocks on the disk to allocate. Hence **fileplace** always displays the start and end range in terms of *block addressing*.

So if the fragment size and block size are same, then **fileplace** display seems to be meaningful output, but if the block size and fragment size are not the same, then the output may be a little bit confusing. Actually **fileplace** always displays the address ranges in terms of start and end address of a block and not a fragment, even though the addressing is done based on fragments.

The formula **fileplace** uses to display the mapping of physical address, logical address, and fragments is:

Number of fragments = (End Address - Start Address) + (Block Size / Frag Size)

For more information refer also to "Understanding Fragments" in *AIX 5L Version 5.2 System Management Concepts: Operating System and Devices*.

To illustrate the addressing, consider an example in AIX where the word size is 4 bytes, which means that addressing is done for each and every 4 bytes. This example applies to the case of an array of the longlong type:

```
longlong word[10];
```

The starting address of word[0] is 123456. The display of the range of addresses occupied by this array is:

```
Start Address: 123456
End Address: 123474
Total no. of words occupied: 20
```

However, if you calculate $123474 - 123456 + 1 = 19$ words, this is one word less. The end address is nothing but the address of word[10], which occupies two words, so the actual formula in this case is:

(Endaddress - startaddress) + (Data size / wordsize)

With our example above it would be:

$(123474 - 123456) + (8 / 4) = 20$ words

Analyzing the indirect block report

The **fileplace -i** flag will display any indirect block(s) used for the file in addition to the default display or together with the **-l**, **-p**, or **-v** flags. Indirect block(s) are

needed for files larger than 32 KB. An single indirect block is used for storing addresses to data blocks when the inode's number of data block addresses is not sufficient. A double indirect block is used to store addresses to other blocks that in their turn store addresses to data blocks. For more detail on the use of the indirect block see *AIX 5L Version 5.2 System User's Guide: Operating System and Devices*.

The only additional fields to the physical or logical reports, when the `-i` option is used with `fileplace`, are interpreted as:

INDIRECT BLOCK	The physical/logical address of a data block that contains pointers (addresses) to data blocks.
DOUBLE INDIRECT BLOCK	The physical/logical address of a block that contains pointers (addresses) to other indirect blocks.
INDIRECT BLOCKS	The physical/logical address of a data block(s) that contains pointers (addresses) to data blocks.

In Example 26-5 using the logical report (`-l`), the indirect block's logical address is 24.

Example 26-5 Indirect block, logical view

```
# fileplace -il index.db
```

```
File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no
```

```
INDIRECT BLOCK: 00024
```

```
Logical Fragment
```

```
-----
```

0000016-0000023	8 frags	32768 Bytes,	1.8%
0000025-0000028	4 frags	16384 Bytes,	0.9%
0000544-0000974	431 frags	1765376 Bytes,	97.3%

Example 26-6, using the physical report (`-p`), shows that the indirect block's physical address is 537144.

Example 26-6 Indirect block, physical view

```
# fileplace -ip index.db
```

```
File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no
```

```
INDIRECT BLOCK: 537144
```

```
Physical Addresses (mirror copy 1)
```

```
Logical Fragment
```

```

-----
0537136-0537143 hdisk1      8 frags   32768 Bytes,  1.8%  0000016-0000023
0537145-0537148 hdisk1      4 frags   16384 Bytes,  0.9%  0000025-0000028
0537664-0538094 hdisk1    431 frags 1765376 Bytes, 97.3%  0000544-0000974
-----

```

Example 26-7, using the default logical report (-i), shows that the double indirect block's logical address is 01170, and the two currently existing indirect blocks' logical addresses are 00029 and 01171.

Example 26-7 Double indirect block and indirect blocks

```
# fileplace -i bolshoi.tar
```

```
File: bolshoi.tar Size: 5724160 bytes Vol: /dev/vg101v1
Blk Size: 4096 Frag Size: 4096 Nfrags: 1398 Compress: no
```

```
DOUBLE INDIRECT BLOCK: 01170
INDIRECT BLOCKS: 00029 01171
```

```
Logical Fragment
```

```

-----
0000144-0000147          4 frags   16384 Bytes,  0.3%
0000150-0001169       1020 frags 4177920 Bytes, 73.0%
0001172-0001545        374 frags 1531904 Bytes, 26.8%
-----

```

Analyzing the volume report

The volume report displays information about the file and its placement, including statistics about how widely the file is spread across the volume and the degree of fragmentation in the volume.

Logical report

In Example 26-8 the statistics are expressed in terms of logical fragment numbers. This is the logical block's placement on the logical volume, for each of the logical copies of the file.

Example 26-8 Using fileplace -vl

```
# fileplace -vl index.db
```

```
File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no
Inode: 17 Mode: -rw-r--r-- Owner: root Group: sys
```

```
Logical Fragment
```

```

-----
0000016-0000023          8 frags   32768 Bytes,  1.8%
0000025-0000028          4 frags   16384 Bytes,  0.9%
0000544-0000974       431 frags 1765376 Bytes, 97.3%
-----

```

443 frags over space of 959 frags: **space efficiency = 46.2%**

If the application primarily accesses this file sequentially, the logical fragmentation is important. When VMM reads a file sequentially, by default it uses read ahead. (For more information about tuning the read ahead size, see “Sequential read-ahead” on page 241.) At the end of each fragment, read ahead stops. The fragment size is therefore very important. High space efficiency means that the file is less fragmented. In the example above, the file has only 46.2 percent space efficiency for the logical fragmentation. Because the file in the example above is larger than 32 KB, it will never have 100 percent space efficiency because of the use of the indirect block.

Space efficiency is calculated as the number of non-null fragments (N) divided by the range of fragments assigned to the file (R) and multiplied by 100:

$$(N / R) * 100$$

Range is calculated as the highest assigned address ($MaxBlk$) minus the lowest assigned address ($MinBlk$) plus 1:

$$MaxBlk - MinBlk + 1$$

In Example 26-9 we use the logical (-l), indirect (-i), and volume (-v) flags with **fileplace** to show all interesting information from a logical point of view of a file.

Example 26-9 Using fileplace -liv

```
# fileplace -liv bolshoi.tar
```

```
File: bolshoi.tar Size: 5724160 bytes Vol: /dev/vg10lv1
Blk Size: 4096 Frag Size: 4096 Nfrags: 1398 Compress: no
Inode: 29 Mode: -rw-rw-r-- Owner: root Group: sys
```

```
DOUBLE INDIRECT BLOCK: 01170
INDIRECT BLOCKS: 00029 01171
```

```
Logical Fragment
```

```
-----
0000144-0000147          4 frags   16384 Bytes,   0.3%
0000150-0001169       1020 frags  4177920 Bytes,  73.0%
0001172-0001545        374 frags  1531904 Bytes,  26.8%
```

```
1398 frags over space of 1402 frags: space efficiency = 99.7%
3 fragments out of 1398 possible: sequentiality = 99.9%
```

This file uses double indirection for data block addresses. Both space efficiency and sequentiality are at very good levels (99.7 and 99.9 percent respectively).

Example 26-10 shows a file with zero sequentiality. It is a sparse file (see “Sparsely allocated files” on page 492) but the importance is the distance between the allocated blocks (1204 and 1205).

Example 26-10 Zero sequentiality

fileplace -liv ugly.file

File: ugly.file Size: 512001 bytes Vol: /dev/data1v
 Blk Size: 4096 Frag Size: 4096 Nfrags: 2 Compress: no
 Inode: 182 Mode: -rw-r--r-- Owner: root Group: sys

INDIRECT BLOCK: 01218

Logical Fragment

unallocated	12 frags	49152 Bytes,	0.0%
0001204	1 frags	4096 Bytes,	50.0%
unallocated	112 frags	458752 Bytes,	0.0%
0001205	1 frags	4096 Bytes,	50.0%

2 frags over space of 2 frags: space efficiency = 100.0%

2 fragments out of 2 possible: **sequentiality = 0.0%**

Physical report

In Example 26-11 the statistics are expressed in terms of physical volume fragment numbers. This is the logical block placement on physical volume(s) for each of the logical copies of the file.

Example 26-11 fileplace -vp

fileplace -vp index.db

File: /data/index.db Size: 1812480 bytes Vol: /dev/data1v
 Blk Size: 4096 Frag Size: 4096 Nfrags: 443 Compress: no
 Inode: 17 Mode: -rw-r--r-- Owner: root Group: sys

Physical Addresses (mirror copy 1)

Physical Addresses (mirror copy 1)				Logical Fragment	
-----				-----	
0537136-0537143	hdisk1	8 frags	32768 Bytes,	1.8%	0000016-0000023
0537145-0537148	hdisk1	4 frags	16384 Bytes,	0.9%	0000025-0000028
0537664-0538094	hdisk1	431 frags	1765376 Bytes,	97.3%	0000544-0000974

443 frags over space of 959 frags: **space efficiency = 46.2%**

3 fragments out of 443 possible: **sequentiality = 99.5%**

If the application primarily accesses this file randomly, the physical fragmentation is important. The closer the information is in the file, the less latency when accessing the physical data blocks. High sequentiality means that the file's

physical blocks are allocated more contiguously. In the example above, the file has a 99.5 percent sequentiality.

Sequential efficiency is defined as 1 minus the number of gaps (nG) divided by number of possible gaps (nPG): $1 - (nG / nPG)$.

The number of possible gaps equals N minus 1:

$$nPG = N - 1$$

In Example 26-12, we use the physical (-p), indirect (-i), and volume (-v) flags to **fileplace** to show us all of the interesting information from a physical point of view of a file.

Example 26-12 Using fileplace -piv

```
# fileplace -piv bolshoi.tar
```

```
File: bolshoi.tar Size: 5724160 bytes Vol: /dev/vg10lv1
Blk Size: 4096 Frag Size: 4096 Nfrags: 1398 Compress: no
Inode: 29 Mode: -rw-rw-r-- Owner: root Group: sys
```

```
DOUBLE INDIRECT BLOCK: 01714
```

```
INDIRECT BLOCKS: 00573 01715
```

Physical Addresses (mirror copy 1)					Logical Fragment
-----					-----
0000688-0000691	hdisk10	4 frags	16384 Bytes,	0.3%	0000144-0000147
0000694-0001713	hdisk10	1020 frags	4177920 Bytes,	73.0%	0000150-0001169
0001716-0002089	hdisk10	374 frags	1531904 Bytes,	26.8%	0001172-0001545

```
1398 frags over space of 1402 frags: space efficiency = 99.7%
```

```
3 fragments out of 1398 possible: sequentiality = 99.9%
```

The output shows that this file uses double indirection for data block addresses. Both space efficiency and sequentiality are at very good levels (99.7 and 99.9 percent respectively).

Example 26-13 shows a file with zero sequentiality. It is a sparse file (explained in the next section), but the importance is the distance between the allocated blocks (0538324 and 0538325).

Example 26-13 Zero sequentiality

```
# fileplace -piv ugly.file
```

```
File: ugly.file Size: 512001 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 2 Compress: no
Inode: 182 Mode: -rw-r--r-- Owner: root Group: sys
```

INDIRECT BLOCK: 538338

Physical Addresses (mirror copy 1)				Logical Fragment
-----				-----
unallocated		12 frags	49152 Bytes, 0.0%	unallocated
0538324	hdisk1	1 frags	4096 Bytes, 50.0%	0001204
unallocated		112 frags	458752 Bytes, 0.0%	unallocated
0538325	hdisk1	1 frags	4096 Bytes, 50.0%	0001205

2 frags over space of 2 frags: space efficiency = 100.0%

2 fragments out of 2 possible: sequentiality = 0.0%

Sparsely allocated files

A file is a sequence of indexed blocks of arbitrary size. The indexing is accomplished through the use of direct mapping or indirect index blocks from the file inode as shown in Example 26-5 on page 487. Each index within a file's address range is not required to map to an actual data block.

A file that has one or more inode data block indexes that are not mapped to an actual data block is considered *sparsely allocated* or called a sparse file. A sparse file will have a size associated with it (in the inode), but it will not have all of the data blocks allocated that match this size.

A sparse file is created when an application extends a file by seeking a location outside the currently allocated indexes, but the data that is written does not occupy all of the newly assigned indexes. The new file size reflects the farthest write into the file.

A read to a section of a file that has unallocated data blocks results in a default value of null (0x00) bytes being returned. A write to a section of a file that has *unallocated* data blocks causes the necessary data blocks to be allocated and the data written, but there may not be enough free blocks in the file system any more. The result is that the write will fail. Database systems in particular maintain data in sparse files.

The problem with sparse files occurs first when unallocated space is needed for data being added to the file. Problems caused by sparse files can be avoided if the file system is large enough to accommodate all of the file's defined sizes, and of course to not have any sparse files in the file system.

It is possible to check for the existence of sparse files within a file system by using the **fileplace** command. Example 26-14 on page 493 shows how to use the **ls**, **du**, and **fileplace** commands to identify that a file is not sparse.

Example 26-14 Checking a file that is not sparse

```
# ls -l happy.file
-rw-r--r-- 1 root  sys           37 May 30 11:51 happy.file

# du -k happy.file
4      happy.file

# fileplace happy.file

File: happy.file Size: 37 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no

Logical Fragment
-----
0050663                               1 frags    4096 Bytes, 100.0%
```

The example output above shows that the size of the file `happy.file` is 37 bytes, but because the file system block (fragment) size is 4096 bytes and the smallest allocation size in a file system is one (1) block, `du` and `fileplace` show that the file actually uses 4 KB of disk space. Example 26-15 shows how the same type of report could look if the file was sparse.

Example 26-15 Checking a sparse file

```
# ls -l unhappy.file
-rw-r--r-- 1 root  sys          512037 May 30 11:55 unhappy.file

# du -k unhappy.file
4      unhappy.file

# fileplace unhappy.file

File: unhappy.file Size: 512037 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no

Logical Fragment
-----
unallocated                            125 frags   512000 Bytes, 0.0%
0050665                                1 frags    4096 Bytes, 100.0%
```

In the example output, the `ls -l` command shows the size information stored about the `unhappy.file` file in the file's *inode* record, which is the size in bytes (512037). The `du -k` command shows the number of allocated blocks for the file (in this case only one 4 KB block). The `fileplace` command shows how the blocks (Logical Fragments) are allocated. In the `fileplace` output above there are 125 unallocated blocks and one allocated at logical address 50665, so the `unhappy.file` file is sparse.

Creating a sparse file

To create a sparse file you can use the **dd** command with the **seek** option. In the following examples we show how to check the file system utilization during the process of creating a sparse file.

First we check the file system for our current directory with the **df** command to see how much apparent space is available. Note the number of inodes that are currently used (1659) to compare with the **df** output in Example 26-16.

Example 26-16 Using df

```
# df $PWD
Filesystem      512-blocks      Free %Used    Iused %Iused Mounted on
/dev/data1v    655360          393552   40%      1659     3% /data
```

Then we use the **dd** command to seek within one byte (-1 in the calculation in the Example 26-17) of the maximum allowed file size for our user. **ulimit -f** shows the current setting, in this case the default of 2097151 bytes or 1 GB). The input was just a new line character (\n) from the **echo** command. Now we have created a sparse file.

Example 26-17 Creating a sparse file

```
# echo|dd of=ugly.file seek=$((ulimit -f)-1)
0+1 records in.
0+1 records out.
```

Example 26-18 shows the examination of the file's space utilization with the **ls**, **fileplace**, and **df** commands. The first example below shows the output of the **ls** command that displays the file's inode byte counter. Note that the **-s** flag will report the actual number of KB blocks allocated, as does the **du** command.

Example 26-18 Using ls on the sparse file

```
# ls -sl ugly.file
4 -rw-r--r-- 1 root  sys      1073740801 May 31 17:13 /test2/ugly.file
```

According to the **ls** output in the previous example, the file size is 1073740801 bytes but only 4 (1 KB) blocks. Now we know that this is a sparse file. In Example 26-19 we use the **fileplace -l** command to look at the allocation in detail, first from a logical view.

Example 26-19 Using fileplace -l on the sparse file

```
# fileplace -l ugly.file
```

```
File: ugly.file  Size: 1073740801 bytes  Vol: /dev/lv09
Blk Size: 4096  Frag Size: 4096  Nfrags: 1  Compress: no
```

```

Logical Fragment
-----
unallocated                262143 frags 1073737728 Bytes,  0.0%
0000014                    1 frags    4096 Bytes, 100.0%

```

The logical report above shows that logical block 14 is allocated for the file occupying 4 KB, and the rest is unallocated. Example 26-20 shows the physical view of the file using the **fileplace -p** command.

Example 26-20 Using fileplace -p on the sparse file

```
# fileplace -p ugly.file
```

```
File: ugly.file Size: 1073740801 bytes Vol: /dev/lv09
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no
```

Physical Addresses (mirror copy 1)		Logical Fragment	
-----		-----	
unallocated		262143 frags 1073737728 Bytes, 0.0%	unallocated
0631342	hdisk1	1 frags 4096 Bytes, 100.0%	0000014

The physical report shows that physical block 631342 is allocated for the logical block 14 and it resides on hdisk1. Example 26-21 shows the volume report (logical view) for the file using the **fileplace -v** command.

Example 26-21 Using fileplace -lv on the sparse file

```
# fileplace -lv ugly.file
```

```
File: ugly.file Size: 1073740801 bytes Vol: /dev/lv09
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no
Inode: 18 Mode: -rw-r--r-- Owner: root Group: sys
```

```

Logical Fragment
-----
unallocated                262143 frags 1073737728 Bytes,  0.0%
0000014                    1 frags    4096 Bytes, 100.0%

```

```

1 frags over space of 1 frags: space efficiency = 100.0%
1 fragment out of 1 possible: sequentiality = 100.0%

```

The volume report, for the logical view, shows that the file has 100 percent space efficiency and sequentiality. The next and final **fileplace** command report on this file (in Example 26-22 on page 496) shows the volume report for the physical view of the file.

Example 26-22 Using fileplace -pv on the sparse file

```
# fileplace -pv ugly.file
```

```
File: ugly.file Size: 1073740801 bytes Vol: /dev/lv09
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no
Inode: 18 Mode: -rw-r--r-- Owner: root Group: sys
```

```
Physical Addresses (mirror copy 1)                Logical Fragment
-----
unallocated                262143 frags 1073737728 Bytes,  0.0%  unallocated
0631342                    1 frags   4096 Bytes, 100.0%  0000014

1 frags over space of 1 frags: space efficiency = 100.0%
1 fragment out of 1 possible: sequentiality = 100.0%
```

The volume report above, for the physical view, also shows that the file has 100 percent space efficiency and sequentiality.

Sparse files in large file enabled file systems

File data in a large file enabled file system (after the file size has increased over 4 MBs) will use 32 contiguous 4 KB blocks (so-called large disk blocks) as opposed to one 4 KB block for a normal JFS file system. To illustrate the point, we will show a series of examples using the **fileplace** command to examine the allocation of a file. First we verify that the file system is a large file system with the **lsfs** command, then we create a file without data, and finally we examine the inode information with the **ls** command and then the block allocation with the **fileplace** command as shown in Example 26-23.

Example 26-23 Creating a file in a large file-enabled file system

```
# lsfs -cq $PWD|tail -1
(lv size 655360:fs size 655360:frag size 4096:nbpi 4096:compress no:bf true:ag 64)
# >ugly.file
# ls -sl ugly.file
 0 -rw-r--r--  1 root   sys           0 May 31 17:59 ugly.file
# fileplace ugly.file

File: ugly.file Size: 0 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 0 Compress: no

Logical Fragment
-----
```

In the example above we see that it is indeed a large file enabled file system because **bf** is true, the **ls** command shows zero blocks allocated, and the **Size** is zero bytes as well. The **fileplace** command shows that the size is zero and that

there are no blocks allocated. Now we seek 4 MB (4194304 bytes) to the new end of the file and examine it again with the **ls** and **fileplace** commands as shown in Example 26-24.

Example 26-24 Seeking 4 MB to the end of file

```
# dd if=/dev/null of=ugly.file bs=1 seek=4194304
0+0 records in.
0+0 records out.
# ls -sl ugly.file
 4 -rw-r--r-- 1 root    sys          4194304 May 31 18:14 ugly.file
# fileplace ugly.file

File: ugly.file Size: 4194304 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 1 Compress: no

Logical Fragment
-----
unallocated          1023 frags  4190208 Bytes,  0.0%
0001205              1 frags    4096 Bytes, 100.0%
```

In the output above the **ls** command shows four blocks allocated, and that the size is 4 MB. The **fileplace** command shows that the size is 4 MB and that four blocks (1 KB per block) is allocated. Now we add one byte to the file and examine it again, as shown in Example 26-25.

Example 26-25 File size after adding one byte

```
# echo >>ugly.file

# ls -sl ugly.file
132 -rw-r--r-- 1 root    sys          4194305 May 31 18:19 ugly.file
# fileplace ugly.file

File: ugly.file Size: 4194305 bytes Vol: /dev/data1v
Blk Size: 4096 Frag Size: 4096 Nfrags: 33 Compress: no

Logical Fragment
-----
unallocated          1023 frags  4190208 Bytes,  0.0%
0001205              1 frags    4096 Bytes,  3.0%
0001218             32 frags   131072 Bytes, 97.0%
```

In the output above the **ls** command shows 132 blocks (1 KB per block) allocated, and that the size is 4 MB and one byte. The **fileplace** command shows that the size is 4 MB and one byte, and that there are 33 blocks (4 KB per block) allocated. The byte we added to the file has caused 32 blocks (4 KB per block) to be added because it is a large file system.

Searching for sparse files

To find sparse files in file systems we can use the **find** command with the **-ls** flag. Example 26-26 shows how this can be done.

Example 26-26 Using find to find sparse files

```
root@wlmhost:/data: find /test0 -type f -xdev -ls
 17   4 -rw-r--r--  1 root    sys           1 May 31 12:23 /test0/file
 18   4 -rw-r--r--  1 root    sys       1073740801 May 31 17:13 /test0/ugly.file
```

The second column is the allocated block size, the seventh column is the byte size and the 11th column is the file name. In the output above it is obvious that this will be time consuming if done manually because the **find** command lists all files by using the **-type f** flag. Because we cannot limit the output further by only using the **find** command, we do it with a script.

The script in Example 26-27 takes as an optional parameter the file system to scan. If no parameter is given, it will list all file systems in the system with the **lsfs** command (except **/proc**) and stores this in the **fs** variable. The **find** command, on the last line in the script, searches all file systems specified in the **fs** variable for files (**-type f**), does not traverse over file system boundaries (**-xdev**), and lists inode information about the file (**-ls**). The output from the **find** command is then examined by **awk** in the pipe. The **awk** command compares the sizes of a normalized block and byte value and, if they do not match, **awk** will print the filename, block, and byte sizes.

Example 26-27 Shell script to search for sparse files

```
:
fs=${1:-"${lsfs -c|awk -F: 'NR>2&&!/\proc/{print $1}'}"}
find $fs -xdev -type f -ls 2>/dev/null|awk '{if (int($2*1024)<int($7/1024)) print $11,$2,$7}'
```

The **awk** built in **int()** function is used because **awk** returns floating point values as the result of calculations, and the comparison should be done with integers. Example 26-28 is sample output from running the script above.

Example 26-28 Sample output from sparse file search script

```
/home/mysp1 4 512000001
/tmp/mysp 4 512000001
...(lines omitted)...
/tmp/ugly.file 4 1073740801
/data/mysp3 128 1073740801
/test0/ugly.file 4 1073740801
```

To find out how many sparse files the script found, just pipe the output to the `wc` command with `-l` flag, or change the script to perform this calculation as well (it was not included above for readability), as Example 26-29 shows.

Example 26-29 Enhanced shell script to search for sparse files

```
:
fs=${1:-"$(lsfs -c|awk -F: 'NR>2&&!/\|/proc/{print $1}')}
find $fs -xdev -type f -ls 2>/dev/null|
awk 'BEGIN{n=0}
     {if (int($2*1024)<int($7/1024)) {print $11,$2,$7;n++}}
     END{print "\nTotal no of sparse files",n}'
```

The variable `n` is incremented each time a file matching the calculation is found. The sample output in Example 26-30 shows on the last line how many sparse files the script found (110).

Example 26-30 Sample output from the enhanced sparse file search script

```
...(lines omitted)...
/test0/ugly.file 4 1073740801
```

Total no of sparse files **110**

Displaying logical to physical map for a logical volume

Example 26-31 shows the use of `-m` flag to display logical to physical map of a logical volume.

Example 26-31 Using `-m` flag

```
# fileplace -m /dev/hd2
Device: /dev/hd2   Partition Size: 32 MB   Block Size = 4096
Number of Partitions: 149   Number of Copies: 1
```

Physical Addresses (mirror copy 1)	Logical Fragment
-----	-----
1794592-1810975 hdisk0 16384 blocks 67108864 Bytes, 1.3%	0000000-0016383
1843744-2286111 hdisk0 442368 blocks 1811939328 Bytes, 36.2%	0016384-0458751
2384416-2638367 hdisk0 253952 blocks 1040187392 Bytes, 20.8%	0458752-0712703
1286688-1655327 hdisk0 368640 blocks 1509949440 Bytes, 30.2%	0712704-1081343
2662944-2802207 hdisk0 139264 blocks 570425344 Bytes, 11.4%	1081344-1220607

The `lslv`, `lspv`, and `lsvg` commands

Many times it is useful to determine the layout of logical volumes on disks and volume groups to identify whether rearranging or changing logical volume definitions might be appropriate. Some of the commands that can be used are `lslv`, `lspv`, and `lsvg`:

- ▶ The `lslv` command displays the characteristics and status of the logical volume.
- ▶ The `lspv` command is useful for displaying information about the physical volume, its logical volume content, and the logical volume allocation layout.
- ▶ The `lsvg` command displays information about volume groups.

The `lslv`, `lsvg`, and `lspv` commands read different Logical Volume Manager (LVM) volume groups and logical volume descriptor areas from physical volumes.

When information from the Object Data Manager (ODM) Device Configuration database is unavailable, some of the fields will contain a question mark (?) in place of the missing data.

These commands reside in `/usr/sbin` and are part of the `bos.rte.lvm` fileset, which is installed by default from the AIX base installation media.

27.1 lslv

The syntax of the **lslv** command is:

```
lslv [ -L ] [ -l | -m ] [ -n DescriptorPV ] LVname  
lslv [ -L ] [ -n DescriptorPV ] -p PVname [ LVname ]
```

Flags

- L** Specifies no waiting to obtain a lock on the volume group. If the volume group is being changed, using the **-L** flag gives unreliable data.
- l** Lists the following fields for each physical volume in the logical volume: PV, Copies, In band, Distribution
- m** Lists the following fields for each logical partition: LPs, PV1, PP1, PV2, PP2, PV3, PP3
- n PhysicalVolume** Accesses information from the specific descriptor area of the PhysicalVolume variable. The information may not be current because the information accessed with the **-n** flag has not been validated for the logical volumes. If you do not use the **-n** flag, the descriptor area from the physical volume that holds the validated information is accessed and therefore the information that is displayed is current. The volume group need not be active when you use this flag.
- p PhysicalVolume** Displays the logical volume allocation map for the PhysicalVolume variable. If you use the LogicalVolume parameter, any partition allocated to that logical volume is listed by logical partition number.

Parameters

- LogicalVolume** The logical volume to examine.

27.2 lspv

The syntax of the **lspv** command is:

```
lspv [-L] [-M | -l | -p] [-n DescriptorPV] [-v VGid] PVname
```

Flags

- L** Specifies no waiting to obtain a lock on the volume group. Note that if the volume group is

- being changed, using the -L flag gives unreliable data.
- l** Lists the following fields for each logical volume on the physical volume: LVname, LPs, PPs, Distribution, Mount Point
 - M** Lists the following fields for each logical volume on the physical volume: PVname, PPnum, LVname, LPnum, Copynum, PPstate
 - n DescriptorPhysicalVolume** Accesses information from the variable descriptor area specified by the DescriptorPhysicalVolume variable. The information may not be current because the information accessed with the -n flag has not been validated for the logical volumes. If you do not use the -n flag, the descriptor area from the physical volume that holds the validated information is accessed, and therefore the information displayed is current. The volume group does not have to be active when you use this flag.
 - p** Lists the following fields for each physical partition on the physical volume: Range, State, Region, LVname, Type, Mount point
 - v VolumeGroupID** Accesses information based on the volume groupID variable. This flag is needed only when the **lspv** command does not function due to incorrect information in the Device Configuration Database. The volume groupID variable is the hexadecimal representation of the volume group identifier, which is generated by the **mkvg** command.

Parameters

PhysicalVolume

The physical volume to examine.

27.3 lsvg

The syntax of the **lsvg** command is:

```
lsvg [-o] [[-L] -n PVname] | -p ] volume group ...
lsvg [-L] [-i] [-M | -l | -p] VGname...
```

Flags

- L** Specifies no waiting to obtain a lock on the volume group. If the volume group is being changed, using the -L flag gives unreliable data.
- p** Lists the following information for each physical volume within the group specified by the volume group parameter: Physical volume, PVstate, Total PPs, Free PPs, Distribution
- l** Lists the following information for each logical volume within the group specified by the volume group parameter: LV, Type, LPs, PPs, PVs, Logical volume state, Mount point
- i** Reads volume group names from standard input.
- M** Lists the following fields for each logical volume on the physical volume: PVname, PPnum, LVname, LPnum, Copynum, PPstate
- n DescriptorPhysicalVolume** Accesses information from the descriptor area specified by the DescriptorPhysicalVolume variable. The information may not be current because the information accessed with the -n flag has not been validated for the logical volumes. If you do not use the -n flag, the descriptor area from the physical volume that holds the most validated information is accessed, and therefore the information displayed is current. The volume group need not be active when you use this flag.
- o** Lists only the active volume groups (those that are varied on). An active volume group is one that is available for use.

Parameters volume group

The name of the volume group to examine.

27.4 Examples for Islv, lspv, and lsvg

When starting to look for a potential I/O-related performance bottleneck, we often need to find out more about the disks in use, such as their content and purpose. Here are a few of the actions we need to perform:

- ▶ Determine the volume group the disks in question belong to.
- ▶ Determine the logical volume layout on the disks in question.
- ▶ Determine the logical volume layout of all of the disks in question on the volume group.

To accomplish this we use mainly the **lsvg**, **lspv**, and **lslv** commands.

To monitor disk I/O we usually start with the **iostat** command (see Chapter 4, “The iostat command” on page 81), which shows the load on different disks in great detail. The output in Example 27-1 is the summary since boot time (if the **iostat** attribute has been enabled for the sys0 logical device driver).

Example 27-1 Starting point with iostat

```
# iostat -ad
```

Adapter:	Kbps	tps	Kb_read	Kb_wrtn
scsi0	21.1	3.6	6018378	4343544

Paths/Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn
hdisk1_Path0	0.0	0.2	0.0	103951	2004
hdisk0_Path0	1.0	20.1	3.4	5534703	4341540
cd0	0.0	0.8	0.2	379724	0

Adapter:	Kbps	tps	Kb_read	Kb_wrtn
scsi1	71.3	7.6	21588850	13463040

Paths/Disks:	% tm_act	Kbps	tps	Kb_read	Kb_wrtn
hdisk2_Path0	2.1	38.5	3.4	12226787	6695708
hdisk3_Path0	3.1	32.8	4.2	9362063	6767332

This system has two SCSI adapters and two disks on each adapter. Since IPL the disks have not been very active. To find out how long the statistics have been gathering, use the uptime command as shown in Example 27-2.

Example 27-2 Using uptime

```
# uptime
```

11:57AM up 5 days, 1:13, 11 users, load average: 0.00, 0.00, 0.00

The example tells us that the statistics have been collected over five days. Also note that the output of `iostat` will show an average over 24 hours during that time. We know that our system is only used during normal working hours so we could check the current running statistics as in Example 27-3.

Example 27-3 Using iostat

```
# iostat -ad 1 2
...(lines omitted)...
Adapter:                Kbps      tps      Kb_read  Kb_wrtn
scsi0                   0.0      0.0      0        0

Paths/Disks:           % tm_act  Kbps      tps      Kb_read  Kb_wrtn
hdisk1_Path0           0.0      0.0      0.0      0        0
hdisk0_Path0           0.0      0.0      0.0      0        0
cd0                    0.0      0.0      0.0      0        0

Adapter:                Kbps      tps      Kb_read  Kb_wrtn
scsi1                  1834.2    192.8    1720     316

Paths/Disks:           % tm_act  Kbps      tps      Kb_read  Kb_wrtn
hdisk2_Path0           47.7     1228.8    97.3     1260     104
hdisk3_Path0           61.3     605.4     95.5     460      212
```

And now we see that the system performs quite a bit of I/O on `hdisk1`, so we should check how the layout is for these disks. First let's find out what volume groups the disks belong to in Example 27-4.

Example 27-4 Using lspv to examine the disk versus volume group mapping

```
# lspv
hdisk0      000bc6adc9ee6b3a      rootvg      active
hdisk1      000bc6ade881de45      vg0         active
hdisk2      000bc6adc472a478      vg0         active
hdisk3      000bc6adc9ec9be3      vg0         active
```

The disks we are examining (`hdisk2` and `hdisk3`) belong to the `vg0` volume group. Because the two disks belongs to the same volume group, we can go ahead and list some information about the disks from the volume group perspective using `lsvg` as shown in Example 27-5.

Example 27-5 Using lsvg to check the distribution

```
# lsvg -p vg0
vg0:
PV_NAME      PV STATE      TOTAL PPs  FREE PPs  FREE DISTRIBUTION
hdisk1       active        542        509      109..75..108..108..109
hdisk2       active        542        397      47..25..108..108..109
hdisk3       active        542        397      47..25..108..108..109
```

Now we see that the disks have the same number of physical partitions, and because volume groups have one physical partition size, they must be of the same size.

The **lsvg -p** fields are interpreted as follows:

PV_NAME	A physical volume within the group.
PV STATE	State of the physical volume.
TOTAL PPs	Total number of physical partitions on the physical volume.
FREE PPs	Number of free physical partitions on the physical volume.
FREE Distribution	The number of physical partitions allocated within each section of the physical volume: outer edge, outer middle, center, inner middle, and inner edge of the physical volume.

Now we can find out which logical volumes occupy the vg0 volume group, as shown in Example 27-6.

Example 27-6 Using lsvg to get all logical volumes within the volume group

```
# lsvg -l vg0
vg0:
LV NAME      TYPE      LPs  PPs  PVs  LV STATE    MOUNT POINT
lv03         jfs2log   1    1    1    open/syncd  N/A
lv04         jfs2      62   62   1    open/syncd  /work/fs1
lv05         jfs2      62   62   1    open/syncd  /work/fs2
lv06         jfs       62   124  2    closed/syncd N/A
lv07         jfs       63   63   3    closed/syncd N/A
data1v       jfs       10   10   1    open/syncd  /data
loglv00      jfslog    1    1    1    open/syncd  N/A
```

This tells us that there are both JFS and JFS2 filesystems, a couple of logical volumes without entries in `/etc/filesystems` (the mount point show up as N/A), and that one logical volume is mirrored (lv06) and one logical volume is spread over three disks (lv07). The output above also shows that we have two external log logical volumes; lv03 that is used by JFS2 file systems and loglv00 that is used by JFS file systems. The report does not tell us which of the file systems uses which log logical volume, nor if any of them uses inline logs either.

The **lsvg -l** report has the following format:

LV NAME	A logical volume within the volume group.
TYPE	Logical volume type.
LPs	Number of logical partitions in the logical volume.

PPs	Number of physical partitions used by the logical volume.
PVs	Number of physical volumes used by the logical volume.
LV STATE	State of the logical volume. <code>Opened/stale</code> indicates that the logical volume is open but contains partitions that are not current. <code>Opened/syncd</code> indicates that the logical volume is open and synchronized. <code>Closed</code> indicates that the logical volume has not been opened.
MOUNT POINT	File system mount point for the logical volume, if applicable.

At this point it would be a good idea to check which of the file systems are the most used with the **filemon** (Chapter 25, “The filemon command” on page 457) or **lvmstat** (Chapter 28, “The lvmstat command” on page 519) commands. For instance, Example 27-7 with **lvmstat** shows the five busiest logical volumes.

Example 27-7 Checking busy logical volumes with lvmstat

```
# lvmstat -v vg0 -c 5
```

Logical Volume	iocnt	Kb_read	Kb_wrtn	Kbps
lv05	2073116	7886628	5052576	25.91
lv04	1592894	9036912	4985908	28.08
lv03	2	0	8	0.00
loglv00	0	0	0	0.00
datav	0	0	0	0.00

We can clearly see that both `lv04` and `lv05` are the most utilized logical volumes. Now we need to get more information about the layout on the disks. If the workload shows a significant degree of I/O dependency (although it has a lot of I/O we cannot conclude the complete workload from the **iostat** or **lvmstat** output only), we can investigate the physical placement of the files on the disk to determine whether reorganization at some level would yield an improvement. To view the placement of the partitions of logical volume `lv04` within physical volume `hdisk2`, the **lslv** command could be used as shown in Example 27-8.

Example 27-8 Using lslv -p

```
# lslv -p hdisk2 lv04
```

```
hdisk2:lv04:/work/fs1
```

USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	1-10
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	11-20
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	21-30
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	31-40
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	41-50
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	51-60

USED	USED	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	61-70
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	71-80
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	81-90
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	91-100
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	101-109
0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	110-119
0011	0012	0013	0014	0015	0016	0017	0018	0019	0020	120-129
0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	130-139
0031	0032	0033	0034	0035	0036	0037	0038	0039	0040	140-149
0041	0042	0043	0044	0045	0046	0047	0048	0049	0050	150-159
0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	160-169
0061	0062	USED	USED	USED	USED	USED	USED	USED	USED	170-179
USED	USED	USED	USED	USED	USED	USED	USED	USED	USED	180-189
USED	USED	USED	FREE	FREE	FREE	FREE	FREE	FREE	FREE	190-199
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	200-209
FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	FREE	210-217

...(lines omitted)...

The USED label tells us that this partition is allocated by another logical volume, the FREE label tells us that it is not allocated, and the numbers 0001-0062 indicate that this belongs to the logical volume we wanted to check, in our case 1v04. A STALE partition (not shown in the example above) is a physical partition that contains data you cannot use.

Example 27-9 shows a similar output from **lspv** to find out the intra disk layout of logical volumes on **hdisk2** and **hdisk3**.

Example 27-9 Using lspv to check the intra disk policy

```
# lspv -l hdisk2;lspv -l hdisk3
hdisk2:
LV_NAME          LPs  PPs  DISTRIBUTION      MOUNT POINT
1v06              62   62   62..00..00..00..00  N/A
1v04            62   62   00..62..00..00..00  /work/fs1
1v07              21   21   00..21..00..00..00  N/A
hdisk3:
LV_NAME          LPs  PPs  DISTRIBUTION      MOUNT POINT
1v06              62   62   62..00..00..00..00  N/A
1v05            62   62   00..62..00..00..00  /work/fs2
1v07              21   21   00..21..00..00..00  N/A
```

Each of our hot file systems is allocated on a separate disk and on the same part of the disks, and is contiguously allocated there. Example 27-10 on page 510 shows the intra disk layout in another, more readable, way with the **lspv** command.

Example 27-10 Using lspv to check the intra disk layout

```
# lspv -p hdisk2;lspv -p hdisk3
hdisk2:
PP RANGE STATE REGION LV ID TYPE MOUNT POINT
  1-62 used outer edge lv06 jfs N/A
  63-109 free outer edge
110-171 used outer middle lv04 jfs2 /work/fs1
 172-192 used outer middle lv07 jfs N/A
 193-217 free outer middle
 218-325 free center
 326-433 free inner middle
 434-542 free inner edge
hdisk3:
PP RANGE STATE REGION LV NAME TYPE MOUNT POINT
  1-62 used outer edge lv06 jfs N/A
  63-109 free outer edge
110-171 used outer middle lv05 jfs2 /work/fs2
 172-192 used outer middle lv07 jfs N/A
 193-217 free outer middle
 218-325 free center
 326-433 free inner middle
 434-542 free inner edge
```

The output above shows us the same information. If we had a fragmented layout for our logical volumes this would have meant that the disk arms would have to move across the disk platter whenever the end of the first part of the logical volume was reached. This is usually the case when file systems are expanded during production and this is an excellent feature of Logical Volume Manager Device Driver (LVMDD). After some time in production, the logical volumes must be reorganized so that they occupy contiguous physical partitions. We can also examine how the logical volume partitions are organized with the `lslv` command. Example 27-11 shows a quick look at the two log logical volumes.

Example 27-11 Using lslv to check the logical volume disk layout

```
# lslv -m lv04;lslv -m lv05
lv04:/work/fs1
LP PP1 PV1 PP2 PV2 PP3 PV3
0001 0110 hdisk2
0002 0111 hdisk2
...(lines omitted)...
0061 0170 hdisk2
0062 0171 hdisk2
lv05:/work/fs2
LP PP1 PV1 PP2 PV2 PP3 PV3
0001 0110 hdisk3
0002 0111 hdisk3
...(lines omitted)...
```

```
0061 0170 hdisk3
0062 0171 hdisk3
```

The output simply shows what physical partitions are allocated for each logical partition. In a more complex allocation it can be most useful to check the locations used for different very active logical volumes, compare where they are allocated on the disk, and, if possible, move the hot spots closer together. Example 27-12 shows how the logical partitions are mapped against the physical partitions on the disks for the two logical volumes (1v04 and 1v05).

Example 27-12 Using `lslv` to check the logical volume partition allocation

```
# lslv -m 1v04; lslv -m 1v05
1v04:/work/fs1
LP   PP1  PV1                PP2  PV2                PP3  PV3
0001 0110 hdisk2
0002 0111 hdisk2
...(lines omitted)...
0072 0202 hdisk2
0073 0203 hdisk2
1v05:/work/fs2
LP   PP1  PV1                PP2  PV2                PP3  PV3
0001 0110 hdisk3
0002 0111 hdisk3
...(lines omitted)...
0071 0201 hdisk3
0072 0202 hdisk3
```

The output tells us that the physical partitions are contiguous, there is only one physical partition (PV1) for each logical partition (LP), and each logical volume has all of its physical partitions on a single disk each (PV1).

The `lslv -m` report has the following format:

LPs	Logical partition number.
PV1	Physical volume name where the logical partition's first physical partition is located.
PP1	First physical partition number allocated to the logical partition.
PV2	Physical volume name where the logical partition's second physical partition (first copy) is located.
PP2	Second physical partition number allocated to the logical partition.
PV3	Physical volume name where the logical partition's third physical partition (second copy) is located.

PP3 Third physical partition number allocated to the logical partition.

When looking at the two log volumes, lv03 and loglv00 in Example 27-13, we know that they both use only one physical partition. This could be a good allocation for each log logical volume, but it depends on where they are allocated.

Example 27-13 Using lslv to check the logical volumes partition distribution

```
# lslv -l lv03;lslv -l loglv00
lv03:N/A
PV          COPIES      IN BAND      DISTRIBUTION
hdisk1      001:000:000  100%         000:001:000:000:000
loglv00:N/A
PV          COPIES      IN BAND      DISTRIBUTION
hdisk1      001:000:000  100%         000:001:000:000:000
```

Each log volume is properly allocated (100% IN BAND). This is simple because each log logical volume only consists of one physical and logical partition in this example. However, if this value is less than 100 percent, reorganization should be in order. But they are a bit apart (physical partition 110 and 142) and each time a JFS and J2 file system changes meta data, each log logical volume will have to be updated, causing the disk arm to move from the log logical volume to the file system and back to the log logical volume.

To continue examining the layout for our hot logical volumes lv04 and lv05, now would be a good time to check what is going on in the file system. For this we need to use `filemon` (Chapter 25, “The filemon command” on page 457) and perhaps `fileplace` (Chapter 26, “The fileplace command” on page 479).

27.4.1 Using lslv

The `lslv` command displays the characteristics and status of the logical volume, as Example 27-14 shows.

Example 27-14 Logical volume fragmentation with lslv

```
# lslv -l hd6
hd6:N/A
PV          COPIES      IN BAND      DISTRIBUTION
hdisk0      288:000:000  37%          000:108:108:072:000
```

As can be seen above, the `lspv` and `lslv` show the same distribution for the logical volume hd6. The `lslv` command also shows that it has 288 LPs but no additional copies. It also says that the intra-policy of center is only 37 % in band, which means that 63 % is out of band (that is, not in the center).

The `lslv -l` report has the following format:

PV	Physical volume name.
COPIES	These three fields are displayed: <ul style="list-style-type: none">– The number of logical partitions containing at least one physical partition (no copies) on the physical volume– The number of logical partitions containing at least two physical partitions (one copy) on the physical volume– The number of logical partitions containing three physical partitions (two copies) on the physical volume
IN BAND	The percentage of physical partitions on the physical volume that belong to the logical volume and were allocated within the physical volume region specified by intra-physical allocation policy.
DISTRIBUTION	The number of physical partitions allocated within each section of the physical volume. The DISTRIBUTION shows how the physical partitions are placed in each part of the intrapolicy; that is: edge : middle : center : inner-middle : inner-edge

The higher the IN BAND percentage, the better the allocation efficiency. Each logical volume has its own intra policy. If the operating system cannot meet this requirement, it chooses the best way to meet the requirements.

27.4.2 Using `lspv`

The `lspv` command is useful for displaying information about the physical volume, its logical volume content, and logical volume allocation layout, as Example 27-15 shows.

Example 27-15 Logical volume fragmentation with `lspv -l`

```
# lspv -l hdisk0
hdisk0:
LV NAME          LPs   PPs   DISTRIBUTION      MOUNT POINT
hd5              1     1     01..00..00..00   N/A
hd6             288   288   00..108..108..72..00 N/A
```

This example shows that the `hd6` logical volume is nicely placed in the center area of the disk, the distribution being 108 logical partitions in the center, 108 logical partitions in the outer middle, and 72 logical partitions in the inner middle part of the disk.

The **lspv -l** report has the following format:

LV NAME	Name of the logical volume to which the physical partitions are allocated.
LPs	The number of logical partitions within the logical volume that are contained on this physical volume.
PPs	The number of physical partitions within the logical volume that are contained on this physical volume.
DISTRIBUTION	The number of physical partitions belonging to the logical volume that are allocated within each of the following sections of the physical volume: outer edge, outer middle, center, inner middle, and inner edge of the physical volume.
MOUNT POINT	File system mount point for the logical volume, if applicable.

Another way to use **lspv** is with the **-p** parameter as in Example 27-16.

Example 27-16 Logical volume fragmentation with lspv -p

```
# lspv -p hdisk0
hdisk0:
PP RANGE STATE REGION LV NAME TYPE MOUNT POINT
  1-1 used outer edge hd5 boot N/A
  2-109 free outer edge
110-217 used outer middle hd6 paging N/A
218-325 used center hd6 paging N/A
326-397 used inner middle hd6 paging N/A
398-433 free inner middle
434-542 free inner edge
```

As shown in the output above, this output is easier to read.

The **lspv -p** report has the following format:

PP RANGE	A range of consecutive physical partitions contained on a single region of the physical volume.
STATE	The current state of the physical partitions; free, used, stale, or vgda.
REGION	The intra-physical volume region in which the partitions are located.
LV ID	The name of the logical volume to which the physical partitions are allocated.
TYPE	The type of the logical volume to which the partitions are allocated.
MOUNT POINT	File system mount point for the logical volume, if applicable.

27.4.3 Using lsvg

The `lsvg` command is useful for displaying information about the volume group and its logical and physical volumes.

First we need to understand the basic properties of the volume group, such as:

- ▶ Its general characteristics
- ▶ Its currently allocated size
- ▶ Its physical partition size
- ▶ Whether there are any STALE partitions
- ▶ How much space is already allocated
- ▶ How much is not allocated

Example 27-17 shows how to obtain this basic information about a volume group.

Example 27-17 Using lsvg to obtain volume group basics

```
# lsvg -L datavg
VOLUME GROUP:  datavg                VG IDENTIFIER:
0021768a00004c00000000f
44fe55821
VG STATE:      active                 PP SIZE:      32 megabyte(s)
VG PERMISSION: read/write            TOTAL PPs:    542 (17344 megabytes)
MAX LVs:       256                   FREE PPs:     312 (9984 megabytes)
LVs:           7                      USED PPs:     230 (7360 megabytes)
OPEN LVs:      6                      QUORUM:       2
TOTAL PVs:     1                      VG DESCRIPTORS: 2
STALE PVs:     0                      STALE PPs:    0
ACTIVE PVs:    1                      AUTO ON:      yes
MAX PPs per PV: 1016                 MAX PVs:      32
LTG size:      128 kilobyte(s)       AUTO SYNC:    no
HOT SPARE:     no                     BB POLICY:    relocatable
```

The volume group shown in the example has six logical volumes and one disk with a physical partition size of 32 MB.

We also need to find out which logical volumes are created on this volume group and if they all are open and in use as shown in Example 27-18. If they are not open and in use they might be old, corrupted and forgotten, or only used occasionally, and if we were to need more space to reorganize the volume group we might be able to free that space.

Example 27-18 Using lsvg to check the logical volume state

```
# lsvg -l datavg
datavg:
LV NAME      TYPE      LPs  PPs  PVs  LV STATE  MOUNT POINT
loglv00     jfslog    1    1    1  open/syncd  N/A
lv00        jfs       32   32   1  open/syncd  /home/db2inst1
```

lv01	jfs	63	63	1	open/syncd	/install
lv02	jfs	4	4	1	open/syncd	/home/db2as
lv03	jfs	1	1	1	open/syncd	/home/db2fenc1
loglv01	jfs2log	1	1	1	open/syncd	N/A
lv05	jfs	160	160	1	open/syncd	/bigfs
fslv00	jfs2	19	19	1	open/syncd	/jfs2

As the example above shows, there are six logical volumes with file systems and two different types of jfslog for each kind of jfs allocated in this volume group. We can have two types of jfs: a journal file system or an Enhanced Journaled File System (JFS2). For further information, refer to *AIX 5L Version 5.2, System Management Guide: Operating System and Devices*.

Remember that the physical partition size was 32 MB, so even though the logs logical volume only has one (1) logical partition it is a 32 MB partition. Example 27-19 shows the disks that are allocated for this volume group.

Example 27-19 Using lsvg to determine disks allocated to the volume group

```
# lsvg -p datavg
datavg:
PV_NAME          PV STATE          TOTAL PPs   FREE PPs   FREE DISTRIBUTION
hdisk1           active            542         261        90..00..00..62..109
```

So there is only one disk in this volume group and mirroring is not activated for the logical volumes. When finding out information about volume groups it is often necessary to know what kind of disks are being used to make up the volume group. To examine disks we can use the **lspv**, **lsdev**, and **lscfg** commands.

27.4.4 Acquiring more disk information

Example 27-20 uses the **lsdev** command to obtain information about the types of disks in the volume group.

Example 27-20 Using lsdev to examine a disk device

```
# lsdev -Cl hdisk6
hdisk6 Available 10-70-L      SSA Logical Disk Drive
```

The output tells us that it is an SSA logical disk. Example 27-21 shows the **ssaxlate** command to find out which physical disks belong to the logical disk.

Example 27-21 Using the ssaxlate command

```
# ssaxlate -l hdisk6
pdisk0 pdisk2 pdisk1 pdisk3
```

This shows that the logical disk `hdisk6` is composed of four physical disks (`pdisk0-3`) and could be some sort of SSA RAID configuration (the *hdisks* consists of more than one *pdisk*). To find out, we used the `ssaraid` command as in Example 27-22.

Example 27-22 Using ssaraid to check the logical disk

```
# ssaraid -M|xargs -i ssaraid -l {} -Ihz -n hdisk6
#name      id              state              size
hdisk6    156139E312C44C0 good              36.4GB RAID-10 array
```

The output confirms that it is a RAID-defined disk. If it had not been, the output would have looked similar to Example 27-23.

Example 27-23 Using ssaraid to check the logical disk

```
# ssaraid -M|xargs -i ssaraid -l {} -Ihz -n hdisk6
#name      id              use              member_stat size
pdisk5    000629D465DC00D system          n/a          9.1GB Physical disk
```

To find all SSA-configured RAID disks controlled by SSA RAID managers in the system, run the `ssaraid` command as shown in Example 27-24.

Example 27-24 More examples of the use of ssaraid

```
# ssaraid -M|xargs -i ssaraid -l {} -Ihz
#name      id              use              member_stat size
pdisk0     000629D148ED00D member          n/a          18.2GB Physical disk
pdisk1     000629D2781600D member          n/a          18.2GB Physical disk
pdisk2     000629D278C500D member          n/a          18.2GB Physical disk
pdisk3     000629D282C500D member          n/a          18.2GB Physical disk
hdisk6    156139E312C44C0 good              36.4GB RAID-10 array
```

In the example above only `hdisk6` is a RAID-defined disk; the other `pdisks` are only used as Just a Bunch Of Disks (JBODs).

Archived

The `lvmstat` command

The `lvmstat` command reports input and output statistics for logical partitions, logical volumes, and volume groups. `lvmstat` is useful in determining whether a physical volume is becoming a hindrance to performance by identifying the busiest physical partitions for a logical volume.

`lvmstat` can help identify particular logical volume partitions that are used more than other partitions (hot spots or high-traffic partitions). If these partitions reside on the same disk or are spread out over several disks, it may be necessary to migrate them to new disks or, when the volume group only has one disk, put them closer together on the same disk to reduce the performance penalty.

The `lvmstat` command resides in `/usr/sbin` and is part of the `bos.rte.lvm` fileset, which is installed by default from the AIX base installation media.

28.1 lvmstat

The syntax of the **lvmstat** command is:

```
lvmstat {-l|-v} <name> [-e|-d] [-F] [-C] [-c count] [-s] [interval [iterations]]
```

Flags

- c Count** Prints only the specified number of lines of statistics.
- C** Causes the counters that keep track of the `iocnt`, `Kb_read`, and `Kb_wrtn` to be cleared for the specified logical volume or volume group.
- d** Specifies that statistics collection should be disabled for the logical volume or volume group specified.
- e** Specifies that statistics collection should be enabled for the logical volume or volume group specified.
- F** Causes the statistics to be printed in colon-separated format.
- l** Specifies the name of the stanza to list.
- s** Suppresses the header from the subsequent reports when Interval is used.
- v** Specifies that the Name specified is the name of the volume group.

Parameters

- Name** Specifies the logical volume or volume group name to monitor.
- Interval** The Interval parameter specifies the amount of time, in seconds, between each report. If Interval is used to run **lvmstat** more than once, no reports are printed if the statistics did not change since the last run. A single period (.) is printed instead.
- Count** If the Count parameter is specified, only the top Count lines of the report are generated. If the Iterations parameter is specified, **lvmstat** generates reports continuously.

28.1.1 Information about measurement and sampling

The **lvmstat** command generates reports that can be used to change logical volume configuration to better balance the input and output load between physical disks.

By default, the statistics collection is not enabled. Using the `-e` flag enables the Logical Volume Device Driver (LVMD) to collect the physical partition statistics for each specified logical volume or the logical volumes in the specified volume

group. Enabling the statistics collection for a volume group enables it for all logical volumes in that volume group. On every I/O call done to the physical partition that belongs to an enabled logical volume, the I/O count for that partition is incremented by LVMDD. All data collection is done by the LVMDD, and the **lvmstat** command reports on those statistics.

The first report section generated by **lvmstat** provides statistics concerning the time since the statistical collection was enabled. Each subsequent report section covers the time since the previous report. All statistics are reported each time **lvmstat** runs. The report consists of a header row, followed by a line of statistics for each logical partition or logical volume depending on the flags specified.

28.1.2 Examples for lvmstat

If the statistics collection has not been enabled for the volume group or logical volume you wish to monitor, the output from **lvmstat** will look like Example 28-1.

Example 28-1 Using lvmstat without enabling statistics collection

```
# lvmstat -v rootvg
0516-1309 lvmstat: Statistics collection is not enabled for this logical
device.
        Use -e option to enable.
```

To enable statistics collection for all logical volumes in a volume group (in this case the rootvg volume group), use the **-e** option together with the **-v <volume group>** flag as follows:

```
lvmstat -v rootvg -e
```

When you do not need to continue collecting statistics with **lvmstat**, it should be disabled because it has an impact on system performance. To *disable* statistics collection for all logical volumes in a volume group (in this case the rootvg volume group), use the **-d** option together with the **-v <volume group>** flag as follows:

```
lvmstat -v rootvg -d
```

If there is no activity on the partitions of the monitored device, **lvmstat** will print a period (.) for the time interval where no activity occurred. In Example 28-2 there was no activity at all in the vg0 volume group:

Example 28-2 No activity

```
# date;lvmstat -v vg0 1 10;print;date
Mon May 28 18:40:35 CDT 2001
.....
Mon May 28 18:40:45 CDT 2001
```

Monitoring logical volume utilization

Because the `lvmstat` command enables you to monitor the I/O on logical partitions, it is a powerful tool to use when monitoring logical volume utilization. In the following scenario we start by using `lvmstat` to list the volume group statistics by using the `-v <volume group>` flag as is shown in Example 28-3.

Example 28-3 Using lvmstat with a volume group

```
# lvmstat -v datavg
```

Logical Volume	iocnt	Kb_read	Kb_wrtn	Kbps
lv05	7449	4	118840	0.34
fs1v00	7366	16	626004	1.78
datavg	31	24	100	0.00
lv01	26	100	4	0.00
lv02	11	28	16	0.00
lv03	7	28	0	0.00
lv00	7	28	0	0.00

This output shows that the most-utilized logical volumes since we turned on the statistical collection are `lv05` and `fs1v00`. Example 28-4 shows the use of the `<logical volume>` flag to look at the logical partition statistics for logical volume `lv05` and `fs1v00`.

Example 28-4 Using lvmstat with a single logical volume

```
# lvmstat -l lv05
```

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
2	1	2048	0	32768	0.09
3	1	1920	0	30720	0.09
1	1	1873	4	29624	0.08
4	1	1608	0	25728	0.07
5	1	0	0	0	0.00
6	1	0	0	0	0.00
7	1	0	0	0	0.00
8	1	0	0	0	0.00
9	1	0	0	0	0.00
10	1	0	0	0	0.00

...(lines omitted)...

```
# lvmstat -l fs1v00
```

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
13	1	560	0	32768	0.09
14	1	554	0	32768	0.09
12	1	550	0	32768	0.09
11	1	544	0	32768	0.09
10	1	542	0	32640	0.09
5	1	532	0	32768	0.09

4	1	443	0	32768	0.09
2	1	442	0	32640	0.09
1	1	422	16	36660	0.10

...(lines omitted)...

From the output we see that the most-utilized logical partition for the 1v05 logical volume is logical partition number 2, and logical partition number 13 for fs1v00.

To continue our scenario, in Example 28-5 we use the `migrate1p` command to move the hot logical partitions of 1v05 and fs1v00 logical partition closer together because the volume group only has one disk. (For more information about using the `migrate1p` command, refer to *AIX 5L Version 5.2 Commands Reference*.)

Example 28-5 Using lsvg to determine the number of disks in a volume group

```
# lsvg -p datavg
datavg:
PV_NAME          PV STATE          TOTAL PPs   FREE PPs   FREE DISTRIBUTION
hdisk1           active            542         261        90..00..00..62..109
```

In Example 28-6 you can see the placement of the logical partitions for 1v05 and fs1v00, which shows output from the `lslv` command.

Example 28-6 Using lslv to view the logical partition placement

```
# lslv -m 1v05
1v05:/bigfs
LP   PP1  PV1          PP2  PV2          PP3  PV3
0001 0212 hdisk1
0002 0213 hdisk1
0003 0214 hdisk1
0004 0215 hdisk1
0005 0216 hdisk1
0006 0217 hdisk1
0007 0218 hdisk1
0008 0219 hdisk1
0009 0220 hdisk1
...(lines omitted)...

# lslv -m fs1v00
fs1v00:/jfs2
LP   PP1  PV1          PP2  PV2          PP3  PV3
0001 0091 hdisk1
0002 0092 hdisk1
0003 0093 hdisk1
0004 0094 hdisk1
0005 0095 hdisk1
0006 0096 hdisk1
0007 0097 hdisk1
```

```
0008 0098 hdisk1
0009 0099 hdisk1
0010 0100 hdisk1
0011 0101 hdisk1
0012 0102 hdisk1
0013 0103 hdisk1
0014 0104 hdisk1
...(lines omitted)...
```

This output also shows us which disk the partitions are allocated on. To illustrate use of the **migrate1p** command, we will move lv05 from physical partition 213 to a free physical partition. First we must determine which partitions on the disk are not in use. To do this we use the **lspv** command as in Example 28-7.

Example 28-7 Using lspv to determine whether a physical partition is free

```
# lspv -M hdisk1 | grep -
hdisk1:1-90
hdisk1:372-542
```

The output in this example shows us that physical partitions 1-90 and 372-542 are unused. So now we move lv05 logical partition 2 from physical partition 213 to physical partition 373, as shown in Example 28-8.

Example 28-8 Using migrate1p

```
# migrate1p lv05/2 hdisk1/373
0516-1291 migrate1p: Mirror copy 1 of logical partition 2 of logical volume
lv05 migrated to physical partition 373 of hdisk1.
```

First **migrate1p** created a mirror copy of the logical partition, and then deleted the original logical partition.

We can now easily verify that our logical partition has been moved to the desired physical partitions, as shown in Example 28-9.

Example 28-9 Using lspv to verify physical/logical partition allocation

```
# lspv -M hdisk1 | grep 373
hdisk1:373      lv05:2
```

Monitoring all logical volumes in a volume group

To monitor all logical volumes in a volume group with **lvmstat**, use the **-v <volume group>** flag as shown in Example 28-10 on page 525.

Example 28-10 Using lvmstat on a volume group level

```
# lvmstat -v rootvg
```

Logical Volume	iocnt	Kb_read	Kb_wrtn	Kbps
lv05	682478	16	8579672	16.08
loglv00	0	0	0	0.00
datav	0	0	0	0.00
lv07	0	0	0	0.00
lv06	0	0	0	0.00
lv04	0	0	0	0.00
lv03	0	0	0	0.00

The **lvmstat** command report above is per logical volume statistics in the volume group. The report has the following format:

Logical Volume	The device name of the logical volume
iocnt	Number of read and write requests
Kb_read	The total number of kilobytes read
Kb_wrtn	The total number of kilobytes written
Kbps	The amount of data transferred in kilobytes per second

The output in Example 28-10 shows that lv05 is the most used of all of the logical volumes in this volume group. To map the logical volume name to a file system (if the logical volume has a stanza in /etc/filesystems), we use the **lsfs** command as in Example 28-11.

Example 28-11 Using lsfs to determine file system name for a logical volume

```
# lsfs -q /dev/lv05
Name      Nodename  Mount Pt      VFS  Size  Options  Auto Accounting
/dev/lv05  --        /work/fs2    jfs2 2359296 rw      yes  no
(lv size: 2359296, fs size: 2359296, block size: 4096, sparse files: yes, inline log: yes, inline log size: 10240)
```

By using the **-q** flag with the **lsfs** command we get statistics that include the logical volume information such as the file system name, logical volume name, file system type, and fragmentation sizes. The file system for this logical volume is /work/fs2, its size is 1.1 GB (2359296 / 2 / 1024 / 1024) with a 4 KB block size, and it is a J2 file system with an inline log.

To monitor only the logical volumes in the volume group that has the highest number of read and write requests (**iocnt**), use the **-c #** flag to the **lvmstat** command, where **#** is the number of lines to display. In Example 28-12 on page 526 we want to see the three highest-use logical volumes (**lvmstat** places the logical volume with the highest **iocnt** at the top), and the number of measurements will be five with a three-second interval (**-sc 3 3 5**).

Example 28-12 Using lvmstat on a volume group level with the highest iocnt

```
# lvmstat -v vg0 -sc 3 3 5
```

Logical Volume	iocnt	Kb_read	Kb_wrtn	Kbps
lv05	724778	32	9115128	17.06
lv05	181	0	2012	631.71
lv05	223	0	892	279.84
lv05	379	0	1516	476.36

As can be seen in the output above, the first part is the summary for the volume group because statistics collection was enabled. The following lines show the logical volumes with the highest number of read and write requests (**iocnt**). We can see that **lv05** is the logical volume that has the most I/O during our measurement.

Monitoring a single logical volume

To monitor a single logical volume with **lvmstat** you only need to use the **-l <logical volume>** flag as in Example 28-13.

Example 28-13 Using lvmstat on a single logical volume

```
# lvmstat -l lv05
```

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
72	1	37736	0	263036	0.50
66	1	7960	0	199956	0.38
71	1	7330	0	170024	0.32
67	1	2835	0	64732	0.12
65	1	1735	0	37704	0.07
63	1	242	0	968	0.00
64	1	179	0	716	0.00
68	1	33	0	132	0.00
62	1	27	0	108	0.00
1	1	0	0	0	0.00
...(lines omitted)...					
70	1	0	0	0	0.00

lvmstat reports on each individual logical partition with a one-line output for each as can be seen in the output above. The report has the following format:

Log_part	Logical partition number
mirror#	Mirror copy number of the logical partition
iocnt	Number of read and write requests
Kb_read	The total number of kilobytes read

Kb_wrtn The total number of kilobytes written
 Kbps The amount of data transferred in kilobytes per second

We now see that there is a group of partitions that are used the most, so we limit our scope with the `-c` flag with the number of rows to show. `lvmstat` orders the list top down based on the number of `iocnt`. In Example 28-14, which iterates once every 60 seconds, we save the output in a file as well as display it onscreen with the `tee` command.

Example 28-14 lvmstat run on a single logical volume with top 10 logical partitions

```
# lvmstat -l lv05 -c 10 60|tee /tmp/lvmstat.out
```

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
72	1	67221	0	467148	0.89
66	1	14066	0	353832	0.67
71	1	12991	0	300912	0.57
67	1	4951	0	113056	0.21
65	1	3079	0	66788	0.13
63	1	485	0	1940	0.00
64	1	340	0	1360	0.00
68	1	59	0	236	0.00
62	1	48	0	192	0.00

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
72	1	3704	0	23432	369.23
66	1	616	0	15408	242.79
71	1	575	0	13128	206.86
67	1	299	0	6612	104.19
65	1	142	0	2932	46.20
63	1	37	0	148	2.33
64	1	30	0	120	1.89
62	1	4	0	16	0.25
68	1	4	0	16	0.25

Log_part	mirror#	iocnt	Kb_read	Kb_wrtn	Kbps
72	1	3258	0	21660	340.99
71	1	736	0	17868	281.30
66	1	612	0	15384	242.19
67	1	222	0	5012	78.90
65	1	132	0	2892	45.53
64	1	13	0	52	0.82
63	1	4	0	16	0.25
62	1	2	0	8	0.13
68	1	2	0	8	0.13

...(lines omitted)...

By looking at the utilization, we get a feel for how the logical volume is used. In the output, access to logical partition 72 stands out, but logical partition 71 and 66

are very close when it comes to the amount of data that is written. To find out the physical partition where each of these hot logical partitions is located on disk, we use the `lslv` command.

Summarizing I/O utilization per physical partition

To summarize the physical partition utilization, we create and use a simple script that we call `lvmstat.sum`, shown in Example 28-16. This script uses the saved output file from our previous `lvmstat` command and summarizes the partition utilization as shown in Example 28-15.

Example 28-15 Using a script to summarize most-used partitions

```
# lvmstat.sum /tmp/lvmstat.out
Log_part mirror# iocnt  Kb_read  Kb_wrtn
  72      1 158860      0 1097940
  66      1  32470      0  815236
  71      1  30511      0  706512
  67      1  11696      0  266008
  65      1   7211      0  154420
  63      1   1249      0   4996
  64      1    897      0   3588
  68      1    131      0    524
  62      1    121      0    484
```

Note that we include the `mirror` number in the output because if the logical volume is mirrored, we could find the right physical partition for the logical partition. The output above shows us that the logical partitions that are most used are consecutive from the logical volumes perspective, and all of them mirror copy 1. However, it is interesting to note that the `iocnt` value for logical partition 72 is almost five times higher than the `iocnt` value for logical partition 66 but has only 25% more written data. The `lvmstat.sum` script is shown in Example 28-16.

Example 28-16 lvmstat.sum script

```
1 cat $1|
2 (
3     printf "%-8s %8s %s %9s %9s\n" "Log_part" "mirror#" "iocnt" "Kb_read"
   "Kb_wrtn"
4     awk '
5     $1~/[0-9]/&&i>=2{
6         iocnt[$1,$2]=iocnt[$1,$2]+$3
7         read[$1,$2]=read[$1,$2]+$4
8         write[$1,$2]=write[$1,$2]+$5
9     }
10    /Log/{i++}
11    END{
12        for (f in iocnt)
13            printf " %8s %8s%8s%10s%10s\n",
```

```
14         substr(f,0,length(f)-1), substr(f,length(f)-1), iocnt[f],  
read[f], write[f]  
15     }' i=0 | sort -k3nr  
16 )
```

The `lvmstat.sum` script works by extracting the logical partition number, mirror number, I/O count, KB read, and KB write values from the saved `lvmstat` output. It will discard the first report section because it is the accumulation since the statistical collection was enabled. (If it is set to a value higher than zero it will include this report in the summary as well.) The `awk` command uses a table for summarizing the I/O counts, KB read, and KB write for each logical partition using the logical partition and the mirror number as indices. At the end (END statement) it loops through the tables using the indices in the for loop and prints the logical partition part of each index first, then the mirror number part of the same index, and then the summarized I/O count, KB read, and KB write. When `awk` has produced the output lines, we use the `sort` command to sort the output using the summarized I/O count (third field) numerically and in reverse (descending) order.

Archived

Network-related performance tools

This part describes the tools that monitor the performance-relevant data and statistics for networks. This includes tools to:

- ▶ monitor the network adapters
- ▶ monitor the different layers of TCP/IP networks
- ▶ monitor the system resources used by the networking software
- ▶ trace data sent and received on the networks
- ▶ monitor Network File System (NFS) usage on client and server systems
- ▶ set and change network performance relevant system parameters

Knowledge of the basics of network communication and the network protocols used is required to understand the data gathered by the tools discussed in this chapter. The *AIX 5L Version 5.2 System User's Guide: Communications and Networks* provides the necessary information.

This part contains detailed information about these network monitoring and tuning tools:

- ▶ Network adapter statistics monitoring tools, described in Chapter 29, “atmstat, entstat, estat, fddistat, and tokstat commands” on page 539:
 - The **atmstat** command is used to monitor Asynchronous Transfer Mode (ATM) adapter statistics.
 - The **entstat** command is used to monitor Ethernet adapter statistics.
 - The **estat** command is used to monitor RS/6000 SP Switch adapter statistics.
 - The **fddistat** command is used to monitor the Fiber Distributed Data Interface (FDDI) network adapter statistics.
 - The **tokstat** command is used to monitor token-ring network adapter statistics.
- ▶ The **netstat** command described in Chapter 31, “The netstat command” on page 619 provides data and statistics for the different network layers, system resources used by networks, and network configuration information such as:
 - Statistics for the different network protocols used
 - Statistics for the communications memory buffer (mbuf) usage
 - Information about the configured network interfaces
 - Routing information
- ▶ The **no** command discussed in Chapter 34, “The no command” on page 665 is used to display, set, and change the network parameters.
- ▶ Chapter 33, “The nfsstat command” on page 655 discusses the use of the **nfsstat** command to monitor Remote Procedure Call (RPC) and NFS statistics on NFS server and client systems.
- ▶ The **nfso** command described in Chapter 32, “The nfso command” on page 645 is used to display, set, and change NFS variables and to remove file locks from NFS client systems on an NFS server.
- ▶ To trace data sent to and received from the network, the following commands can be used:
 - The **iptrace** command discussed in “iptrace” on page 569 is used to gather the data sent to and received from the network.

- The **ipfilter** command described in “ipfilter” on page 573 can be used to sort or extract a part of the data previously gathered by the **iptrace** command.
- The **tcpdump** command discussed in “tcpdump” on page 587 is used to gather and display packets sent to and received from the network.
- The **ipreport** command described in “ipreport” on page 572 is used to format the data gathered by the **iptrace** or **tcpdump** commands.
- The **trpt** command discussed in “trpt” on page 612 can be used to trace Transmission Control Protocol (TCP) sockets.

For more detailed information about the TCP/IP protocols, refer to:

- ▶ 1.5, “Network performance” on page 31<<QUESTION-Xref>>
- ▶ *AIX 5L Version 5.2 Performance Management Guide*
- ▶ *AIX 5L Version 5.2 System Management Guide: Communications and Networks*
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376
- ▶ *RS/6000 SP System Performance Tuning Update*, SG24-5340
- ▶ <http://www.rs6000.ibm.com/support/sp/perf>
- ▶ Appropriate Request For Comment (RFC) at <http://www.rfc-editor.org/>

There are also excellent books available on the subject but a good starting point is RFC 1180: A TCP/IP Tutorial.

TCP/IP protocol and services tables

Table 1 is an extraction from the */etc/protocols* file that shows some interesting protocol types and their numeric value.

Table 1 `grep -v ^#/etc/protocols`

Symbolic name	Numeric ID	Protocol	Description
ip	0	IP	Dummy for the Internet Protocol
icmp	1	ICMP	Internet control message protocol
igmp	2	IGMP	Internet group multicast protocol
tcp	6	TCP	Transmission control protocol
udp	17	UDP	User datagram protocol

Table 2 is an extraction from the `/etc/services` file that shows some interesting services, ports, and the protocol used on that port.

Table 2 Selection from /etc/services

Symbolic name	Port	Protocol	Description
echo	7	tcp	Used by the <code>ping</code> command
echo	7	udp	Used by the <code>ping</code> command
ftp-data	20	tcp	Used by the <code>ftp</code> command
ftp	21	tcp	Used by the <code>ftp</code> command
telnet	23	tcp	Used by the <code>telnet</code> command
smtp	25	tcp	Used by the mail commands
domain	53	udp	Used by nameserver commands
pop	109	tcp	Used by postoffice mail commands
pop3	110	tcp	Used by postoffice3 mail commands
exec	512	tcp	Used by remote commands
login	513	tcp	Used by remote commands
shell	514	tcp	Used by remote commands
printer	515	tcp	Used by print spooler commands
route	520	udp	Used by router (outed) commands

ICMP message type table

Table 3 lists some Internet Control Message Protocol (ICMP) message types. The table includes some of the more interesting message types. For a detailed description of the message type and its specific ICMP packet format refer to the appropriate Request For Comment (RFC).

Table 3 Some ICMP message types

Symbolic	Numeric ID	RFC
Echo Reply	0	RFC792
Destination Unreachable	3	RFC792
Source Quench	4	RFC792
Redirect	5	RFC792

Symbolic	Numeric ID	RFC
Echo	8	RFC792
Router Advertisement	9	RFC1256
Router Solicitation	10	RFC1256
Time Exceeded	11	RFC792
Parameter Problem	12	RFC792
Time stamp	13	RFC792
Time Stamp Reply	14	RFC792
Information Request	15	RFC792
Information Reply	16	RFC792
Traceroute	30	RFC1393

Packet header formats

The following are schematic layouts for the token-ring, Ethernet (V2 and 802.3), IP, TCP, and UDP header formats. For a more thorough explanation of the TCP/IP protocol headers, refer to the appropriate RFC and the "TCP/IP Protocols chapter" in the *AIX 5L Version 5.2 System Management Guide: Communications and Networks*.

Token-ring frame header

In Table 4, the scale is in bytes (B).

Table 4 Token-ring frame header

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1...			
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+
	S		A		F		D	S	T			S	R	C			R	I		(0	-	3)										
	D		C		C																													
+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+	-	+
SD																	Starting delimiter																	
AC																	Access control																	
FC																	Frame control																	
DST																	Destination host address																	
SRC																	Source host address																	
RI																	Routing information. Can have variable length.																	

Ethernet V2 frame header

In Table 5 on page 536, the scale is in bytes (B).

Table 5 Ethernet V2 frame header

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
+-+																					
PA					DST					SRC					L						
															N						
+-+																					

PA Preamble
 DST Destination host address
 SRC Source host address
 LN Length of client protocol data

Ethernet 802.3 frame header

In Table 6, the scale is in bytes (B).

Table 6 Ethernet 802.3 frame header

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
+-+																					
PA					S DST					SRC					T						
					F										Y						
+-+																					

PA Preamble
 SF Start frame delimiter
 DST Destination host address
 SRC Source host address
 TY Type of client protocol

IP V4 (RFC 791) packet header

Table 7 illustrates the IP V4 header according to RFC 791. (Refer to this RFC at <http://www.rfc-editor.org/> for a detailed explanation.) The struct ip can be found in /usr/include/netinet/ip.h. The first line shows the byte index; the second line shows the bit index. The last byte for each row is on the right side of the header layout.

Table 7 IP V4 (RFC 791) packet header

0							1							2							3										
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
+-+																															
Version				IHL				Type of Service				Total Length																			
+-+																															
Identification										Flags				Fragment Offset																	
+-+																															
Time to Live				Protocol				Header Checksum																							
+-+																															
Source Address																															

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Destination Address																
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Options												Padding				

TCP (RFC 793) packet header

Table 8 illustrates the TCP header according to RFC 793. (Refer to this RFC at <http://www.rfc-editor.org/> for a detailed explanation.) The struct `tcphdr` can be found in `/usr/include/netinet/tcp.h`. The first line shows the byte index, and the second line shows the bit index. The last byte for each row is on the right side of the header layout.

Table 8 TCP (RFC 793) packet header

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16											
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7				
Source Port								Destination Port																			
Sequence Number																											
Acknowledgment Number																											
Data Offset				Reserved				URG				ACK				RST				SYN				FIN			
Checksum												Window															
Options												Urgent Pointer															
data																											

UDP (RFC 768) packet header

Table 9 illustrates the UDP header according to RFC 768. (Refer to this RFC at <http://www.rfc-editor.org/> for a detailed explanation.) The struct `udphdr` can be found in `/usr/include/netinet/udp.h`. The first line shows the byte index, and the second line shows the bit index. The last byte for each row is on the right side of the header layout.

Table 9 UDP (RFC 768) packet header

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

Source Port	Destination Port
Length	Checksum

ICMP (RFC 792) packet header

Table 10 illustrates the basic¹ ICMP header according to RFC 792. (Refer to this RFC at <http://www.rfc-editor.org/> for a detailed explanation.) The struct `icmp6_hdr` can be found in `/usr/include/netinet/icmp6.h`. The first line shows the byte index, and the second line shows the bit index. The last byte for each row is on the right side of the header layout. Refer to Table 3 on page 534 for more information about the type field.

Table 10 ICMP (RFC 792) packet header

0	1	2	3
0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
Type	Code	Checksum	
unused			
Internet Header + 64 bits of Original Data Datagram			

¹ Various ICMP messages use different packet types.

atmstat, entstat, estat, fddistat, and tokstat commands

The **atmstat**, **entstat**, **estat**, **fddistat**, and **tokstat** commands are performance monitoring tools that display device-driver statistics for the associated (network) device, which are:

- ▶ Asynchronous Transfer Mode (ATM) device driver
- ▶ Ethernet device driver
- ▶ RS/6000 SP switch device driver
- ▶ Fiber Distributed Data Interface (FDDI) device driver
- ▶ Token-ring device driver

The **atmstat**, **entstat**, **fddistat**, and **tokstat** commands reside in `/usr/sbin`, which is linked to `/usr/bin`. These commands are part of the `devices.common.IBM.atm.rte`, `devices.common.IBM.ethernet.rte`, `devices.common.IBM.fddi.rte`, and `devices.common.IBM.tokenring.rte` filesets, which are installable from the AIX base operation system installation media.

The **estat** command resides in `/usr/lpp/ssp/css/css` and is part of the `ssp.css` fileset, which is installable from the IBM Parallel System Support Programs (PSSP) installation media.

29.1 atmstat

The syntax of the **atmstat** command is:

```
atmstat [-drt] <device name>
```

Flags

- d** Displays detailed statistics.
- r** Resets all statistics to their initial values. This flag can be issued only by privileged users.
- t** Toggles debug trace in some device drivers.

Parameters

Device_Name The name of the ATM device (for example, atm0). If an invalid device name is specified, the **atmstat** command produces an error message stating that it could not connect to the device.

29.1.1 Information about measurement and sampling

The **atmstat** command used without flags provides generic statistics that consist of transmit statistics, receive statistics, and general statistics. This includes packets and bytes transmitted and received, information about hardware and software queues usage, and error counters. If the **-d** flag is used, device-specific statistics are displayed along with the device-driver statistics.

The **atmstat** command provides a snapshot of the device-driver statistics collected by the Network Device Driver (NDD). The header file `/usr/include/sys/ndd.h` defines the used data structure `ndd_genstats` as well as the `ioctl()` operation `NDD_GET_ALL_STATS`, which is used to read the data from the NDD. **atmstat** uses a device-dependent routine defined in the ODM to display the device-specific statistics. This device-dependent routine is a command that is executed using `fork()` and `exec()` out of **atmstat**. In a busy system there may be some delay doing this. In case the system is running out of resources (for example, low on memory), the necessary `fork()` may fail. All device-dependent routines can be found using the command `odmget -q attribute=addl_stat PdAt`. All statistic values displayed by **atmstat** are the absolute values since startup or the last reset of these values, which is done by using `atmstat -r Device_Name`.

The device-driver statistics are read out of the NDD at execution time of **atmstat**. The device-specific statistics are read from the device driver using the `ioctl()` system call. The data gets displayed and **atmstat** exits. Using the `-r` flag, **atmstat** first displays the current statistic values and then resets them.

The device-specific data for Microchannel (MCA) ATM and Peripheral Component Interconnect (PCI) ATM adapters are different.

The output of the **atmstat** command consists of five sections: the title fields, the transmit statistics fields, the receive statistics fields, the general statistics fields, and the adapter specific statistic fields. Refer to the *AIX 5L Version 5.2 Commands Reference* for a description of all output fields.

29.1.2 Examples for atmstat

The output of **atmstat** always shows the device-driver statistics. On request, using the `-d` flag, more detailed data is displayed.

Example 29-1 shows the output of **atmstat** on an MCA system.

Example 29-1 Displaying ATM device-driver statistics on an MCA system

```
# atmstat -d atm0
-----
ATM STATISTICS (atm0) :
Device Type: Turboways 155 MCA ATM Adapter
Hardware Address: 40:00:30:31:00:31
Elapsed Time: 11 days 1 hours 36 minutes 43 seconds

Transmit Statistics:                               Receive Statistics:
-----
Packets: 3969322                                   Packets: 3852487
Bytes: 3011576880                                  Bytes: 731915050
Interrupts: 0                                       Interrupts: 3893792
Transmit Errors: 0                                  Receive Errors: 0
Packets Dropped: 0                                 Packets Dropped: 0
Bad Packets: 0

Max Packets on S/W Transmit Queue: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Cells Transmitted: 6422555                          Cells Received: 17232251
Out of Xmit Buffers: 0                               Out of Rcv Buffers: 0
Current HW Transmit Queue Length: 0                 CRC Errors: 0
Current SW Transmit Queue Length: 0                 Packets Too Long: 0
                                                    Incomplete Packets: 0
                                                    Cells Dropped: 0
```

```
General Statistics:
-----
No mbuf Errors: 16
Adapter Loss of Signals: 0
Adapter Reset Count: 0
Driver Flags: Up Running Simplex
                64BitSupport
Virtual Connections in use: 12
Max Virtual Connections in use: 14
Virtual Connections Overflow: 0
SVC UNI Version: auto_detect
```

```
Turboways ATM Adapter Specific Statistics:
-----
Packets Dropped - No small DMA buffer: 0
Packets Dropped - No medium DMA buffer: 0
Packets Dropped - No large DMA buffer: 0
Receive Aborted - No Adapter Receive Buffer: 0
Transmit Attempted - No small DMA buffer: 0
Transmit Attempted - No medium DMA buffer: 0
Transmit Attempted - No large DMA buffer: 0
Transmit Attempted - No MTB DMA buffer: 0
Transmit Attempted - No Adapter Transmit Buffer: 0
Max Hardware transmit queue length: 45
Small Mbuf in Use: 0
Medium Mbuf in Use: 0
Large Mbuf in Use: 66
Huge Mbuf in Use: 0
MTB Mbuf in Use: 0
Max Small Mbuf in Use: 0
Max Medium Mbuf in Use: 44
Max Large Mbuf in Use: 302
Max Huge Mbuf in Use: 0
MTB Mbuf in Use: 0
Small Mbuf overflow: 0
Medium Mbuf overflow: 0
Large Mbuf overflow: 16
Huge Mbuf overflow: 0
MTB Mbuf overflow: 0
```

Example 29-2 shows **atmstat** on a PCI system.

Example 29-2 Displaying ATM device-driver statistics on a PCI system

```
# atmstat -d atm0
-----
ATM STATISTICS (atm0) :
Device Type: IBM PCI 155 Mbps ATM Adapter (14104f00)
Hardware Address: 00:04:ac:ad:29:16
```

Elapsed Time: 6 days 0 hours 45 minutes 0 seconds

Transmit Statistics:

Packets: 171920
Bytes: 7953953
Interrupts: 0
Transmit Errors: 0
Packets Dropped: 0

Max Packets on S/W Transmit Queue: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Cells Transmitted: 276313
Out of Xmit Buffers: 0
Current HW Transmit Queue Length: 0
Current SW Transmit Queue Length: 0

Receive Statistics:

Packets: 171919
Bytes: 7145739
Interrupts: 172154
Receive Errors: 0
Packets Dropped: 0
Bad Packets: 0

Cells Received: 276306
Out of Rcv Buffers: 0
CRC Errors: 0
Packets Too Long: 0
Incomplete Packets: 0
Cells Dropped: 13

General Statistics:

No mbuf Errors: 0
Adapter Loss of Signals: 0
Adapter Reset Count: 0
Driver Flags: Up Running Simplex
64BitSupport PrivateSegment
Virtual Connections in use: 15
Max Virtual Connections in use: 18
Virtual Connections Overflow: 0
SVC UNI Version: uni3.1

IBM PCI 155 Mbps ATM Adapter Specific Statistics:

Total 4K byte Receive Buffers: 96 Using: 64
Maximum 4K byte Receive Buffers used 96
Maximum Configurable 4K byte Receive Buffers 800

The major fields of interest concerning performance and performance monitoring are:

Elapsed Time The real-time period that has elapsed since the last time the statistics were reset.

Transmit and Receive Packets The number of packets successfully transmitted and received by the device.

Transmit and Receive Bytes	The number of bytes successfully transmitted and received by the device. These values and their related packet counts can show how the system is using this network adapter. For example, transmit and receive values may be close to equal or they may differ by a huge margin.
Transmit and Receive Interrupts	The number of transmit and receive interrupts received by the driver from the adapter. If these counters increase fast, then the number of interrupts to be handled by the operating system may reach a level where overall system performance may be affected. Other monitoring tools vmstat can be used to control the interrupts per second handled by the system.
Transmit and Receive Cells	The number of cells transmitted by this device.
Out of Xmit Buffers	The number of packets dropped because transmit buffers are full. Tuning the adapter's <code>sw_txq_size</code> value is required. The lsattr -E1 atm0 command shows the current value set for the adapters transmit queue size. lsattr -R1 atm0 -a sw_txq_size displays the possible values for <code>sw_txq_size</code> . Use the chdev -l atm0 -a sw_txq_size=xxx command to change this value.
Out of Rcv Buffers	The number of packets dropped because of out of receive buffers condition. If this counter is not zero, then the <code>rx_req_size</code> parameter of the adapter may be changed. To get the current <code>rx_que_size</code> value, use the lsattr -E1 atm0 command. If this adapter parameter is zero, which is the default, then the calculation for receive buffers is based on available communications memory buffer (mbufs). mbuf tuning using the no command is required in this case. Refer to Chapter 34, "The no command" on page 665 for more details. If <code>rx_que_size</code> is not zero, then increasing it using the chdev -l atm0 -a rx_que_size=nnn command could be necessary. However, keep in mind that each receive buffer requires memory and a further mbuf tuning may be necessary.
Current HW Transmit Queue Length	The current number of transmit packets on the hardware queue.

No mbuf Errors	The number of times mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain mbuf buffers to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. This may cause retransmission by the sending system, which increases load on the system as well as the additional network load. The netstat command can be used to confirm this. For details refer to Chapter 31, “The netstat command” on page 619.
Driver Flag	The neighborhood discovery daemon flags. It should not be in <i>Limbo</i> state, which is an indication of a missing signal on the adapter. The cables should be checked in this case.
Virtual Connections in use	The number of virtual connections that are currently allocated or in use.
Max Virtual Connections in use	The maximum number of virtual connections allocated since the last reset of the statistics.
Virtual Connections Overflow	The number of virtual connection requests that have been denied. If this is not zero, then an adjustment of the adapter parameter <code>max_vc</code> may be necessary. Use lsattr -E1 Device_Name (for example, lsattr -E1 atm0) to get the current <code>max_vc</code> value. The lsattr -R1 atm0 -a max_vc command can be used to see which values are permitted. To change <code>max_vc</code> use the chdev -l atm0 -a max_vc=xxxx command.

The Turboways ATM Adapter Specific Statistics in Example 29-1 on page 541 shows statistics for adapter buffer usage. This adapter uses mbufs in five fixed sizes:

- ▶ Small mbufs are 256 bytes.
- ▶ Medium mbufs are 4096 bytes.
- ▶ Large mbufs are 8192 bytes.
- ▶ Huge mbufs are 16384 bytes.
- ▶ MTB mbufs are of variable size in the range of 32 KB to 1024 KB

In case any of the Mbuf overflow statistics are not zero, the corresponding adapter parameter should be tuned. An overflow is not catastrophic. The device driver will attempt to get the next smaller size of buffer. However, this is inefficient and costs performance. The minimum and the maximum mbuf number allocated by the adapter can be set using System Management Interface Tool (SMIT) by running **smitty chg_atm**. For more information, see *RS/6000 and Asynchronous Transfer Mode*, SG24-4796.

The IBM PCI 155 Mbps ATM Adapter Specific Statistics part of Example 29-2 on page 542 shows the device-specific statistics for this adapter. These statistics show the values for the 4 KB byte pre-mapped receive buffers, which are used for Direct Memory Access (DMA) data transfers of mbufs from the adapter to the system protocol stacks. The minimum number of buffers allocated by the adapter is stored in ODM as the `rv_buf4k_min` attribute of the adapter. Use `lsattr -El atm0` to get the current value for this attribute. Setting the `rv_buf4k_min` attribute to a higher value decreases the chance of running out of buffers when an application has high bursts of small packets. The statistic field `Maximum 4K byte Receive Buffers used` shows the high water mark for pre-mapped receive buffers the system reached. Changing the `rv_buf4k_min` attribute value should be done with care. SMIT or the `chdev` command can be used to change the value.

Note: Changing ATM adapter parameters using `smit chg_atm` or the `chdev` command is only possible if the adapter is not in use. Using the `-P` flag on the `chdev` command stores the changes only in the ODM database. This is useful for devices that cannot be made unavailable and cannot be changed while in the available state. The changes can be applied to the device by restarting the system.

Monitoring a ATM adapter on a regular basis using `atmstat` can find possible problems before the users notice any slowdown. The problem can be taken care of by redesigning the network layout or tuning either the adapter parameters using the `chdev` command or the network options using the `no` command. (See Chapter 34, “The `no` command” on page 665.)

29.2 entstat

The syntax of the `entstat` command is:

```
entstat [ -drt ] Device_Name
```

Flags

- d** Displays all of the statistics, including the device-specific statistics. Some adapters may not have any device-specific statistics.
- r** Resets all statistics to their initial values. This flag can only be issued by privileged users.
- t** Toggles debug trace in some device drivers.

Parameters

Device_Name The name of the Ethernet device (for example, ent0). If an invalid device name is specified, the **entstat** command produces an error message stating that it could not connect to the device.

29.2.1 Information about measurement and sampling

The **entstat** command used without flags provides generic statistics that consist of transmit statistics, receive statistics, and general statistics. This includes packets and bytes transmitted and received, and information about hardware and software queue usage as well as error counters. Using the **-d** flag displays device-specific statistics as well as device-driver statistics.

The **entstat** command provides a snapshot of the device-driver statistics collected by the NDD. The header file `/usr/include/sys/ndd.h` defines the used data structure `ndd_genstats` as well as the `ioctl()` operation `NDD_GET_ALL_STATS`, which is used to read the data from the NDD. **entstat** uses a device-dependent routine defined in the ODM to display the device-specific statistics. This device-dependent routine is a command that is executed using `fork()` and `exec()` out of **entstat**. In a busy system there may be some delay doing this. In case the system is running out of resources (for example low on memory), the necessary `fork()` may fail. All device-dependent routines can be found using the command `odmget -q attribute=addl_stat PdAt`. All statistic values displayed by **entstat** are the absolute values since startup or the last reset of these values, which is done by using **entstat -r Device_Name**.

Hardware error recovery may cause some statistic values to be reset. If this happens, a second Elapsed Time is displayed in the middle of the statistic's output reflecting the time elapsed since the reset.

The device-driver statistics are read out of the NDD at execution time of **entstat**. The device-specific statistics are read from the device driver using the `ioctl()` system call. The data gets displayed and **entstat** exits. If the **-r** flag is used, **entstat** first displays the current statistic values and then resets them.

Some adapters may not support a specific statistic. In this case the non-supported statistic fields are always zero.

The output of the **entstat** command consists of five sections; the title fields, the transmit statistics fields, the receive statistics fields, the general statistics fields, and the adapter specific statistic fields. Refer to the *AIX 5L Version 5.2 Commands Reference* for a description of all output fields.

29.2.2 Examples for entstat

The output of **entstat** always shows the device-driver statistics. When using the **-d** flag, the additional device-specific statistics are displayed. Some adapters may not have any device-specific statistics.

Example 29-3 shows the **entstat** output including device-specific statistics.

Example 29-3 Displaying Ethernet device-driver statistics

```
# entstat -d ent0
-----
ETHERNET STATISTICS (ent0) :
Device Type: 10/100 Mbps Ethernet PCI Adapter II (1410ff01)
Hardware Address: 00:02:55:af:1a:72
Elapsed Time: 11 days 3 hours 19 minutes 51 seconds

Transmit Statistics:                               Receive Statistics:
-----
Packets: 2121360                                Packets: 2493230
Bytes: 307990132                                Bytes: 368003398
Interrupts: 0                                    Interrupts: 2493091
Transmit Errors: 0                                  Receive Errors: 1
Packets Dropped: 0                                 Packets Dropped: 0
                                                    Bad Packets: 0

Max Packets on S/W Transmit Queue: 37
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 71173                        Broadcast Packets: 87040
Multicast Packets: 2                               Multicast Packets: 2
No Carrier Sense: 0                               CRC Errors: 0
DMA Underrun: 0                                   DMA Overrun: 0
Lost CTS Errors: 0                                Alignment Errors: 0
Max Collision Errors: 0                           No Resource Errors: 0
Late Collision Errors: 0                          Receive Collision Errors: 1082
Deferred: 4554                                    Packet Too Short Errors: 1
SQE Test: 0                                       Packet Too Long Errors: 0
Timeout Errors: 0                                 Packets Discarded by Adapter: 0
Single Collision Count: 1723                    Receiver Start Count: 0
Multiple Collision Count: 515
Current HW Transmit Queue Length: 1

General Statistics:
-----
No mbuf Errors: 0
Adapter Reset Count: 1
Adapter Data Rate: 200
Driver Flags: Up Broadcast Running
                Simplex AlternateAddress 64BitSupport
```

10/100 Mbps Ethernet PCI Adapter II (1410ff01) Specific Statistics:

```

-----
Link Status: Up
Media Speed Selected: 100 Mbps Full Duplex
Media Speed Running: 100 Mbps Full Duplex
Receive Pool Buffer Size: 1024
Free Receive Pool Buffers: 1024
No Receive Pool Buffer Errors: 0
Receive Buffer Too Small Errors: 0
Entries to transmit timeout routine: 0
Transmit IPsec packets: 0
Transmit IPsec packets dropped: 0
Receive IPsec packets: 0
Receive IPsec packets dropped: 0
Inbound IPsec SA offload count: 0
Transmit Large Send packets: 0
Transmit Large Send packets dropped: 0
Packets with Transmit collisions:
  1 collisions: 0          6 collisions: 0          11 collisions: 0
  2 collisions: 0          7 collisions: 0          12 collisions: 0
  3 collisions: 0          8 collisions: 0          13 collisions: 0
  4 collisions: 0          9 collisions: 0          14 collisions: 0
  5 collisions: 0         10 collisions: 0          15 collisions: 0

```

The major fields of interest concerning performance and performance monitoring are:

Elapsed Time	The real-time period that has elapsed since the last time the statistics were reset. During error recovery, when a hardware error is detected part of the statistics may be reset. In this case another Elapsed Time is displayed in the middle of the output reflecting the time elapsed since the reset. In this example there was no such event so there is no additional Elapsed Time displayed.
Transmit and Receive Packets	The number of packets successfully transmitted and received by the device.
Transmit and Receive Bytes	The number of bytes successfully transmitted and received by the device. These values and their related packet count can show how the system is using this network adapter. For example, transmit and receive values may be close to equal, or they may differ by a huge margin.

Transmit and Receive Interrupts	The number of transmit and receive interrupts received by the driver from the adapter. If these counters increase fast, then the number of interrupts to be handled by the operating system may reach a level where overall system performance may be affected. Other monitoring tools like vmstat can be used to control the interrupts per second handled by the system.
Max Packet on S/W Transmit Queue	The maximum number of outgoing packets ever queued to the software transmit queue. If this value reaches the <code>xt_que_size</code> set for the adapter then the <code>xt_que_size</code> of the adapter is not set to an adequate value. The command lsattr -E1 Device_Name , like lsattr -E1 ent0 , shows the current adapter settings including <code>xt_que_size</code> . Use SMIT or chdev to increase <code>xt_que_size</code> if necessary and possible. The possible values allowed to set can be found using the ODM as shown in Example 29-7 on page 565 or the lsattr -R1 ent0 -a xt_que_size command.
S/W Transmit Queue Overflow	The number of outgoing packets that overflowed the software transmit queue. If this is not zero then you must increase the transmit queue size <code>xt_que_size</code> , as shown in the description for the field Max Packets on S/W Transmit Queue.
Current S/W + H/W Transmit Queue Length	The number of pending outgoing packets on either the software transmit queue or the hardware transmit queue. This reflects the current load on the adapter. This is the sum of the Current SW Transmit Queue Length and Current HW Transmit Queue Length fields.
Broadcast Packets	The number of broadcast packets transmitted and received without any error. A high value compared to the total transmitted and received packets indicates that the system is sending and receiving many broadcasts. Broadcasts increase network load and may increase the load on all the other systems on the same subnetwork.
Receive Collision Errors	The number of incoming packets with the collision errors during the reception. This number, compared with number of packets received, should stay low.
Single Collision Count	The number of outgoing packets with single (only one) collision encountered during transmission. This number, compared with the number of packets transmitted, should stay low.
Multiple Collision Count	The summary of outgoing packets with multiple (up to 15) collisions encountered during transmission.

Current HW Transmit Queue Length	The number of outgoing packets currently on the hardware transmit queue.
No mbuf Errors	The number of times communications mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain mbufs to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. This may cause retransmission by the sending system, which increases load on the system as well as additional network load. The netstat command can be used to confirm this. For details refer to Chapter 31, “The netstat command” on page 619.

An increasing number of collisions could be caused by too much load on the subnetwork. A split of this subnetwork into two or more subnetworks may be necessary.

If the statistics for errors, such as the transmit errors, are increasing fast, these errors should be corrected first. Some errors may be caused by hardware problems. These hardware problems need to be fixed before any software tuning is performed. The error counter should stay close to zero.

Sometimes it is useful to know how many packets an application or task sends or receives. Use **entstat -r Device_Name** to reset the counters to zero, then run the application or task. After the completion of the application or task, run **entstat Device_Name** again to get this information. An example for using **entstat** to monitor Ethernet statistics during execution of one program is:

```
entstat -r ent0; ping -f 10.11.12.13 64 2048; entstat ent0
```

In other cases it may be of interest to collect Ethernet statistics for a fixed time frame. This can be done using **entstat** as shown in the following command:

```
entstat -r ent0;sleep 300;entstat ent0
```

The numbers of packets, bytes, and broadcasts transmitted and received depend on many factors, like the applications running on the system or the number of systems connected to the subnetwork. There is no rule about how much is too much. Monitoring an Ethernet adapter on a regular basis using **entstat** can point out possible problems before users notice any slowdown. The problem can be taken care of by redesigning the network layout or tuning the adapter parameters using the **chdev** command, or tuning network options using the **no** command. (See Chapter 34, “The no command” on page 665.)

29.3 estat

The syntax of the **estat** command is:

```
/usr/lpp/ssp/css/css/estat [ -d -r ] Device_Name
```

Flags

- d** Displays all device-driver statistics, including the device-specific statistics.
- r** Resets all statistics to their initial values. This flag can only be issued by privileged users.

Parameters

Device_Name The name of the switch device, for example `css0`. If an invalid device name is specified, the **estat** command will produce an error message stating that it could not connect to the device.

29.3.1 Information about measurement and sampling

The **estat** command used without flags provides generic statistics that consist of transmit statistics, receive statistics, and general statistics. This includes packets and bytes transmitted and received, and information about hardware and software queue usage as well as error counters. If the **-d** flag is used, device-specific statistics are displayed along with the device-driver statistics. Currently device-specific statistics show only the current number of communication windows opened by the adapter.

The **estat** command provides a snapshot of the device-driver statistics. The output of the **estat** command consists of five sections; the title fields, the transmit statistics fields, the receive statistics fields, the general statistics fields, and the adapter specific statistic fields.

Refer to the *RS/6000 SP System Performance Tuning Update*, SG24-5340 for more detailed information about tuning an RS/6000 SP system, and the Internet site <http://techsupport.services.ibm.com/server/spperf/> for the latest information about tuning topics for the RS/6000 SP system.

29.3.2 Examples for estat

The output of **estat** always shows the device-driver statistics. If the **-d** flag is used, the device-specific statistics are also displayed.

Example 29-4 shows the output of **estat**.

Example 29-4 Output of the estat command

```
# /usr/lpp/ssp/css/estat -d css0
-----
CSS STATISTICS (css0) :
Elapsed Time: 97 days 10 hours 6 minutes 36 seconds

Transmit Statistics:                               Receive Statistics:
-----
Packets: 9798614                               Packets: 5439592
Bytes: 2529885036                             Bytes: 600249096
Interrupts: 0                                    Interrupts: 5437107
Transmit Errors: 0                               Receive Errors: 0
Packets Dropped: 0                              Packets Dropped: 0
Max Packets on S/W Transmit Queue: 0            Bad Packets: 0
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 0                             Broadcast Packets: 0

General Statistics:
-----
No mbuf Errors: 0

High Performance Switch Specific Statistics:
-----
Windows open: 2
```

The major fields of interest concerning performance and performance monitoring are:

Elapsed Time	The real-time period that has elapsed since the last time the statistics were reset. During error recovery, when a hardware error is detected, part of the statistics may be reset. In this case another Elapsed Time is displayed in the middle of the output reflecting the time elapsed since the reset. In this example there was no such event so there is no additional Elapsed Time displayed.
Transmit and Receive Packets	The number of packets successfully transmitted and received by the device.

Transmit and Receive Bytes The number of bytes successfully transmitted and received by the device. These values and their related packet count can show how the system is using this network adapter. For example, transmit and receive values may be close to equal, or they may differ by a huge margin.

No mbuf Errors The number of times communications mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain mbufs to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. This may cause retransmission by the sending system, which increases load on the system as well as additional network load. The **netstat** command can be used to confirm this. For details refer to Chapter 31, “The netstat command” on page 619.

The RS/6000 SP switch adapter uses special communications memory buffers for all packets greater than 256 bytes. For better performance these buffer pools are allocated in pinned kernel memory. The device driver uses AIX mbufs only when these pinned buffer pools are exhausted. Use the **lsattr -E1 css0** command to get the current buffer pool settings. The attribute fields are *rpoolsize* for the receive buffer pool and *spoolsize* for the send buffer pool. The buffer pool sizes can be changed using the **/usr/lpp/ssp/css/cghcss -l css0 -a Attribute=Value** command, where **Attribute** is either *rpoolsize* or *spoolsize* and **Value** is the new buffer size in bytes. On systems using the RS/6000 SP Switch, a restart of the node is required to activate the new pool settings. On systems using the RS/6000 SP Switch2 the changes take place immediately.

Sometimes it is useful to know how many packets an application or task sends or receives. Use **/usr/lpp/ssp/css/estat -r Device_Name** to reset the counters to zero, then running the application or task. After the completion of the application or task, run **/usr/lpp/ssp/css/estat Device_Name** again to get this information. An example of using **estat** to monitor RS/6000 SP Switch statistics during execution of one program is:

```
alias estat=/usr/lpp/ssp/css/estat
estat -r css0; ping -f 10.10.10.200 8000 1024;estat css0
```


In other cases it may be of interest to collect RS/6000 SP Switch statistics for a fixed time frame. This can be done using **estat** as shown in the following commands:

```
alias estat=/usr/lpp/ssp/css/estat
estat -r css0;sleep 300;estat css0
```

The numbers of packets and bytes transmitted and received depend on many factors, like the applications running on the system or the number of systems connected to the subnetwork. There is no rule about how much is too much. Monitoring an RS/6000 SP Switch adapter on a regular basis using **estat** can point out possible problems before users notice any slowdown. The problem can be taken care of by tuning the adapter parameters using the **chgcss** command or tuning network options using the **no** command. (See Chapter 34, “The no command” on page 665.)

Upcoming releases and versions of IBM PSSP may add new features and tools for monitoring and tuning the RS/6000 SP Switch. New RS/6000 SP Switch hardware may offer new monitoring and tuning options as well. For detailed and up-to-date information about RS/6000 SP switch tuning, refer to <http://techsupport.services.ibm.com/server/spperf/> and the IBM Redbook *RS/6000 SP System Performance Tuning Update*, SG24-5340.

29.4 fddistat

The syntax of the **fddistat** command is:

```
fddistat [ -d -r -t ] Device_Name
```

Flags

- d** Displays all device-driver statistics, including the device-specific statistics. Some FDDI adapters do not support the device-specific statistic. In this case the output will be the same as it would be without the **-d** flag.
- r** Resets all the statistics back to their initial values. This flag can only be issued by privileged users.
- t** Toggles debug trace in some device drivers.

Parameters

Device_Name The name of the FDDI device, for example, fddi0. If an invalid **Device_Name** is specified, the **fddistat** command produces an error message stating that it could not connect to the device.

29.4.1 Information about measurement and sampling

The **fdlistat** command used without flags provides generic statistics that consist of transmit statistics, receive statistics, and general statistics. This includes packets and bytes transmitted and received, and information about hardware and software queue usage as well as error counters. If the **-d** flag is used, device-specific statistics are displayed along with the device-driver statistics.

The **fdlistat** command provides a snapshot of the device-driver statistics collected by the NDD. The header file `/usr/include/sys/ndd.h` defines the used data structure `ndd_genstats`. **fdlistat** uses a device-dependent routine defined in the ODM to display the device-specific statistics. This device-dependent routine is a command that will be executed using `fork()` and `exec()` out of **fdlistat**. In a busy system there may be some delay doing this. If the system is running out of resources (for example low on memory), the necessary `fork()` may fail. All the device-dependent routines can be found using the command **odmget -q attribute=addl_stat PdAt**. All statistic values displayed by **fdlistat** are the absolute values since startup or the last reset of these values, which is done by using **fdlistat -r Device_Name**.

Hardware error recovery may cause some statistic values to be reset. If this happens, a second `Elapsed Time` is displayed in the middle of the statistic's output reflecting the time elapsed since the reset.

The device-driver statistics are read out of the NDD at execution time of **fdlistat**. The device-specific statistics are read from the device driver using the `ioctl()` system call. The data gets displayed and **fdlistat** exits. Using the **-r** flag, **fdlistat** first displays the current statistic values and then resets them.

Some adapters may not support a specific statistic. In this case the non-supported statistic fields are always 0.

The output of the **fdlistat** command consists of five sections: the title fields, the transmit statistics fields, the receive statistics fields, the general statistics fields, and the adapter-specific statistic fields. Refer to *AIX 5L Version 5.2 Commands Reference* for a description of all output fields.

29.4.2 Examples for fdlistat

The output of **fdlistat** always shows the device-driver statistics as shown in Example 29-5 on page 557. If the **-d** flag is used and the adapter supports it, the device-specific statistics are displayed as well.

Example 29-5 Using fddistat to display FDDI device-driver statistics

```
# fddistat fddi0
-----
FDDI STATISTICS (fddi0) :
Elapsed Time: 1 days 23 hours 24 minutes 55 seconds

Transmit Statistics:                                Receive Statistics:
-----
Packets: 61478352                                Packets: 54719134
Bytes: 51091616874                               Bytes: 81586386390
Interrupts: 1235849                               Interrupts: 35205866
Transmit Errors: 1                                  Receive Errors: 0
Packets Dropped: 2751646                          Packets Dropped: 2486
                                                    Bad Packets: 0

Max Packets on S/W Transmit Queue: 250
S/W Transmit Queue Overflow: 2751645
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 1340                          Broadcast Packets: 87866
Multicast Packets: 2                               Multicast Packets: 0

General Statistics:
-----
No mbuf Errors: 36455

SMT Error Word: 00000000                          SMT Event Word: 00000000
Connection Policy Violation: 0000                 Port Event: 0000
Set Count Hi: 0000                                Set Count Lo: 0000
Adapter Check Code: 0000                          Purged Frames: 16263

ECM State Machine:      IN
PCM State Machine Port A: ACTIVE
PCM State Machine Port B: ACTIVE
CFM State Machine Port A: THRU
CFM State Machine Port B: THRU
CF State Machine:      THRU
MAC CFM State Machine:  PRIMARY
RMT State Machine:      RING_OP

Driver Flags: Up Broadcast Running
              Simplex AlternateAddress 64BitSupport
```

The major fields of interest concerning performance and performance monitoring are:

Elapsed Time	The real-time period that has elapsed since the last time the statistics were reset. During error recovery, when a hardware error is detected part of the statistics may be reset. In this case another Elapsed Time is displayed in the middle of the output reflecting the time elapsed since the reset. In this example there was no such event so there is no additional Elapsed Time displayed.
Transmit and Receive Packets	The number of packets successfully transmitted and received by the device.
Transmit and Receive Bytes	The number of bytes successfully transmitted and received by the device. These values and their related packet count can show how the system is using this network adapter. For example transmit and receive values may be close to equal, or they may differ by a huge margin.
Transmit and Receive Interrupts	The number of transmit and receive interrupts received by the driver from the adapter. If these counters increase fast, then the number of interrupts to be handled by the operating system may reach a level where overall system performance may be affected. Other monitoring tools like <code>vmstat</code> can be used to control the interrupts per second handled by the system.
Max Packet on S/W Transmit Queue	The maximum number of outgoing packets ever queued to the software transmit queue. If this value reaches the <code>xt_que_size</code> set for the adapter then the <code>xt_que_size</code> of the adapter is not set to an adequate value. The command <code>lsattr -El Device_Name</code> , like <code>lsattr -El fddi0</code> , shows the current adapter settings including <code>xt_que_size</code> . Use <code>SMIT</code> or <code>chdev</code> to increase <code>xt_que_size</code> if necessary and possible. The possible values allowed to set can be found using the ODM as shown in Example 29-7 on page 565 or the <code>lsattr -Rl fddi0 -a xt_que_size</code> command.
S/W Transmit Queue Overflow	The number of outgoing packets that overflowed the software transmit queue. If this is not zero then you need to increase the transmit queue size <code>xt_que_size</code> , as shown in the description for the field Max Packets on S/W Transmit Queue.
Current S/W + H/W Transmit Queue Length	The number of pending outgoing packets on either the software transmit queue or the hardware transmit queue. This reflects the current load on the adapter. This is the sum of the Current SW Transmit Queue Length and Current HW Transmit Queue Length fields.

Broadcast Packets	The number of broadcast packets transmitted and received without any error. A high value compared to the total transmitted and received packets indicates that the system is sending and receiving many broadcasts. Broadcasts increase network load, and may increase the load on all other systems on the same subnetwork.
No mbuf Errors	The number of times communications mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain mbufs to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. This may cause retransmission by the sending system, which increases load on the system as well as additional network load. The netstat command can be used to confirm this. For details refer to Chapter 31, “The netstat command” on page 619.

Some FDDI adapters for AIX use mbuf buffers for their transmit queue. In this case Packets Dropped in the transmit statistics could be caused by a No mbuf Errors count greater than zero.

If the statistics for errors, such as the transmit errors, are increasing fast they should be corrected first; error counters should stay close to zero. Some errors may be caused by hardware problems. These hardware problems must be fixed before any software tuning is performed.

Example 29-5 on page 557 shows the output of **fdiostat** on a system with two different problems.

- ▶ The field `S/W Transmit Queue Overflow` shows a large number of overflows. This is too high for the two days of `Elapsed Time`. The value 250 for the field `Max Packets on S/W Transmit Queue` indicates that the `tx_queue_size` for this adapter may be set to 250. `lsattr -El fddi0` and `lsattr -Rl fddi0 -a tx_queue_size` should be used to see if the transmit queue size can be increased. `SMIT` or `chdev` should then be used to raise the value for `tx_queue_size`.
- ▶ The field `No mbuf Errors` indicates a shortage of mbufs. The **netstat -m** command should be used to verify this; refer to Chapter 31, “The netstat command” on page 619 for details about the **netstat** command and the proper tuning in case of mbuf errors.

Fixing the software transmit queue overflows and the mbuf errors will reduce, if not eliminate, the dropped packets errors. Verification can be done by resetting the FDDI device-driver statistics with **fddistat -r fddi0**, then running the system normally for two days. After these two days another **fddistat fddi0** output should be created and compared to the previous one.

Sometimes it is useful to know how many packets an application or task sends or receives. Use **fddistat -r Device_Name** to reset the counters to zero, then run the application or task. After the completion of the application or task, run **fddistat Device_Name** again to get this information. An example for using **fddistat** to monitor FDDI statistics during execution of one program is:

```
fddistat -r fddi0; ping -f 10.10.10.10 64 1024; fddistat fddi0
```

In other cases it may be of interest to collect FDDI statistics for a fixed time frame. This can be done using **fddistat** as shown in the following command:

```
fddistat -r fddi0;sleep 3600;fdistat fddi0
```

The numbers of packets, bytes, and broadcasts transmitted and received depend on many factors, such as the applications running on the system or the number of systems connected to the subnetwork. There is no rule about how much is too much. Monitoring an FDDI adapter on a regular basis using **fddistat** can point out possible problems before users notice any slowdown. The problem can be taken care of by redesigning the network layout, or tuning the adapter parameters using the **chdev** command or network options using the **no** command. (See Chapter 34, “The no command” on page 665.)

29.5 tokstat

The syntax of the **tokstat** command is:

```
tokstat [ -d -r -t ] Device_Name
```

Flags

- d** Displays all the device-driver statistics, including the device-specific statistics.
- r** Resets the statistics to their initial values. This flag can only be issued by privileged users.
- t** Toggles debug trace in some device drivers.

Parameters

Device_Name The name of the token-ring device, such as tok0. If an invalid device name is specified, the **tokstat** command produces an error message stating that it could not connect to the device.

29.5.1 Information about measurement and sampling

The **tokstat** command used without flags provides generic statistics that consist of transmit statistics, receive statistics, and general statistics. This includes packets and bytes transmitted and received, information about hardware and software queue usage as well as error counters. Using the **-d** flag displays device-specific statistics in addition to the device-driver statistics.

The **tokstat** command provides a snapshot of the device-driver statistics collected by the NDD. The header file `/usr/include/sys/ndd.h` defines the used data structure `ndd_genstats` as well as the `ioctl()` operation `NDD_GET_ALL_STATS`, which is used to read the data from the NDD. **tokstat** uses a device-dependent routine defined in the ODM to display the device-specific statistics. This device-dependent routine is a command that will be executed using `fork()` and `exec()` out of **tokstat**. In a busy system there may be some delay doing this. In case the system is running out of resources (for example, low on memory), the necessary `fork()` may fail. All device-dependent routines can be found using the command `odmget -q attribute=addl_stat PdAt`. All statistic values displayed by **tokstat** are the absolute values since startup or the last reset of these values, which is done by using `tokstat -r Device_Name`.

Hardware error recovery may cause some statistic values to be reset. If this happens, a second Elapsed Time is displayed in the middle of the statistic's output reflecting the time elapsed since the reset.

The device-driver statistics are read out of the NDD at execution time of **tokstat**. The device-specific statistics are read from the device driver using the `ioctl()` system call. The data gets displayed and **tokstat** exits. Using the **-r** flag, **tokstat** first displays the current statistic values and then resets them.

Some adapters may not support a specific statistic. In this case the non-supported statistic fields are always zero.

The output of the **tokstat** command consists of five sections: the title fields, the transmit statistics fields, the receive statistics fields, the general statistics fields, and the adapter specific statistic fields. Refer to the *AIX 5L Version 5.2 Commands Reference* for a description of all output fields.

29.5.2 Examples for tokstat

The output of **tokstat** always shows the device-driver statistics. If the **-d** flag is used, the device-specific statistics are displayed.

Example 29-6 shows the output of **tokstat** including the device-specific statistics.

Example 29-6 Displaying token-ring device-driver statistics

```
# tokstat -d tok0
-----
TOKEN-RING STATISTICS (tok0) :
Device Type: IBM PCI Tokenring Adapter (14103e00)
Hardware Address: 00:60:94:8a:07:5b
Elapsed Time: 0 days 3 hours 27 minutes 47 seconds

Transmit Statistics:                               Receive Statistics:
-----
Packets: 48476                                  Packets: 67756
Bytes: 41102959                                Bytes: 38439965
Interrupts: 13491                               Interrupts: 67733
Transmit Errors: 0                                Receive Errors: 0
Packets Dropped: 0                              Packets Dropped: 0
Bad Packets: 0

Max Packets on S/W Transmit Queue: 890
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 10                          Broadcast Packets: 26634
Multicast Packets: 0                              Multicast Packets: 4341
Timeout Errors: 0                                Receive Congestion Errors: 0
Current SW Transmit Queue Length: 0
Current HW Transmit Queue Length: 0

General Statistics:
-----
No mbuf Errors: 0                               Lobe Wire Faults: 0
Abort Errors: 0                                  AC Errors: 0
Burst Errors: 0                                  Frame Copy Errors: 0
Frequency Errors: 0                              Hard Errors: 0
Internal Errors: 0                               Line Errors: 0
Lost Frame Errors: 0                            Only Station: 0
Token Errors: 0                                 Remove Received: 0
Ring Recovered: 0                              Signal Loss Errors: 0
Soft Errors: 0                                  Transmit Beacon Errors: 0

Driver Flags: Up Broadcast Running
AlternateAddress 64BitSupport ReceiveFunctionalAddr
16 Mbps
```


IBM PCI Tokenring Adapter (14103e00) Specific Statistics:

Media Speed Running: 16 Mbps Half Duplex
Media Speed Selected: 16 Mbps Full Duplex
Receive Overruns : 0
Transmit Underruns : 0
ARI/FCI errors : 0
Microcode level on the adapter :00IHSS2B4
Num pkts in priority sw tx queue : 0
Num pkts in priority hw tx queue : 0
Open Firmware Level : 001PXHL00

The major fields of interest concerning performance and performance monitoring are:

Elapsed Time	The real-time period that has elapsed since the last time the statistics were reset. During error recovery, when a hardware error is detected, part of the statistics may be reset. In this case another Elapsed Time is displayed in the middle of the statistic's output reflecting the time elapsed since the reset. In this example there was no such event so there is no additional Elapsed Time displayed.
Transmit and Receive Packets	The number of packets successfully transmitted and received by the device.
Transmit and Receive Bytes	The number of bytes successfully transmitted and received by the device. These values and their related packet count can show how the system is using this network adapter. For example transmit and receive values may be close to equal, or they may differ by a huge margin.
Transmit and Receive Interrupts	The number of transmit and receive interrupts received by the driver from the adapter. If these counters increase fast, then the number of interrupts to be handled by the operating system may reach a level where overall system performance may be affected. Other monitoring tools like vmstat can be used to control the interrupts per second handled by the system.

Max Packet on S/W Transmit Queue	The maximum number of outgoing packets ever queued to the software transmit queue. If this value reaches the <code>xt_que_size</code> set for the adapter then the <code>xt_que_size</code> of the adapter is not set to an adequate value. The command lsattr -E1 <i>Device_Name</i> , like lsattr -E1 tok0 , shows the current adapter settings including <code>xt_que_size</code> . Use SMIT or chdev to increase <code>xt_que_size</code> if necessary and possible. The possible values allowed to set can be found using the ODM as shown in Example 29-7 on page 565 or the lsattr -R1 tok0 -a xmt_que_size command.
S/W Transmit Queue Overflow	The number of outgoing packets that overflowed the software transmit queue. If this is not zero, you must increase the transmit queue size <code>xt_que_size</code> , as shown in the description for the field <code>Max Packets on S/W Transmit Queue</code> .
Current S/W + H/W Transmit Queue Length	The number of pending outgoing packets on either the software transmit queue or the hardware transmit queue. This reflects the current load on the adapter. This is the sum of the <code>Current SW Transmit Queue Length</code> and <code>Current HW Transmit Queue Length</code> fields.
Broadcast Packets	The number of broadcast packets transmitted and received without any error. A high value compared to the total transmitted and received packets indicates that the system is sending and receiving many broadcasts. Broadcasts increase network load, and may increase the load on all other systems on the same subnetwork.
Current SW Transmit Queue Length	The number of outgoing packets currently on the software transmit queue.
Current HW Transmit Queue Length	The number of outgoing packets currently on the hardware transmit queue.
No mbuf Errors	The number of times communications mbufs were not available to the device driver. This usually occurs during receive operations when the driver must obtain mbufs to process inbound packets. If the mbuf pool for the requested size is empty, the packet will be discarded. This may cause retransmission by the sending system, which increases load on the system as well as additional network load. The netstat command can be used to confirm this. For details refer to Chapter 31, “The netstat command” on page 619.

Example 29-7 shows how to get the possible `xmt_que_size` values for `tok0`.

Example 29-7 Get the possible `xmt_que_size` values for `tok0`

```
# odmget -q name=tok0 CuDv
CuDv:
  name = "tok0"
  status = 1
  chgstatus = 2
  ddins = "pci/cstokdd"
  location = "10-68"
  parent = "pci0"
  connwhere = "104"
  PdDvLn = "adapter/pci/14103e00"

# odmget -q 'uniquetype=adapter/pci/14103e00 and attribute=xmt_que_size' PdAt
PdAt:
  uniquetype = "adapter/pci/14103e00"
  attribute = "xmt_que_size"
  deflt = "8192"
  values = "32-16384,1"
  width = ""
  type = "R"
  generic = "DU"
  rep = "nr"
  nls_index = 7
```

In Example 29-7 on page 565, the following happens:

- ▶ The first `odmget` reads the adapter data from ODM class `CuDv`. We need the value of the `PdDvLn` field, which identifies the adapter in the `PdAt` class for the second `odmget`.
- ▶ The second `odmget` shows the default value for `xmt_que_size` in the `deflt` field and the possible values in the `values` field. In this sample the `xmt_que_size` can be set to values between 32 and 16384 in steps by 1 using the `chdev` command:

```
chdev -l tok -a xmt_que_size=16384 -P
```

Note: The `chdev` command cannot change an active adapter. Using the `-P` flag forces `chdev` to only change the value in ODM. After the next reboot this new value gets used.

If the statistics for errors, for example transmit errors, are increasing fast, then these errors should be corrected first. Some errors may be caused by hardware problems, which should be fixed before any software tuning is performed. These error counters should stay close to zero.

Sometimes it is useful to know how many packets an application or task sends or receives. Use **tokstat -r *Device_Name*** to reset the counters to zero, then run the application or task. After the completion of the application or task, run **tokstat *Device_Name*** again to get this information. An example for using **tokstat** to monitor token-ring statistics during execution of one program is:

```
tokstat -r tok0; ping -f 10.10.10.10 64 1024; tokstat tok0
```

In other cases it may be of interest to collect token-ring statistics for a fixed time frame. This can be done using **tokstat** as shown in the following command:

```
tokstat -r tok0;sleep 300;tokstat tok0
```

The numbers of packets, bytes, and broadcasts transmitted and received depend on many factors, such as the applications running on the system or the number of systems connected to the subnetwork. There is no rule about how much is too much. Monitoring a token-ring adapter on a regular basis using **tokstat** can point out possible problems before users notice any slowdown. The problem can be taken care of by redesigning the network layout or tuning the adapter parameters using the **chdev** command or tuning network options using the **no** command. (See Chapter 34, “The no command” on page 665.)

TCP/IP packet tracing tools

This chapter discusses network packet tracing tools. The tools consist of:

- ▶ IP packet tracing commands: **iptrace**, **ipreport**, and **ipfilter**
- ▶ TCP packet tracing commands: **tcpdump** and **trpt**

These commands reside in `/usr/sbin` and are part of the `bos.net.tcp.server` fileset, which is installable from the AIX base installation media.

30.1 Network packet tracing tools

The **iptrace** command records Internet packets received from configured network interfaces. Command flags provide a filter so that **iptrace** only traces packets meeting specific criteria. Monitoring the network traffic with **iptrace** can often be very useful in determining why network performance is not as expected.

The **ipreport** command formats the data file generated by **iptrace**. The **ipreport** command generates a readable trace report from the specified trace file created by the **iptrace** command. Monitoring the network traffic with **iptrace** or **tcpdump** can often be very useful in determining why network performance is not as expected. The **ipreport** command will format the binary trace reports from either of these commands, or network sniffer, into an ASCII (or EBCDIC) formatted file.

The **ipfilter** command sorts the output file created by the **ipreport** command, provided the **-r** (for NFS/RPC reports) and **-s** (for all reports) flags have been used in generating the report. The **ipfilter** command provides information about NFS, UDP, TCP, IPX, and ICMP headers in table form. Information can be displayed together, or separated by headers into different files. It can also provide separate information about NFS calls and replies.

The **tcpdump** command prints out the headers of packets captured on a network interface. The **tcpdump** command is a very powerful network packet trace tool that allows a wide range of packet filtering criteria. These criteria can range from simple trace-all options to detailed byte and bit level evaluations in packet headers and data parts.

The **trpt** command performs protocol tracing on TCP sockets. Monitoring the network traffic with **trpt** can be useful in determining how applications that use the TCP connection oriented communications protocol perform.

For more detailed information about the TCP/IP protocols, refer to:

- ▶ 1.5, "Network performance" on page 31
- ▶ *AIX 5L Version 5.2 Performance Management Guide*
- ▶ *AIX 5L Version 5.2 System Management Guide: Communications and Networks*
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376
- ▶ *RS/6000 SP System Performance Tuning Update*, SG24-5340
- ▶ Appropriate Request For Comment (RFC) at <http://www.rfc-editor.org/>

30.2 iptrace

The syntax of the **iptrace** command is:

```
iptrace [-a] [-e] [-d Host [-b]] [-u][-s Host [-b]] [-p Port_list]
[-P Protocol_list] [-i Interface] [-L Log_size ] LogFile
```

Flags

-a	Suppresses ARP packets.
-b	Changes the -d or -s flags to bidirectional mode.
-d Host	Records packets headed for the destination host specified by the Host variable.
-e	Enables promiscuous mode on network adapters that support this function.
-i Interface	Records packets received on the interface specified by the Interface variable.
-L Log_size	This option causes iptrace to log data such that the LogFile is copied to LogFile.old at the start, and every time it becomes approximately Log_size bytes long.
-P Protocol_list	Records packets that use the protocol specified by the Protocol_list variable.
-p Port_list	Records packets that use the port number specified by the Port_list variable.
-s Host	Records packets coming from the source host specified by the Host variable.
-u	Unloads the kernel extension that was loaded by the iptrace daemon at startup.

Parameters

LogFile	Specifies the name of the file to save the results of the network trace.
Snaplen	Specifies the number of bytes of data from each packet.
Interface	Network interface to listen for packets on.
Host	If used with the -b and the -d flag, iptrace records packets both going to and coming from the host specified by the Host variable. The Host variable can be a host name or an Internet address in dotted-decimal format.
Log_size	When the output file for network trace data reaches Log_size bytes, it is copied to LogFile.old. Using this flag is

also an indicator that the LogFile file should be copied to LogFile.old at the start.

Protocol_list A list of protocol specifications to monitor. Several protocols can be monitored by a comma-separated list of identifiers. The Protocol_list variable can be a decimal number or name from the /etc/protocols file.

Port_list A list of service/port specifications to monitor. Several services/ports can be monitored by a comma-separated list of identifiers. The Port_list variable can be a decimal number or name from the /etc/services file.

TCP/IP protocol and services tables

Table 30-1 is an extraction from the /etc/protocols file that shows some interesting protocol types and their numeric value.

Table 30-1 Some important protocols

Symbolic name	Numeric ID	Protocol	Description
ip	0	IP	Dummy for the Internet Protocol
icmp	1	ICMP	Internet control message protocol
igmp	2	IGMP	Internet group multicast protocol
tcp	6	TCP	Transmission control protocol
udp	17	UDP	User datagram protocol

Table 30-2 is an extraction from the /etc/services file that shows some interesting services and ports, and the protocol used on those ports.

Table 30-2 Selection from /etc/services

Symbolic name	Port	Protocol	Description
echo	7	tcp	Used by the ping command
echo	7	udp	Used by the ping command
ftp-data	20	tcp	Used by the ftp command
ftp	21	tcp	Used by the ftp command
telnet	23	tcp	Used by the telnet command
smtp	25	tcp	Used by the mail commands
domain	53	udp	Used by nameserver commands

Symbolic name	Port	Protocol	Description
pop	109	tcp	Used by postoffice mail commands
pop3	110	tcp	Used by postoffice3 mail commands
exec	512	tcp	Used by remote commands
login	513	tcp	Used by remote commands
shell	514	tcp	Used by remote commands
printer	515	tcp	Used by print spooler commands
route	520	udp	Used by router (routed commands)

30.2.1 Information about measurement and sampling

The **iptrace** command can monitor more than one network interface at the same time, such as the *SP Switch* network interfaces, and not only one as with the **tcpdump** command (see 30.8, “tcpdump” on page 587). With the **iptrace** command the kernel copies the whole network packet to user space (to the monitoring **iptrace** command) from the kernel space. This can result in a lot of dropped packets, especially if the number of monitored interfaces has not been limited by using the **-i** Interface option to reduce the number of monitored interfaces.

Because network tracing can produce large amounts of data, it is important to limit the network trace either by *scope* (what to trace) or *amount* (how much to trace). Unlike the **tcpdump** command, the **iptrace** command does not offer many options to reduce the scope of the network trace. The **iptrace** command also relies on the **ipreport** command (see 30.3, “ipreport” on page 572) to format the binary network trace data into a readable format (unlike **tcpdump** which can do both). Note that the **iptrace** command will perform any filtering of packets in user space and not in kernel space as the **tcpdump** command does (unless the **-B** flag is used).

The **iptrace** command uses either the network trace kernel extension (`net_xmit_trace` kernel service), which is the default method, or the Berkeley Packet Filter (BPF) packet capture library to capture network packets (**-u** flag). The **iptrace** command can either run as a daemon or under the System Resource Controller (SRC).

For more information about the BPF, see “Packet Capture Library Subroutines” in *AIX 5L Version 5.2 Technical Reference: Communications, Volume 2*.

For more information about the `net_xmit_trace` kernel service, see *AIX 5L Version 5.2 Technical Reference: Kernel and Subsystems, Volume 1*.

30.3 ipreport

The syntax of the **ipreport** command is:

```
ipreport [-CenrsSvx1NT] [-c Count] [-j Pktnum] [-X Bytes] LogFile
```

Flags

-c Count	Display Count number of packets.
-C	Validate checksums.
-e	Show EBCDIC instead of ASCII.
-j Pktnum	Jump to packet number Pktnum.
-n	Number of packets.
-N	Do not do name resolution.
-r	Decodes remote procedure call (RPC) packets.
-s	Start lines with protocol indicator strings.
-S	Input file was generated on a sniffer.
-T	Input file is in tcpdump format.
-v	Verbose.
-x	Print packet in hex.
-X Bytes	Limit hex dumps to bytes.
-1	Compatibility trace was generated on AIX V3.1.

Parameters

LogFile	The LogFile parameter specifies the name of the file containing the results of the Internet Protocol trace. This file can be generated by the iptrace or tcpdump commands.
Count	Number of packets to display.
Bytes	Number of bytes to display for hex dumps.
Pktnum	Start reporting from packet number Pktnum.

30.3.1 Information about measurement and sampling

The **ipreport** uses a binary input file from either the **iptrace** or **tcpdump** commands¹. Usually these network trace commands are executed in such a way that they create a binary file that is then used by **ipreport**. The **ipreport** command can, however, be used in a command pipeline with the **tcpdump** command.

You must be aware that tracing and analyzing network traffic is not easy. You should understand how different applications communicate, what protocols they use, how these network protocols work, and what effect the network tunables have on the protocols traffic flow.

For schematic information about frame and packet headers, refer to “Packet header formats” on page 535.

30.4 ipfilter

The syntax of the **ipfilter** command is:

```
ipfilter [-f [ u n t x c a ]] [-s [ u n t x c a ]] [-n [ -d milliseconds ]]
        ipreport_output_file
```

Flags

- | | |
|-------------------------|--|
| u n t x c a | Specifies operation headers (UDP, NFS, TCP, IPX, and ICMP, and ATM respectively). |
| -d milliseconds | Only Call/Reply pairs whose elapsed time is greater than milliseconds are to be shown. |
| -f [u n t x c] | Selected operations are to be shown in ipfilter.all |
| -n | Generates an nfs.rpt file. |
| -s [u n t x c] | Separate files are to be produced for each of the selected operations. |

Parameters

- | | |
|-----------------------------|---|
| milliseconds | Call/Reply pairs whose elapsed time is greater than milliseconds. |
| ipreport_output_file | Name of file created by the ipreport command. |

¹ Or network sniffer device.

30.4.1 Information about measurement and sampling

ipfilter will read a file created by **ipreport**. The **ipreport** file has to be created by using the **-s** or **-rsn** flag, which specifies that **ipreport** will prefix each line with the protocol header. If no option flags are specified, **ipfilter** will generate a file containing all protocols called **ipfilter.all**.

30.4.2 Protocols and header type options

Table 30-3 shows the mapping between protocol (header types) and the generated output file depending on how the option flags are specified to the **ipfilter** command:

Table 30-3 *ipfilter* header types and options

Header Type	Header type option	Output filename (-s)	Output filename (-f)
NFS (RPC)	n	ipfilter.nfs	ipfilter.all
TCP	t	ipfilter.tcp	ipfilter.all
UDP	u	ipfilter.udp	ipfilter.all
ICMP	c	ipfilter.icmp	ipfilter.all
IPX (PC protocol)	x	ipfilter.ipx	ipfilter.all

30.5 Examples for iptrace, ipreport, and ipfilter

To trace a specific network interface, use the **-i** option with the **iptrace** command as shown in Example 30-1 to trace all traffic on the **tr0** interface (token-ring).

Example 30-1 Using iptrace to trace a network interface

```
# startsrc -s iptrace -a "-i tr0 /tmp/iptrace.tr0"&&  
read &&  
stopsrc -s iptrace
```

Example 30-2 shows a short output from the network trace started in the previous example that shows the **ECHO_REQUEST** from 1.39.7.84 and the **ECHO_REPLY** from 1.3.1.164 (probably someone was using the **ping** command).

Example 30-2 Using ipreport

```
# ipreport -sn /tmp/iptrace.tr0  
IPTRACE version: 2.0
```

```
Packet Number 1
```

```

TOK: ==== ( 106 bytes received on interface tr0 )==== 16:20:46.509067872
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 08:00:5a:fe:21:06, dst = 00:60:94:8a:07:5b]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.39.7.84 > (sp3tr35.itso.ibm.com)
IP:   < DST =      1.3.1.164 > (wlmhost)
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=16278, ip_off=0
IP:   ip_ttl=245, ip_sum=6af1, ip_p = 1 (ICMP)
ICMP: icmp_type=8 (ECHO_REQUEST) icmp_id=12234 icmp_seq=3743

Packet Number 2
TOK: ==== ( 106 bytes transmitted on interface tr0 )==== 16:20:46.509234785
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 00:60:94:8a:07:5b, dst = 08:00:5a:fe:21:06]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.3.1.164 > (wlmhost)
IP:   < DST =      1.39.7.84 > (sp3tr35.itso.ibm.com)
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=45289, ip_off=0
IP:   ip_ttl=255, ip_sum=ef9d, ip_p = 1 (ICMP)
ICMP: icmp_type=0 (ECHO_REPLY) icmp_id=12234 icmp_seq=3743
...(lines omitted)...

```

30.5.1 TCP packets

Example 30-3 shows how to trace bi-directional (-b) TCP connections (-P tcp) to and from system 1.1.1.114, suppressing ARP packets (-a) and saving the output in a file (/tmp/iptrace.tcp). The **iptrace** command runs until the ENTER key is pressed (read shell built-in function), and the **stopsrc** command stops the trace. The double ampersand (&&) means that if the previous command was OK, then execute the following command.

Example 30-3 Using iptrace to trace tcp to and from a system

```

# startsrc -s iptrace -a "-a -b -P tcp -d 1.1.1.114 /tmp/iptrace.tcp"&&
read &&
stopsrc -s iptrace

```

To obtain a readable report from the **iptrace** binary data, use the **ipreport** command, as Example 30-4 on page 576 shows.

Example 30-4 Using ipreport

```
# ipreport -s /tmp/iptrace.tcp
IPTRACE version: 2.0

TOK: ==== ( 62 bytes received on interface tr0 )==== 11:28:29.853288442
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:60:94:87:0a:87, dst = 00:60:94:8a:07:5b]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.3.1.114 > (3b-043)
IP:   < DST =      1.3.1.164 > (wlmhost)
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=50183, ip_off=0 DF
IP:   ip_ttl=128, ip_sum=21ad, ip_p = 6 (TCP)
TCP:  <source port=2423, destination port=23(telnet) >
TCP:  th_seq=357cdd86, th_ack=a0005f0b
TCP:  th_off=5, flags<ACK>
TCP:  th_win=17155, th_sum=3c19, th_urp=0
...(lines omitted)...
```

30.5.2 UDP packets

Example 30-5 shows how to trace bi-directional (-b) UDP connections (-P udp) to and from system 1.1.1.114, suppressing ARP packets (-a) and saving the output in a file (/tmp/iptrace.udp). The **iptrace** command runs until the ENTER key is pressed (read shell built-in function), and the **stopsrc** command stops the trace. The double ampersand (&&) means that if the previous command was OK, then execute the following command.

Example 30-5 Using iptrace to trace udp to and from a system

```
# startsrc -s iptrace -a "-a -b -P udp -d 1.1.1.114 /tmp/iptrace.udp" &&
read &&
stopsrc -s iptrace
```

To obtain a readable report from the **iptrace** binary data, use the **ipreport** command, as Example 30-6 shows.

Example 30-6 Using ipreport

```
# ipreport -s /tmp/iptrace.udp
IPTRACE version: 2.0

TOK: ==== (202 bytes received on interface tr0 )==== 11:30:03.808584556
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
```

```

TOK: [ src = 80:60:94:87:0a:87, dst = c0:00:00:04:00:00]
TOK: routing control field = 8270, 0 routing segments
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.3.1.114 > (3b-043)
IP:   < DST = 229.55.150.208 >
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=178, ip_id=50228, ip_off=0
IP:   ip_ttl=10, ip_sum=658a, ip_p = 17 (UDP)
UDP:  <source port=1346, <destination port=1345 >
UDP:  [ udp length = 158 | udp checksum = fbf5 |
UDP:  00000000    24020209 0133064c 6f636174 65220100 |$....3.Locate"..|
UDP:  00000010    24020209 02330750 726f6475 63740c02 |$....3.Product..|
UDP:  00000020    24020209 03330547 686f7374 0c032402 |$....3.Ghost..$.|
UDP:  00000030    02090433 09436f6d 706f6e65 6e740c04 |...3.Component..|
UDP:  00000040    24020209 05330d43 6f6e6669 675f5365 |$....3.Config_Se|
UDP:  00000050    72766572 0c052402 02090633 044e616d |rver..$...3.Nam|
UDP:  00000060    650c0620 149c207f 9b2abcc2 0a50c17a |e.. .. .*...P.z|
UDP:  00000070    02de9f5f 1789e437 ef240202 09073309 |..._...7.$...3.|
UDP:  00000080    4368616c 6c656e67 650c0720 08f79efd |Challenge.. ....|
UDP:  00000090    0bb44bf2 cb02 |..K...|
...(lines omitted)...

```

30.5.3 UDP domain name server requests and responses

Example 30-7 shows how to trace Domain Name Server (DNS) connections (-p domain), suppressing ARP packets (-a) and saving the output in a file (/tmp/iptrace.dns). The **iptrace** command runs until the ENTER key is pressed (read shell built-in function), and the **stopsrc** command stops the trace. The double ampersand (&&) means that if the previous command was OK, then execute the following command).

Example 30-7 Using iptrace to trace DNS

```

# startsrc -s iptrace -a "-a -p domain /tmp/iptrace.dns" &&
read &&
stopsrc -s iptrace

```

To obtain a readable report from the **iptrace** binary data, use the **ipreport** command, as Example 30-8 on page 578 shows.

Example 30-8 Using ipreport

```
# ipreport -s /tmp/iptrace.dns
IPTRACE version: 2.0

TOK: ====( 90 bytes transmitted on interface tr0 )=== 11:33:55.782893557
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 0, frame control field = 40
TOK: [ src = 00:60:94:8a:07:5b, dst = 00:20:35:3f:7e:11]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.3.1.164 > (wlmhost)
IP:   < DST =      1.3.1.2 > (dude.itso.ibm.com)
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=68, ip_id=28279, ip_off=0
IP:   ip_ttl=30, ip_sum=1987, ip_p = 17 (UDP)
UDP:  <source port=33681, <destination port=53(domain) >
UDP:  [ udp length = 48 | udp checksum = adae ]
DNS Packet breakdown:
  QUESTIONS:
    114.1.3.1.in-addr.arpa, type = PTR, class = IN
  ...(lines omitted)...
```

30.6 Examples for ipreport

In the following examples we show the use of **ipreport** with the **iptrace** and **tcpdump** commands.

30.6.1 Using ipreport with tcpdump

To use **ipreport** on data from **tcpdump**, use the **-T** flag with **ipreport** as in Example 30-9.

Example 30-9 Using ipreport with tcpdump

```
# tcpdump -w - | ipreport -rsT - | more
TCPDUMP

TOK: ====( 80 bytes on interface token-ring )=== 16:42:43.327359881
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 08:00:5a:fe:21:06, dst = 00:20:35:72:98:31]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      1.3.7.140 > (sox5.itso.ibm.com)
IP:   < DST =      1.3.1.41 >
```



```

IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=1500, ip_id=23840, ip_off=0
IP:   ip_ttl=57, ip_sum=442, ip_p = 6 (TCP)
IP:   truncated-ip, 1442 bytes missing
IP:   00000000   043804fa a8fb14da 0937db32 50107d78   |.8.....7.2P.|x|
IP:   00000010   33330000 863fcc52 996d64f2 577d2c2c   |33...?.R.md.W|,,|
IP:   00000020   c5f7c26a leed   |...j..|
...(lines omitted)...

```

Using the `tcpdump` command with the `-w` - (dash) flags specifies that `tcpdump` should write raw packets to `stdout` instead of parsing and printing them out. By specifying a dash (-) as the input file to `ipreport`, it will read from `stdin`. The `-rs` flags tells `ipreport` to start lines with protocol indicator strings and to be aware of RPC packets.

30.6.2 Using ipreport with iptrace

Example 30-10 shows how to trace a bidirectional connection between a server host and a client (remote node), save the network trace output in a file, wait for 30 seconds, and then stop the trace. After `iptrace` is stopped the `ipreport` command is executed to generate a readable report excluding host name lookup, only reporting on the 100 first packets starting from packet number 55, and including RPC information.

Example 30-10 ipreport from iptrace input

```

# startsrc -s iptrace -a "-a -b -d remotenode /tmp/iptrace.out" &&
> sleep 30 &&
> stopsrc -s iptrace
# ipreport -c 100 -j 55 -v -N -rs /tmp/iptrace.out
IPTRACE version: 2.0

TOK: ===( 62 bytes received on interface tr0 )=== 12:51:59.944658222
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:60:94:87:0a:87, dst = 00:60:94:8a:07:5b]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP: < SRC = 1.3.1.114 >
IP: < DST = 1.3.1.164 >
IP: ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=41714, ip_off=0 DF
IP: ip_ttl=128, ip_sum=42c2, ip_p = 6 (TCP)
TCP: <source port=4743, destination port=23(telnet) >
TCP: th_seq=d3b63f19, th_ack=eb301b27
TCP: th_off=5, flags<ACK>
TCP: th_win=16363, th_sum=2f08, th_urp=0

...(lines omitted)...

```

++++++ END OF REPORT ++++++

processed 154 packets
displayed 100 packets
Summary of RPC CALL packets

Example 30-11 shows the initiation of a TCP connection between two hosts (from SRC to DST) on an Ethernet network.

Example 30-11 TCP initiation

```
ETH: ==== ( 74 bytes received on interface en0 )==== 12:22:05.191609117
ETH: [ 00:04:ac:ec:07:98 -> 00:04:ac:ec:08:d0 ] type 800 (IP)
IP: < SRC = 1.40.35.98 >
IP: < DST = 1.40.35.102 >
IP: ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=41626, ip_off=0
IP: ip_ttl=60, ip_sum=9309, ip_p = 6 (TCP)
TCP: <source port=34308, destination port=2049(shilp) >
TCP: th_seq=f47ddc71, th_ack=0
TCP: th_off=10, flags<SYN>
TCP: th_win=65535, th_sum=4b0a, th_urp=0
TCP: mss 1460
TCP: nop
TCP: wscale 1
TCP: nop
...(lines omitted)...
```

Note the request for message segment size of 1460. Example 30-12 is the reply to the initiation request (note the SYN and ACK in the flags field).

Example 30-12 TCP initiation reply

```
ETH: ==== ( 74 bytes transmitted on interface en0 )==== 12:22:05.191741778
ETH: [ 00:04:ac:ec:08:d0 -> 00:04:ac:ec:07:98 ] type 800 (IP)
IP: < SRC = 1.40.35.102 >
IP: < DST = 1.40.35.98 >
IP: ip_v=4, ip_hl=20, ip_tos=0, ip_len=60, ip_id=51148, ip_off=0
IP: ip_ttl=60, ip_sum=6dd7, ip_p = 6 (TCP)
TCP: <source port=2049(shilp), destination port=34308 >
TCP: th_seq=b29207af, th_ack=f47ddc72
TCP: th_off=10, flags<SYN | ACK>
TCP: th_win=59368, th_sum=5530, th_urp=0
TCP: mss 1460
TCP: nop
TCP: wscale 0
TCP: nop
...(lines omitted)...
```

For a more information about the protocol headers, refer to “Packet header formats” on page 535 and the *TCP/IP Protocols* chapter in the *AIX 5L Version 5.2 System Management Guide: Communications and Networks*.

Below we give a brief description of the **ipreport** report shown in the example above:

First line	A network frame summary line.
Second line	Contains the destination and source Media Access Control (MAC) addresses and the frame type number and protocol in the following format: ##:##:##:##:##:## -> ##:##:##:##:##:##] type ### (X)
SRC	The source (sender) of the packet.
DST	The destination (receiver) for the packet.
source port	The port that the sending service used to transmit the application layer data from.
destination port	The port that the receiving service uses to receive data.
ip_	The ip_ fields correspond to the IP header fields (see “IP V4 (RFC 791) packet header” on page 536). For example, the ip_len field is the size including all packet information, the ip_hl is the IP header size, the ip_v is the IP version (4 or 6), and ip_p is the transport layer protocol (such as 1 for ICMP, 6 for TCP, and 17 for UDP). If the field is “unknown internet protocol” check the /etc/protocols file and remove the comment (#) for the protocol line with the matching ip_p number.
th_seq	th_seq will be a large number for the first packet. After the three-way handshake has established a TCP connection, th_seq will equal the th_ack from the last packet from the other system. A th_ack of zero indicates that this is the first packet of the sequence. After this it will contain the th_seq from the last system plus the number of data bytes in the last packet.
th_win	Indicates the number of bytes of receive buffer space available from the message originator. If the th_win field is zero (0), it means that the other side is not accepting any more data for the moment.

flags

The flags field contains the control bits to identify the purpose of the packet. A brief explanation of some flags and combinations:

- SYN** Synchronize the sequence numbers, packet from the first part in the connection, and the first part of initial connection setup three-way handshake.
- ACK** Acknowledgement of receipt, and also the third part of initial connection setup three-way handshake from the first part in the connection.
- FIN** Indicates that the sender has reached the end of its byte stream.
- PUSH** Segment requests a PUSH (to deliver data).
- URG** Urgent pointer field is valid.
- RST** Resets the connection.
- SYNACK** Synchronize the sequence numbers and acknowledge from the second part in the connection, and the second part of initial connection setup three-way handshake from the second part in the connection.
- FINIACK** Acknowledge receipt, and the sender indicates that it is finished with this connection. Either part can indicate the completion of a connection. However, data may still be sent.
- PUSHIACK** Acknowledge receipt and PUSH data to application.

30.7 Examples for ipfilter

The **ipfilter** command summarizes TCP/IP traffic flow by using the ASCII output created by the **ipreport** command. Because the **ipreport** command uses input from either **iptrace** or **tcpdump**, these commands must first be used to create a binary trace of the network communication. In the following examples we show how to create the input file for **ipfilter** and then show the different reports that **ipfilter** can produce.

30.7.1 Tracing TCP/IP traffic

To trace TCP/IP traffic and use this as input to the **ipfilter** command, we can use either the **iptrace** command, as shown in Example 30-13, or the **tcpdump** command as shown in Example 30-14.

Example 30-13 Start and stop iptrace for ipreport and ipfilter

```
# startsrc -s iptrace -a "-P tcp $PWD/iptrace.tcp"
0513-059 The iptrace Subsystem has been started. Subsystem PID is 19602.
# stopsrc -s iptrace
0513-044 The iptrace Subsystem was requested to stop.
# lssrc -s iptrace
Subsystem      Group      PID      Status
iptrace        tcpip     inoperative
```

The first command line starts the trace using the SRC to control the execution. This makes it easier to stop the trace with the **stopsrc** command instead of using **ps** and **kill**. In the example above we let **iptrace** create a file in the current directory (**\$PWD**).

The **-w** flags to the **tcpdump** command specifies that it should write raw packets to *stdout* instead of parsing and printing them out. By specifying **-** as the input file to **ipreport**, it will read from *stdin*. The **-rs** flags tells **ipreport** to start lines with protocol indicator strings and to be aware of RPC packets.

Example 30-14 Using ipreport with tcpdump

```
# tcpdump -vvSNs4096 -c 512 -w - tcp | ipreport -rsT - >$PWD/ipreport.tcp
tcpdump: listening on tr0
1261 packets received by filter
53 packets dropped by kernel
```

To create the **ipreport** file needed by **ipfilter**, run the **ipreport** command as follows (still using the directory defined in the **PWD** environment variable as the path to the input and output files for the commands):

```
ipreport -s $PWD/iptrace.out >$PWD/ipreport.tcp
```

Example 30-15 is a short extract from the **ipreport.tcp** file that we created with the **ipreport** command for the **iptrace** created binary network trace output as shown above in Example 30-13.

Example 30-15 ipreport output for ipfilter

IPTRACE version: 2.0

```
TOK: ==( ( 62 bytes received on interface tr0 )== 12:51:53.809120113
TOK: 802.5 packet
TOK: 802.5 MAC header:
```

```
TOK: access control field = 10, frame control field = 40
TOK: [ src = 00:60:94:87:0a:87, dst = 00:60:94:8a:07:5b]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP: < SRC = 1.3.1.114 > (3b-043)
IP: < DST = 1.3.1.164 > (wlmhost)
IP: ip_v=4, ip_hl=20, ip_tos=0, ip_len=40, ip_id=41691, ip_off=0 DF
IP: ip_ttl=128, ip_sum=42d9, ip_p = 6 (TCP)
TCP: <source port=4743, destination port=23(telnet) >
TCP: th_seq=d3b63f0b, th_ack=eb301af8
TCP: th_off=5, flags<ACK>
TCP: th_win=16410, th_sum=2f16, th_urp=0
```

...(lines omitted)...

As you can see, the lines are prefixed with a keyword for the protocol header type the output of each line belongs to. TOK specifies the token-ring frame type from the *network interface layer*, IP is the IP protocol from the *network layer*, and TCP is the Transmission Control Protocol from the *transport layer*. (See Example 30-15 on page 583.) The only difference between the **ipreport** output for **iptrace** and **tcpdump** is the first line in the report file indicating the source of the network trace data:

- ▶ **iptrace** has the header of IPTRACE version: 2.0
- ▶ **tcpdump** has the header of TCPDUMP

To generate a summarized report for all protocols using the **ipreport** output, run the **ipfilter** command:

```
ipfilter /tmp/ipreport.out
```

To limit the report to NFS (RPC) only, use the **-n** flag:

```
ipfilter -n /tmp/ipreport.out
```

30.7.2 NFS tracing

Example 30-16 shows a sample output taken from **nfs.rpt** generated by **ipfilter -n**. It can be used to get an overview of the NFS traffic.

Example 30-16 **nfs.rpt**

```

                                NFS REPORT

                                Elapsed Milliseconds Cut Off= 0
                                -----CALL-----
Transaction ID Request  Status  Packet  Send Time (secs) Size  Packet  Send Time (secs) Size  Elapsed msec
                                -----REPLY-----
-----
224607473 NFSPROC3_GETATTR SUCCESS  0        53.233199  174      0        53.234672  178      0.000
224607474 NFSPROC3_GETATTR SUCCESS  0        53.235347  174      0        53.236720  178      0.000
224607475 NFSPROC3_GETATTR SUCCESS  0        53.253619  174      0        53.254668  178      0.000

```

```

224607476 NFSPROC3_GETATTR SUCCESS 0 53.255145 174 0 53.256369 178 0.000
224607477 NFSPROC3_GETATTR SUCCESS 0 0.387394 174 0 0.388726 178 0.000
224607478 NFSPROC3_GETATTR SUCCESS 0 24.852327 174 0 24.853634 178 0.000
224607479 NFSPROC3_GETATTR SUCCESS 0 24.854309 174 0 24.855359 178 0.000
224607480 NFSPROC3_GETATTR SUCCESS 0 24.855896 174 0 24.857220 178 0.000
224607481 NFSPROC3_GETATTR SUCCESS 0 24.857594 174 0 24.858897 178 0.000
...(lines omitted)...

```

The *nfs.rpt* file (and the other **ipfilter** generated files) can either be analyzed by browsing the file or by extracting selected parts of the information. The following are a few examples of how to extract some interesting parts from the *nfs.rpt* file. In all the examples below we exclude the header (lines 1 to 7) and all records for UNKNOWN_PROC in the Request field.

The following sample commands extract **ipfilter** results for specific conditions:

- ▶ Extracts all records that have more than 0.0 in the Elapsed msec field:

```
awk '$10>0.0 && $2!~/UNKNOWN_PROC/ && NR>7' nfs.rpt
```

- ▶ Extracts all records that have a Size larger than 512 bytes:

```
awk '$9>512 && $2!~/UNKNOWN_PROC/ && NR>7' nfs.rpt
```

- ▶ Extracts all records that have Status of SUCCESS:

```
awk '$3!~/SUCCESS/ && $2!~/UNKNOWN_PROC/ && NR>7' nfs.rpt
```

- ▶ Extracts all records that have a Request type matching _READ:

```
awk '$2~/_READ/ && NR>7' nfs.rpt
```

30.7.3 TCP tracing

Example 30-17 shows a sample output taken from *ipreport.tcp* generated by **ipfilter -s t**. It gives a good overview of the TCP packet flow.

Example 30-17 ipfilter.tcp

```

Operation Headers: TCP

```

pkt.	Time	Source	Dest.	Length	Seq #	Ack #	Ports		Net_Interface	Operation
							Source	Destination		
22	18:06:44.509308	1.3.1.114	1.3.1.164	41	d3b5fa52,	eb1263e6	4743,	23(telnet)	tr0	TCP ACK PUSH
23	18:06:44.509938	1.3.1.164	1.3.1.114	41	eb1263e6,	d3b5fa53	23(telnet),	4743	tr0	TCP ACK PUSH
26	18:06:44.668588	1.3.1.114	1.3.1.164	41	d3b5fa53,	eb1263e7	4743,	23(telnet)	tr0	TCP ACK PUSH
27	18:06:44.669122	1.3.1.164	1.3.1.114	41	eb1263e7,	d3b5fa54	23(telnet),	4743	tr0	TCP ACK PUSH
30	18:06:44.782295	1.3.1.114	1.3.1.164	40	d3b5fa54,	eb1263e8	4743,	23(telnet)	tr0	TCP ACK
33	18:06:44.827938	1.3.1.114	1.3.1.164	41	d3b5fa54,	eb1263e8	4743,	23(telnet)	tr0	TCP ACK PUSH
34	18:06:44.828334	1.3.1.164	1.3.1.114	41	eb1263e8,	d3b5fa55	23(telnet),	4743	tr0	TCP ACK PUSH
35	18:06:44.913575	1.3.1.114	1.3.1.164	41	d3b5fa55,	eb1263e9	4743,	23(telnet)	tr0	TCP ACK PUSH

...(lines omitted)...

30.7.4 UDP tracing

Example 30-18 shows a sample output taken from `ipreport.udp` generated by `ipfilter -s u`. It shows the UDP packet flow.

Example 30-18 `ipreport.udp`

Operation Headers: UDP

pkt.	Time	Source	Dest.	Length	Seq #	Ack #	Ports		Net_Interface	Operation
							Source	Destination		
11	18:06:42.021652	1.3.1.114	221.55.150.208	178,			1346,	1345	tr0	UDP
59	18:06:46.253304	1.3.1.144	1.3.1.255	265,			138(netbios-	138(netbios-	tr0	UDP
62	18:06:46.580319	1.3.1.103	221.55.150.208	178,			1346,	1345	tr0	UDP
67	18:06:46.749541	1.3.1.164	1.3.1.2	73,			37310,	53(domain)	tr0	UDP
68	18:06:46.750645	1.3.1.2	1.3.1.164	149,			53(domain),	37310	tr0	UDP
69	18:06:46.750974	1.3.1.164	1.3.1.2	53,			37311,	53(domain)	tr0	UDP
70	18:06:46.751848	1.3.1.2	1.3.1.164	129,			53(domain),	37311	tr0	UDP
71	18:06:46.752215	1.3.1.164	1.3.1.2	73,			37312,	53(domain)	tr0	UDP
72	18:06:46.753022	1.3.1.2	1.3.1.164	149,			53(domain),	37312	tr0	UDP
73	18:06:46.753331	1.3.1.164	1.3.1.2	53,			37313,	53(domain)	tr0	UDP
74	18:06:46.754269	1.3.1.2	1.3.1.164	129,			53(domain),	37313	tr0	UDP
78	18:06:46.789004	1.3.1.164	1.3.1.2	68,			37314,	53(domain)	tr0	UDP
79	18:06:46.789946	1.3.1.2	1.3.1.164	163,			53(domain),	37314	tr0	UDP
...(lines omitted)...										

30.7.5 ICMP tracing

Example 30-19 shows a sample output taken from `ipreport.icmp` generated by `ipfilter -s c`. It shows the ICMP packet flow.

Example 30-19 `ipfilter.icmp`

Operation Headers: ICMP

pkt.	Time	Source	Dest.	Length	Seq #	Ack #	Ports		Net_Interface	Operation
							Source	Destination		
1	18:06:40.734626	1.31.7.84	1.3.1.164	84,			0	0	tr0	ICMP
2	18:06:40.734790	1.3.1.164	1.31.7.84	84,			0	0	tr0	ICMP
3	18:06:40.818499	1.31.7.76	1.3.1.164	84,			0	0	tr0	ICMP
4	18:06:40.818664	1.3.1.164	1.31.7.76	84,			0	0	tr0	ICMP
7	18:06:41.733939	1.31.7.84	1.3.1.164	84,			0	0	tr0	ICMP
8	18:06:41.734051	1.3.1.164	1.31.7.84	84,			0	0	tr0	ICMP
9	18:06:41.817298	1.31.7.76	1.3.1.164	84,			0	0	tr0	ICMP
10	18:06:41.817430	1.3.1.164	1.31.7.76	84,			0	0	tr0	ICMP
...(lines omitted)...										

30.7.6 IPX tracing

Example 30-20 on page 587 shows a sample output taken from `ipreport.ipx` generated by `ipfilter -s x`. It shows the IPX packet flow.

Example 30-20 ipfilter.ipx

Operation Headers: IPX

pkt.	Time	Source	Dest.	Length	Seq #	Ack #	Ports		
							Source	Destination	Net_Interface
123	18:06:50.182097	0	0	58			0	0	tr0 IPX
138	18:06:51.178240	0	0	58			0	0	tr0 IPX
157	18:06:52.178317	0	0	58			0	0	tr0 IPX
165	18:06:53.178040	0	0	58			0	0	tr0 IPX
...(lines omitted)...									

30.7.7 ALL protocol tracing

Example 30-21 shows a sample output taken from `ipreport.all` generated by `ipfilter` without parameters. It summarizes all protocols.

Example 30-21 ipfilter.all

Operation Headers: ICMP IPX NFS TCP UDP

pkt.	Time	Source	Dest.	Length	Seq #	Ack #	Ports			Operation
							Source	Destination	Net_Interface	
1	18:06:40.734626	1.31.7.84	1.3.1.164	84,			0	0	tr0 ICMP	
2	18:06:40.734790	1.3.1.164	1.31.7.84	84,			0	0	tr0 ICMP	
3	18:06:40.818499	1.31.7.76	1.3.1.164	84,			0	0	tr0 ICMP	
4	18:06:40.818664	1.3.1.164	1.31.7.76	84,			0	0	tr0 ICMP	
5	18:06:40.832805	1.3.1.1	224.0.0.5	68,			0	0	tr0	
6	18:06:41.409446	[1.3.1.24]	[1.3.1.188]	52			0	0	tr0 ARP	
...(lines omitted)...										
13258	18:07:07.581747	1.3.1.164	1.3.1.164	1492,	b9d0c9e6,	daad2ac1	35648,	20(ftp-data)	1o0 TCP ACK	
13259	18:07:07.581757	1.3.1.164	1.3.1.164	1492,	b9d0cf92,	daad2ac1	35648,	20(ftp-data)	1o0 TCP ACK	
13260	18:07:07.581820	1.3.1.164	1.3.1.164	1492,	b9d0d53e,	daad2ac1	35648,	20(ftp-data)	1o0 TCP ACK	
...(lines omitted)...										
13381	18:07:14.592564	1.3.1.114	1.3.1.164	41,	d3b5fac8,	eb126778	4743,	23(telnet)	tr0 TCP ACK PUSH	
13382	18:07:14.593005	1.3.1.164	1.3.1.114	41,	eb126778,	d3b5fac9	23(telnet),	4743	tr0 TCP ACK PUSH	
13383	18:07:14.638730	1.3.1.114	1.3.1.164	42,	d3b5fac9,	eb126779	4743,	23(telnet)	tr0 TCP ACK PUSH	
13384	18:07:14.639378	1.3.1.164	1.3.1.114	42,	eb126779,	d3b5facb	23(telnet),	4743	tr0 TCP ACK PUSH	
0	18:07:14.639378	0	0	0	0		0	0	ACK	
...(lines omitted)...										

30.8 tcpdump

The syntax of the `tcpdump` command is:

```
tcpdump [-defIlNOpqStvx] [-c Count] [-i Interface] [-F File] [-r File]
        [-w File] [-s Snaplen] [Expression]
```

Flags

-c Count

Exits after receiving Count packets.

- d** Dumps the compiled packet-matching code to standard output, then stops.
- e** Prints the link-level header on each dump line. If the **-e** flag is specified, the link level header is printed out. On Ethernet and token-ring, the source and destination addresses, protocol, and packet length are printed.
- f** Prints foreign internet addresses numerically rather than symbolically.
- F File** Uses File as input for the filter expression. The **-F** flag ignores any additional expression given on the command line.
- i Interface** Listens on Interface. If unspecified, the **tcpdump** command searches the system interface list for the lowest numbered and configured interface that is up. This search excludes loopback interfaces. For supported interfaces refer to 30.8.1, “Information about measurement and sampling” on page 589.
- I** (Capital i) Specifies immediate packet capture mode. The **-I** flag does not wait for the buffer to fill up.
- l** (Lowercase L) Buffers the standard out (*stdout*) line. This flag is useful if you want to see the data while capturing it.
- n** Omits conversion of addresses to names.
- N** Omits printing domain name qualification of host names. For example, the **-N** flag prints dude instead of dude.itso.ibm.com.
- O** Omits running the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.
- p** Specifies that the interface not run in promiscuous mode.
- q** Quiets output. The **-q** flag prints less protocol information so output lines are shorter.
- r File** Reads packets from File (which is created with the **-w** option). Standard input is used if File is "-".
- s Snaplen** Captures Snaplen bytes of data from each packet rather than the default of 80. Eighty bytes is adequate for IP, ICMP, TCP, and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with [*proto*], where *proto* is

	the name of the protocol level at which the truncation has occurred.
-S	Prints absolute rather than relative TCP sequence numbers.
-t	Omits the printing of a time stamp on each dump line.
-tt	Prints an unformatted time stamp on each dump line.
-v	Specifies slightly more verbose output. For example, the time to live and the type of service information in an IP packet is printed.
-w File	Writes the raw packets to File rather than parsing and printing them out. They can later be printed with the -r flag. Standard output is used if File is "-".
-x	Prints each packet (minus its link level header) in hex. The smaller of the entire packet or Snaplen bytes will be printed.

Parameters

Interface	Network interface to listen for packets on.
File	The File parameter specifies the name of the file to use as input or output depending on optional flag specified.
Snaplen	Specifies the number of bytes of data from each packet.
Expression	Consists of one or more primitives.

30.8.1 Information about measurement and sampling

The output of the **tcpdump** command is protocol-dependent. The different protocols that can be monitored are:

- ▶ Ethernet frames (ether)
- ▶ IP (ip)
- ▶ ARP and RARP (arp and rarp)
- ▶ TCP (tcp)
- ▶ UDP (udp)

Note: These are the protocol formats that tcpdump can interpret and analyze. Because tcpdump can trace the IP protocol, it can trace other protocol types as well, but it is left to the user to interpret the packet header. In some cases tcpdump can also interpret the application-level packet headers, such as for Domain Name Services (DNS).

The **tcpdump** command will only monitor one interface at a time (rather than several, as the **iptrace** command can). Only Ethernet V2 (en) and Ethernet 802.3 (et), token-ring (tr), FDDI (fddi), ATM (at), and loopback (lo) interfaces are supported to be monitored. For other interfaces the **iptrace** command can be used (30.2, “iptrace” on page 569).

Access to monitor network traffic is controlled by the permissions on /dev/bpf# special files because **tcpdump** uses the Berkeley Packet Filter (BPF) packet capture library. (See “Packet Capture Library Subroutines” in *AIX 5L Version 5.2 Technical Reference: Communications, Volume 2* for further information about the BPF.) You can also check at <http://www.tcpdump.org>; however, the publicly available **tcpdump** command differs from the supported **tcpdump** command supplied with AIX.

You can specify the direction of the communication that is monitored, such as *simplex* in one or the other direction or *duplex* for both directions. It is also possible to monitor *broadcast* and *multicast* packets.

By default, all output lines are preceded by a time stamp. The time stamp is the current clock time in the form of hh:mm:ss.frac.

The time stamps are as accurate as the kernels clock. The time stamp reflects the time the kernel first saw the packet. No attempt is made to account for the time lag between when the network (interface) device driver removed the packet from the wire and when the kernel serviced the new packet interrupt. Time Stamping can be turned off by specifying the -t flag.

Expressions

The Expression parameter consists of one or more primitives. Primitives usually consist of an ID (name or number) preceded by one or more qualifiers. There are three types of qualifier:

type	Specifies the kind of device the ID name or number refers to. Possible types are host, net, and port.
dir	Specifies a particular transfer direction to or from ID. Possible directions are src, dst, src or dst, and src and dst.
proto	Restricts the match to a particular protocol. Possible proto qualifiers are: ether, ip, arp, rarp, tcp, and udp. If there is no proto qualifier, all protocols consistent with the type are assumed.

In addition, there are some special primitive keywords that do not follow the pattern: broadcast, multicast, less, greater, and arithmetic expressions. Note that broadcast and multicast are only supported for the ether protocol type.

Device types and transfer direction primitives

These primitives are allowed:

dst host Host	True if the value of the IP (Internet Protocol) destination field of the packet is the same as the value of the Host variable, which may be either an address or a name.
src host Host	True if the value of the IP source field of the packet is the same as the value of the Host variable.
host Host	True if the value of either the IP source or destination of the packet is the same as the value of the Host variable. Any of the above host expressions can be preceded with the keywords ip, arp, or rarp, such as ip host Host.
dst net Net	True if the value of the IP destination address of the packet has a network number of Net.
src net Net	True if the value of the IP source address of the packet has a network number of Net.
net Net	True if the value of either the IP source or destination address of the packet has a network number of Net.
dst port Port	True if the packet is TCP/IP or IP/UDP (Internet Protocol/User Datagram Protocol) and has a destination port value of Port. The port can be a number or a name used in /etc/services. If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked.
src port Port	True if the value of the Port variable is the same as the value of the source port.
port Port	True if the value of either the source or the destination port of the packet is Port. Any of the above port expressions can be preceded by the keywords tcp or udp as in: tcp src port port
less Length	True if the packet has a length less than or equal to the Length variable.
greater Length	True if the packet has a length greater than or equal to the Length variable.
ip proto Protocol	True if the packet is an IP packet of protocol type Protocol. Protocol can be a number or one of the names icmp, udp, or tcp. The identifiers tcp, udp, and icmp are considered keywords and must be escaped using \ (backslash), such as ip proto \tcp.

ip broadcast	True if the packet is an IP broadcast packet. It checks for the all-zeroes and all-ones broadcast conventions, and looks up the local subnet mask.
ip multicast	True if the packet is an IP multicast packet.
proto Protocol	True if the packet is of type Protocol. Protocol can be a number or a name like ip, arp, or rarp. These identifiers are also keywords and must be escaped using a \ (backslash), such as proto \tcp.

Abbreviations

The following protocol abbreviations can be used:

ip, arp, rarp	Abbreviations for proto directives
tcp, udp, icmp	Abbreviations for ip proto directives

Arithmetic expressions

The following relational expressions can be used:

expr relop expr The relop is one of the following relational operators:

- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- = Equal to
- != Not equal to (exclamation point and equal sign)

The expr is an arithmetic expression composed of integer constants (in standard C syntax) or binary operators, such as:

- + Plus sign
- Minus sign
- * Multiplication sign (asterisk)
- / Division sign (slash)
- & Logical AND sign (ampersand)
- | Logical OR sign (pipe)

The following operators can also be used in expressions:

- len** Length operator
- []** Packet data accessors

Combining primitives

More complex filter expressions are built up by using the words `and`, `or`, and `not` (!) to combine primitives:

not / !	Negation
and	Concatenation
or	Alternation

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. If an identifier is given without a keyword, the most recent keyword is assumed. Primitives may be combined, but must be escaped so that they are not interpreted by the shell:

```
\( a = b \  
"a = b"  
'a = b'
```

Only the last example above will specify to the shell that the string between the citation marks (') should not be parsed at all. The following show how to combine primitives and use expressions to access data (in the 14th byte in the tcp header, bits 7 and 8) in a packet:

```
'(tcp[13] & 6 != 0) or (tcp[13] & 7 != 0)'
```

Accessing data inside a packet

To access data inside the packet, use the following syntax:

```
proto [ expr : size ]
```

`proto` is one of the keywords `ip`, `arp`, `rarp`, `tcp`, or `icmp`, and indicates the protocol layer for the index operation. The byte offset relative to the indicated protocol layer is given by `expr`. The indicator `size` is optional and indicates the number of bytes in the field of interest. It can be either one, two, or four, and defaults to one byte. The length operator, indicated by the keyword `len`, gives the length of the packet.

Note that addressing starts with zero (0) so, for example, byte 14 in a TCP packet header would be written as `tcp[13]`. After the byte position the bit offset can be specified by using `& #`, where `#` represents bit zero (0) to seven (7). When using bit expressions the comparison is binary (zero or one). The following reminder shows the bits and their corresponding value if the bit is on (1):

```
bit   : 0 1 2 3 4 5 6 7  
value : 1 2 4 8 16 32 64 128
```

You can use the `bc` command to convert between different bases, as shown in Example 30-22 on page 594.

Example 30-22 Using the bc command to perform base conversion of numbers

```
# print "obase=2\n90"|bc
1011001
# print "obase=16\n90"|bc
5A
```

In the first example above the value 90 is converted to binary format and in the second example the value 90 is converted to hexadecimal format.

30.9 Examples for tcpdump

Because network tracing can produce large amounts of data, it is important to limit the network trace, either by *scope* (what to trace) or *amount* (how much to trace). The **tcpdump** command offers many options to reduce the scope of the network trace, unlike the **iptrace** command. (See 30.2, “iptrace” on page 569.) The **tcpdump** can also display readable reports as well as saving binary output, for formatting later, from the network trace data.

A good way to use **tcpdump** is to save the network trace to a file with the **-w** flag and then analyze the trace by using different filtering options together with the **-r** flag. The following example show how to run a basic **tcpdump** network trace, saving the output in a file with the **-w** flag (on a Ethernet network interface):

```
tcpdump -w /tmp/tcpdump.en0 -i en0
```

To limit the number of traced packets, use the **-c** flag and specify the number, such as in the following example that traces the first 128 packets (on a token-ring network interface):

```
tcpdump -c 128 -w /tmp/tcpdump.tr0 -i tr0
```

To read the file produced by a previous **tcpdump** command above, use the **-r** flag as shown below:

```
tcpdump -Snr /tmp/tcpdump.en0
```

Note that when reading the **tcpdump** trace we usually want to include both the time stamp and the absolute sequence numbers, and use IP addresses and not host and domain names. Note that **tcpdump** wraps relative sequence numbers (and sometimes it shows the absolute anyway), which can make the analysis more difficult. By displaying only absolute sequence numbers, this problem will not occur.

Using tcpdump with ipreport

The `-w` flag for `tcpdump` specifies that it should write raw packets to `stdout` instead of parsing and printing them out. By specifying `-` as the input file to `ipreport`, it will read from `stdin`. The `-rs` flag tells `ipreport` to start lines with protocol indicator strings and to be aware of RPC packets. Example 30-23 shows how this can be done.

Example 30-23 Using ipreport with tcpdump

```
# tcpdump -w - | ipreport -rsT - | more
TCPDUMP

TOK: ==== ( 80 bytes on interface token-ring )==== 16:42:43.327359881
TOK: 802.5 packet
TOK: 802.5 MAC header:
TOK: access control field = 10, frame control field = 40
TOK: [ src = 08:00:5a:fe:21:06, dst = 00:20:35:72:98:31]
TOK: 802.2 LLC header:
TOK: dsap aa, ssap aa, ctrl 3, proto 0:0:0, type 800 (IP)
IP:   < SRC =      13.7.140 > (sox5.itso.ibm.com)
IP:   < DST =      1.3.1.41 >
IP:   ip_v=4, ip_hl=20, ip_tos=0, ip_len=1500, ip_id=23840, ip_off=0
IP:   ip_ttl=57, ip_sum=442, ip_p = 6 (TCP)
IP:   truncated-ip, 1442 bytes missing
IP:   00000000 043804fa a8fb14da 0937db32 50107d78 |.8.....7.2P.}x|
IP:   00000010 33330000 863fcc52 996d64f2 577d2c2c |33...?.R.md.W},,|
IP:   00000020 c5f7c26a leed |...j..|
...(lines omitted)...
```

Monitoring TCP

For many performance-related TCP/IP communication cases the protocol to tune is TCP. One way of quickly gathering information about how the TCP protocol flow is performing on the local network is to limit the filtering scope by only monitoring initiation and termination of TCP connections.

In Example 30-24 only packets with the TCP header field are monitored. For clarity we reduce the output length by excluding the time stamp information.

Example 30-24 Using tcpdump with TCP to monitor start/stop packets

```
# tcpdump -c 6 -vni s1492 -i tr0 "tcp[13] & 7 != 0"
tcpdump: listening on tr0
20:44:55.884956867 1.3.1.47.3111 > 1.99.41.117.1352: S 2739718210:2739718210(0) win 64240 <mss 1361,nop,nop,sackOK> (DF)] (ttl 128, id 31407)
20:44:55.919483478 1.99.41.117.1352 > 1.3.1.47.3111: S 3917974983:3917974983(0) ack 2739718211 win 65535 <mss 1448>] (ttl 52, id 56924)
20:44:56.664665475 1.3.1.47.3106 > 1.99.140.15.1352: F 2738058183:2738058183(0) ack 819381945 win 64062 (DF)] (ttl 128, id 31418)
```

```
20:44:56.664858098 1.3.1.47.3107 > 1.14.3.93.1080: F 2738572850:2738572850(0) ack 4011668374
win 63132 (DF)] (ttl 128, id 31419)
20:44:56.665211657 1.3.1.47.3109 > 1.165.223.3.1352: F 2739075962:2739075962(0) ack 2156758289
win 63146 (DF)] (ttl 128, id 31420)
20:44:56.665288620 1.3.1.47.3110 > 1.3.1.7.1352: F 2739600920:2739600920(0) ack 16883503 win
63192 (DF)] (ttl 128, id 31421)
51 packets received by filter
0 packets dropped by kernel
```

The output above shows connection setup packets (marked lines with S in the flags field) between host 1.3.1.47 and 1.99.41.117. Note that these are only the first two steps of the TCP three-way handshake. (Refer to “Opening a TCP connection” on page 597.)

The general format of a TCP protocol line is²:

```
SRC > DST: flags data-seqno ack win urg options
```

The following are the explanation of the fields:

SRC	Indicates the source (host) address and port. This field is always specified.
DST	Indicates the destination address and port. This field is always specified.
flags	Specifies some combination of the flags S (SYN), F (FIN), P (PUSH), and R (RST), or a single . (period) to indicate no flags. The flags field is always specified.
data-seqno	Describes the portion of sequence space covered by the data in this packet. In ACK packets the data-seqno part can be split up into FS:LS(NSN) FS First sequence number to acknowledge LS Last sequence number to acknowledge NSN Next sequence number to use
ack	Specifies (by acknowledgement) the sequence number of the next data packet expected from the other direction on this connection.
win	Specifies the number of bytes of receive buffer space available from the other direction on this connection.
urg	Indicates there is urgent data in the packet.
options	Specifies TCP options enclosed in angle brackets.

² Fields, except SRC, DST, and flags, depend on the contents of the packet's TCP protocol header and are output only if appropriate.

Monitoring all TCP traffic

Example 30-25 shows how to use `tcpdump` to monitor all TCP traffic. In the example, `tcpdump` will only report 10 packets (`-c 10`), read 1492³ bytes from each packet, exclude the time stamp (`-t`), reporting will be interactive (`-l`) for TCP (`tcp`) protocol only, and will omit the domain name part of host names (`-N`):

Example 30-25 Using tcpdump to monitor TCP

```
# tcpdump -c 10 -tNiS 1492 tcp
tcpdump: listening on tr0
3b-043.2423 > wlmhost.telnet: . ack 2684017960 win 16281 (DF)]
wlmhost.telnet > 3b-043.2423: P 2684017960:2684018015(55) ack 897371828 win 17424]
wlmhost.telnet > 3b-043.2423: P 2684017960:2684018015(55) ack 897371828 win 17424]
3b-043.2423 > wlmhost.telnet: . ack 55 win 16226 (DF)]
wlmhost.telnet > 3b-043.2423: P 2684018015:2684018247(232) ack 897371828 win 17424]
wlmhost.telnet > 3b-043.2423: P 2684018015:2684018247(232) ack 897371828 win 17424]
3b-043.2423 > wlmhost.telnet: . ack 232 win 15994 (DF)]
wlmhost.telnet > 3b-043.2423: P 2684018247:2684018473(226) ack 897371828 win 17424]
wlmhost.telnet > 3b-043.2423: P 232:458(226) ack 1 win 17424]
3b-043.2423 > wlmhost.telnet: . ack 2684018473 win 17424 (DF)]
34 packets received by filter
0 packets dropped by kernel
```

In the example above there are four packets with the *IP does not fragment* flag set. This is marked with a trailing (DF). This flag is set in the D bit field as shown in the schematic header layout for the IP V4 header. (See “IP V4 (RFC 791) packet header” on page 536.)

Opening a TCP connection

When a TCP connection is opened a *three-way handshake* is performed, as shown in Example 30-26.

Example 30-26 Using tcpdump to monitor TCP

```
# tcpdump -tNiS 1492 "tcp port 23 and host dude.itso.ibm.com"
tcpdump: listening on tr0
...(lines omitted)...
bolshoi.32796 > wlmhost.telnet: S 563275966:563275966(0) win 16384 <mss 1452> (DF) [tos 0x10]
wlmhost.telnet > bolshoi.32796: S 3147194261:3147194261(0) ack 563275967 win 17424 <mss 1452> (DF)]
bolshoi.32796 > wlmhost.telnet: . ack 3147194262 win 17424 (DF) [tos 0x10]
...(lines omitted)...
718 packets received by filter
0 packets dropped by kernel
```

³ Because the MTU size for this particular interface is set to 1492, no IP frames larger than 1492 bytes will be used by our host (for Ethernet 10/100 this will be 1500). However, there may be frames on the token-ring that are larger than the local MTU size sent by other systems with different sizes. Use commands such as `netstat` or `ifconfig` to determine the current MTU size (the `lsattr` command shows the setting used when the network interface device driver was loaded).

Refer to “Monitoring TCP” on page 595 for an explanation of the output format in the previous example.

The three-okway handshake can be described as follows (see Figure 30-1):

1. The initiator (bolshoi) sends a SYN packet to the party (wlmhost) that it wants to connect to. The initial sequence number is 563275966 in the example above (line 1 from bolshoi to wlmhost).
2. The receiving party (wlmhost) responds with its own SYN packet containing its initial sequence number (line 2 from wlmhost to bolshoi). This packet also contains an ACK flag to acknowledge the initiator’s SYN sequence number by incrementing the current sequence number by one, which would be 563275967 (563275966 +1) in the example above.
3. The initiator acknowledges this SYN from the second party by sending a ACK packet with no flag (.) indicator (line 3 from bolshoi to wlmhost) by incrementing the current sequence number by one, which would be 3147194262 (3147194261 +1) in the example above.

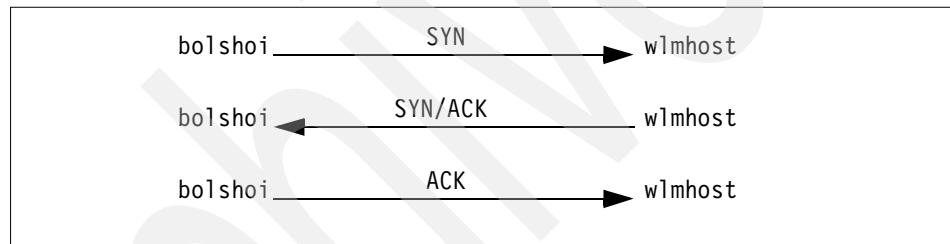


Figure 30-1 Schematic flow during TCP open

Closing a TCP connection

To end a TCP connection, either of the two communication ends can send an end of transmission segment (containing the FIN flag) when it has finished transmitting data, as shown in Example 30-27.

Example 30-27 Using tcpdump to monitor TCP⁴

```
# tcpdump -tN1c 1492 "tcp port 23"
tcpdump: listening on tr0
...(lines omitted)...
wlmhost.telnet > bolshoi.32785: F 1949791116:1949791116(0) ack 3641849741 win 17424 (DF)]
bolshoi.32785 > wlmhost.telnet: . ack 1949791117 win 17424 (DF) [tos 0x10]
bolshoi.32785 > wlmhost.telnet: F 3641849741:3641849741(0) ack 1949791117 win 17424 (DF) [tos 0x10]
wlmhost.telnet > bolshoi.32785: . ack 3641849742 win 17424 (DF)]
...(lines omitted)...
```

⁴ Note that the monitoring was interrupted by using the Ctrl-C key sequence (^C).

718 packets received by filter
0 packets dropped by kernel

Refer to the explanation of Example 30-24 on page 595 for an explanation of the output format above.

The normal *four-way close* can be described as follows (see Figure 30-2):

1. The initiator (`wlmhost`) sends a FIN packet to the other party (`bolshoi`). The absolute sequence number is 1949791116 in the example above (line 1 from `wlmhost` to `bolshoi`).
2. The receiving party (`bolshoi`) acknowledges the initiator's FIN sequence number by incrementing the current sequence number by one to 1949791117 ($1949791116 + 1$) in the example above (line 2 from `bolshoi` to `wlmhost`).
3. The receiving party (`bolshoi`) sends its own FIN packet to the initiator (after the application that is using the connection has closed the socket) with sequence number 3641849741 in the example above (line 3 from `bolshoi` to `wlmhost`).
4. The initiator (`wlmhost`) acknowledges the receiving party's FIN sequence number by incrementing the current sequence number by one to 3641849742 ($3641849741 + 1$) in the example above (line 4 from `wlmhost` to `bolshoi`).

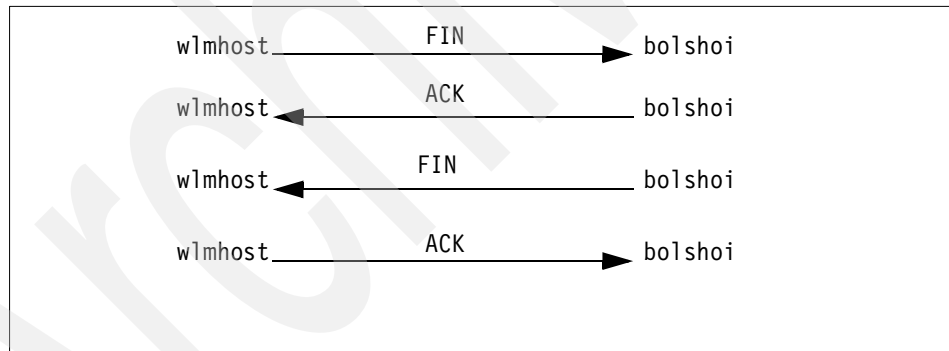


Figure 30-2 Schematic flow during TCP close

Monitoring UDP packets

Example 30-28 on page 600 shows how to use `tcpdump` to monitor UDP traffic. In the example, `tcpdump` will report only 10 packets (`-c 10`), read 1492 bytes from each packet (`-s 1492`), exclude the time stamp (`-t`), reporting will be interactive (`-l`) for UDP (`udp`) protocol only, and will omit the domain name part of host names (`-N`).

Example 30-28 Using tcpdump to monitor UDP

```
# tcpdump -c 10 -tNI 1492 udp
tcpdump: listening on tr0
snecac.1346 > 229.55.1.208.1345: udp 150]
wlmhost.33314 > dhcp001.domain: 29625+ PTR? 111.1.3.1.in-addr.arpa. (40)]
wlmhost.33314 > dhcp001.domain: 29625+ PTR? 111.1.3.1.in-addr.arpa. (40)]
dhcp001.domain > wlmhost.33314: 29625* 2/0/0 PTR snecac. (80)]
wlmhost.33315 > dhcp001.domain: 29626+ PTR? 208.150.1.229.in-addr.arpa. (45)]
wlmhost.33315 > dhcp001.domain: 29626+ PTR? 208.150.1.229.in-addr.arpa. (45)]
dhcp001.domain > wlmhost.33315: 29626 NXDomain 0/1/0 (118)]
wlmhost.33316 > dhcp001.domain: 29627+ PTR? 164.1.3.1.in-addr.arpa. (40)]
wlmhost.33316 > dhcp001.domain: 29627+ PTR? 164.1.3.1.in-addr.arpa. (40)]
dhcp001.domain > wlmhost.33316: 29627 NXDomain* 0/1/0 (135)]
65 packets received by filter
0 packets dropped by kernel
```

Some UDP services are recognized from the source or destination port number, and the higher-level protocol information is printed. In particular, Domain Name service requests and Sun RPC calls to NFS are recognized.

The general format of a UDP protocol line is:

```
SRC > DST: udp size
```

The following are the explanation of the fields:

SRC	Indicates the source (host) address and port. This field is always specified.
DST	Indicates the destination address and port. This field is always specified.
udp	Indicates that it is a UDP datagram.
size	Shows the user data packet's size in bytes.

Monitoring UDP domain name server requests

Example 30-29 shows how to use `tcpdump` to monitor DNS requests. In the example, `tcpdump` will report 10 packets (-c 10), read 143 bytes from each packet (-s 143), exclude the time stamp (-t), reporting will be interactive (-l), DNS (**port domain**) protocol only, and will omit the domain name part of host names (-N).

Example 30-29 Using tcpdump to monitor DNS⁵

```
# tcpdump -tNI 143 port domain
tcpdump: listening on tr0
wlmhost.33309 > dhcp001.domain: 33797+ A? www.ibm.com. (29)]
wlmhost.33309 > dhcp001.domain: 33797+ A? www.ibm.com. (29)]
```

⁵ Note that the monitoring was interrupted by using the Ctrl-C key sequence (^C).

```
wlmhost.33310 > dhcp001.domain: 56418+ PTR? 2.1.3.1.in-addr.arpa. (38]  
wlmhost.33310 > dhcp001.domain: 56418+ PTR? 2.1.3.1.in-addr.arpa. (38]  
^C  
434 packets received by filter  
0 packets dropped by kernel
```

The example shows two different DNS queries: one is for a name to IP address lookup, and the other type is for an IP address to name lookup (reverse lookup).

The two first packets have query ID 33797, have the recursive request flag set (+), and are address (A) records 29 bytes in size. Both the query class (CC_IN) and the query operation (Query) are omitted. The name to resolve is `www.ibm.com`.

The next two packets have query ID 56418, have the recursive request flag set (+), and are IP address (PTR) records 38 bytes in size. Both the query class (CC_IN) and the query operation (Query) are omitted. The address to resolve is `1.3.1.2` (reverse in the request).

DNS requests are formatted as⁶:

```
SRC > DST: id op? flags qtype qclass name (len)
```

The following are the explanation of the fields:

SRC	Indicates the source (host) address and port. This field is always specified.
DST	Indicates the destination address and port. This field is always specified.
id	Specifies the identification number of the query
op	Specifies the type of operation. The default is the query operation, which will be omitted. If the type of operation had been anything else, it would have been printed between the ID and the flags field.
flags	A plus sign (+) indicates that the recursion desired flag is set.
qtype	DNS query type.
qclass	Query class. Will be omitted if CC_IN. Any other qclass will be printed.
name	Name to resolve in reverse order.
(len)	Query length, not including the UDP and IP protocol headers.

Monitoring UDP name server responses

Example 30-30 on page 602 shows how to use `tcpdump` to monitor DNS responses. `tcpdump` will only report 10 packets (-c 10), read 143 bytes from each

⁶ A few anomalies are checked, and may result in extra fields enclosed in square brackets.

packet (-s 143), exclude the time stamp (-t), reporting will be interactive (-l), DNS (**port domain**) protocol only, and will omit the domain name part of host names (-N).

Example 30-30 Using tcpdump to monitor DNS

```
# tcpdump -tNI s 143 port domain
tcpdump: listening on tr0
dhcp001.domain > wlmhost.33360: 60043 2/2/2 CNAME 37.32.123.178.204.in-addr.arpa. (209)]
^C
434 packets received by filter
0 packets dropped by kernel
```

The packet has query ID 60043, two answer records, two DNS records, and two authoritative answers. The resolved address type is CNAME and the address is 204.178.123.32 (reverse in the reply). The packet length is 209 bytes.

DNS replies are formatted as:

```
SRC > DST: id op rcode flags a/n/au type class data (len)
```

The following are the explanation of the fields:

SRC	Indicates the source (host) address and port. This field is always specified.
DST	Indicates the destination address and port. This field is always specified.
id	Specifies the identification number of the query.
op	Specifies the type of operation. The default is the query operation, which will be omitted. If the type of operation had been anything else, it would have been printed between the ID and the flags field.
rcode	Response code, will be omitted if NoError. Any other rcode will be printed.
flags	An asterisk (*) indicates that the authoritative answer bit was set. Other flag characters that might appear are - (recursion available, RA, not set) and l (truncated message, TC, set). A plus sign (+) indicates that the recursion desired flag is set.
a/n/au	Answer records/Name server records/Authority records.
type	DNS record type.
class	DNS Record class. Will be omitted if <i>CC_IN</i> . Any other class will be printed.
data	Resolved reply.
(len)	Response length, not including the UDP and IP protocol headers.

Monitoring all packets

Example 30-31 shows how to use `tcpdump` to monitor all traffic on the default interface (in this case `tr0`). In the example, `tcpdump` will only report 16 packets (`-c 16`), read 17792 bytes from each packet⁷, print absolute rather than relative TCP sequence numbers (`-S`), verbose output (`-v`), not use DNS lookup for IP addresses to names (`-n`), and the reporting should be interactive (`-l`).

Example 30-31 Using tcpdump to monitor all packets

```
# tcpdump -c 16 -Sivns 17792
tcpdump: listening on tr0
19:08:08.148317911 1.3.1.106.1346 > 229.55.150.208.1345: udp 150] (ttl 10, id 24999)
19:08:08.230511146 1.3.1.114.2423 > 1.3.1.164.23: . ack 2686878608 win 16127 (DF)] (ttl 128, id 208)
19:08:08.325292354 0:6:29:1:72:4 80:1:43:0:0:0 0000 30:
      8000 0060 9480 4e20 0000 0000 8000 0060
      9480 4e20 8001 0000 1400 0200 0f00
19:08:08.341996876 1.3.1.75 > 224.1.1.5: OSPFv2-hello 48: rtrid 192.168.31.12 backbone E mask
255.255.255.0 int 10 pri 1 dead 40 dr 1.3.1.1 bdr 1.3.1.75 nbrs 1.3.1.1] [ttl 1] (id 53064)
19:08:08.441327515 1.39.7.76 > 1.3.1.164: icmp: echo request] (ttl 245, id 39306)
19:08:08.441440043 1.3.1.164 > 1.39.7.76: icmp: echo reply] (ttl 255, id 23286)
19:08:08.441742760 1.3.1.164 > 1.39.7.76: icmp: echo reply] (ttl 255, id 23286)
19:08:08.608477546 [[tokenring]
19:08:08.611070026 1.3.187.140.137 > 1.3.1.20.137: udp 68] (ttl 127, id 41648)
19:08:08.625016191 1.3.1.20.137 > 1.3.187.140.137: udp 62] (ttl 128, id 56925)
19:08:08.625763165 1.3.187.140.137 > 1.3.1.20.137: udp 68] (ttl 127, id 41649)
19:08:08.641570137 1.3.1.20.137 > 1.3.187.140.137: udp 62] (ttl 128, id 57181)
19:08:08.642107452 1.3.187.140.137 > 1.3.1.20.137: udp 68] (ttl 127, id 41650)
19:08:08.666561077 1.3.1.20.137 > 1.3.187.140.137: udp 62] (ttl 128, id 57437)
19:08:08.667100392 1.3.187.140.137 > 1.3.1.20.137: udp 68] (ttl 127, id 41651)
19:08:08.683200443 1.3.1.20.137 > 1.3.187.140.137: udp 62] (ttl 128, id 57693)
17 packets received by filter
0 packets dropped by kernel
```

In the example the marked packets are of the types (in top-down sequence):

- ▶ TCP packet
- ▶ Unknown protocol to tcpdump
- ▶ OSPF routing packet
- ▶ ICMP packet
- ▶ Truncated token-ring packet
- ▶ UDP packet

⁷ Note that this size will be copied from the kernel to the `tcpdump` command. Using smaller sizes reduces the risk of losing traced packets due to the increase in overhead that large copies can induce.

Interpreting link-level headers

Example 30-32 shows how to use **tcpdump** to monitor all traffic on the token-ring interface tr0. In the example, **tcpdump** will only report 6 packets (-c 6), exclude the time stamp (-t), read 35 bytes from each packet (-s 35), only include the link-level header (-e), and the reporting should be interactive (-l).

Example 30-32 Using tcpdump to monitor link-level headers

```
# tcpdump -c 6 -tes 35
tcpdump: listening on tr0
0:40:aa:49:4b:1b 0:60:94:8a:7:5b ip 36: [|ip]
0:60:94:8a:7:5b 0:40:aa:49:4b:1b ip 36: [|ip]
0:60:94:8a:7:5b 0:40:aa:49:4b:1b ip 36: [|ip]
0:40:aa:49:4b:1b 0:60:94:8a:7:5b ip 36: [|ip]
0:60:94:8a:7:5b 0:40:aa:49:4b:1b ip 36: [|ip]
0:60:94:8a:7:5b 0:40:aa:49:4b:1b ip 36: [|ip]
71 packets received by filter
0 packets dropped by kernel
```

In the example above the [|ip] indicates that the packet is an IP protocol packet but was truncated by **tcpdump**.

The general format of a the token-ring link-level report is:

SRC DST proto len:

The following are the explanation of the fields:

SRC	Source hardware address (MAC address)
DST	Destination hardware address (MAC address)
proto	The protocol type of the frame data
len	The amount of data read from the frame ⁸

Monitoring ARP packets

Example 30-33 shows how to use **tcpdump** to monitor ARP traffic on a network on a host. In the example, **tcpdump** will report 10 ARP packets (-c 10) and exclude the time stamp (-t), and reporting will be interactive (-l) for ARP (**arp**) on network 1.3.1 (**net**) and will omit the domain name part of host names (-N).

Example 30-33 Using tcpdump to monitor ARP

```
# tcpdump -c 10 -t -N -l arp net 1.3.1
tcpdump: listening on tr0
arp who-has 1.3.1.188 tell ANALYZER
arp who-has 3b-043 tell itsont00
arp reply 3b-043 is-at 0:60:94:87:a:87
arp who-has 1.3.1.188 tell 1.3.1.1
```

⁸ If the frame data is less than the size specified with -s, then the size of the read frame is reported.

```
arp who-has 1.3.1.188 tell wlmhost
arp who-has 1.3.1.188 tell wlmhost
arp who-has 1.3.1.188 tell wlmhost
arp who-has 1.3.1.188 tell wlmhost
arp who-has 3b-043 tell itsont00
arp reply 3b-043 is-at 0:40:aa:49:49:d4
120 packets received by filter
0 packets dropped by kernel
```

Lines starting with `arp who-has` are ARP requests from hosts on the network. Our hosts name is `wlmhost`, and there are four requests from our host to the network to resolve the `1.3.1.188` IP address. Lines starting with `arp reply` are replies to `who-has` requests. Lines with host names show hosts that the local system can resolve, and lines with IP addresses show those hosts that the local system *cannot* resolve.

Note in the output above that the host `itsont00` requests the IP address for host `3b-043` twice and each time it get a different MAC address. This can indicate potential problems that can become a performance issue for the `itsont00` host. If the host does not have enough space available in its ARP tables, this can lead to this behavior that will slow down communication with the `itsont00` host. The reason is that it will free up one ARP table entry each time it has to communicate with a host that does not have a entry. If the `itsont00` host communicates with many other systems, this will lead to what is known as *ARP cache thrashing*. The other issue is why the `itsont00` host receives different MAC addresses for the requested host (`3b-043`). One reason could be that the `3b-043` host uses multiple network adapters (for performance or availability reasons). Another reason could be that the hosts are on different networks with different routers, and at least two routers believe that they are the preferred route between the networks and therefore respond to ARP requests (proxy ARP). Yet a third reason could be that apart from the host itself that replies to ARP requests, there are at least one system on the network that acts like an ARP server but has the wrong MAC address in its ARP table⁹.

Notes on ARP handling

When an application sends an Internet packet to one of the interface drivers, the driver requests the appropriate address mapping. If the mapping is not in the table, an ARP broadcast packet is sent through the requesting interface driver to the hosts on the local area network. ARP requests are link-level broadcasts that go only to the local physical net unless proxy ARP is enabled in a router/gateway. ARP requests normally do not get forwarded through a router. For hosts reachable only through a router, the originating host will ARP only for the gateway, not for the end host.

⁹ When adding ARP table entries manually with the `arp` command and specifying the `pub` option, the local system will respond to requests as a ARP server for this entry.

The kernel maintains the translation table (ARP table) between Internet addresses and MAC addresses. The size of this table is made up of a hash table and bucket entries in this table. The following tunables are used for the translation table (refer to Chapter 34, “The no command” on page 665):

arptab_nb	Specifies the number of hash lines in the ARP table.
arptab_bsiz	Specifies the number of struct arptab entries in the hash table (<code>/usr/include/net/if_arp.h</code>).
arpt_killc	Specifies the time in minutes before a complete entry will be deleted from the translation table.
arpqsize	Specifies the maximum number of packets to queue while waiting for ARP responses. The IP packet to send will be linked to the <code>at_hold</code> pointer in the <code>arptab</code> struct.

IP addresses' places in the translation table are determined first by the remainder after a modulus operation is performed on the IP address (in hex) and by the size of the hash table (the `ARPTAB_HASH` define in `/usr/include/net/if_arp.h`). The bucket for the `arptab` struct is assigned sequentially, as is the bucket search (the `ARPTAB_LOOK` define in `/usr/include/net/if_arp.h`).

Note that the translation table may run out of free bucket entries before all buckets are filled with ARP entries. When an `arp` entry cannot be added to the translation table, the least recently used dynamically added entry is discarded (the `at_timer` variable in the `arptab` struct) includes the entry added by users.

When a packet to be sent to a host that does not have a mapping between IP address and MAC address in the translation table, the packet will be linked to the `arptab` struct entry for that IP address (the `at_hold` variable in the struct). This ensures that the packet will be sent immediately when the ARP request is returned with a IP address to MAC address mapping.

Using expressions

The `tcpdump` command has a powerful filtering mechanism; Example 30-34 illustrates how this can be used. Note that the indexing into packets is based from zero, as per “Accessing data inside a packet” on page 593. To debug your expression statements, use the `-d` flag as in the following example. (The output was piped through the `expand` and `n1` commands for referentiality.)

Example 30-34 Using the decoding of expressions

```
# tcpdump -d 'tcp[6] = 0xffffffff'|expand|n1
 1 (000) 1dh      [38]
 2 (001) jeq      #0x800          jt 2   jf 10
 3 (002) 1db      [49]
 4 (003) jeq      #0x6           jt 4   jf 10
 5 (004) 1dh      [46]
```

6	(005)	jset	#0x1fff	jt 10	jf 6
7	(006)	ldxb	4*([40]&0xf)		
8	(007)	ldb	[x + 46]		
9	(008)	jeq	#0xffffffff	jt 9	jf 10
10	(009)	ret	#80		
11	(010)	ret	#0		

Monitoring initiation and termination of TCP connections

To monitor only initiation and termination of TCP connections, specify the **tcpdump** command as in Example 30-35. (See “Monitoring all TCP traffic” on page 597).

Example 30-35 Using tcpdump to monitor start and stop TCP packets

```
# tcpdump -c 16 -NI 160 -i tr0 "(tcp[13] & 7 != 0)"
tcpdump: listening on tr0
1.3.1.43.2308 > ss12.1080: F 1419758927:1419758927(0) ack 1649681797 win 17000 (DF)]
1.3.1.43.2310 > ss12.1080: S 1419967325:1419967325(0) win 16384 <mss 4016,nop,nop,sack0K> (DF)]
1.3.1.43.2308 > ss12.1080: R 1419758928:1419758928(0) win 0 (DF)]
ss12.1080 > 1.3.1.43.2308: F 1649683845:1649683845(0) ack 1419758928 win 16060]
1.3.1.43.2308 > ss12.1080: R 1419758928:1419758928(0) win 0]
1.3.1.43.2308 > ss12.1080: R 1419758928:1419758928(0) win 0]
ss12.1080 > 1.3.1.43.2310: S 1553633667:1553633667(0) ack 1419967326 win 16060 <mss 1460>]
3b-054.2364 > www.80: R 2201046150:2201046150(0) win 0 (DF)]
3b-054.2370 > 208.80: S 2210195146:2210195146(0) win 16384 <mss 4016,nop,nop,sack0K> (DF)]
208.80 > 3b-054.2370: S 3491660221:3491660221(0) ack 2210195147 win 16384 <mss 1432>]
1.3.1.43.2310 > ss12.1080: R 1419967756:1419967756(0) win 0 (DF)]
1.3.1.43.2312 > ss12.1080: S 1420173609:1420173609(0) win 16384 <mss 4016,nop,nop,sack0K> (DF)]
204.174.18.152.80 > 3b-054.2369: F 3488338654:3488338654(0) ack 2209986206 win 16384]
3b-054.2369 > 204.174.1.152.80: F 2209986206:2209986206(0) ack 3488338655 win 16480 (DF)]
3b-054.2371 > www.80: S 2210246478:2210246478(0) win 16384 <mss 4016,nop,nop,sack0K> (DF)]
1.3.1.43.2310 > ss12.1080: R 1419967756:1419967756(0) win 0]
794 packets received by filter
0 packets dropped by kernel
```

In the output above the S is SYN, F is FIN, R is RST, and ack is ACK. In the SYN packet the connection setup options start with win (TCP window size), which is 16384. The mss option is the Maximum Segment Size (MSS) which is 4016. The next two options are nop (No operation). The last option is sack0K which is the Selective ACKnowledgement (SACK). This negotiated option is useful for recovering fast from multiple packet drops in a window of data, but it must be negotiated because not all TCP implementations support it.

Short IP packets

To monitor IP packets shorter than 1492 bytes sent through gateway 1.3.1.1, specify the **tcpdump** command as shown in Example 30-36 on page 608. Instead of MTU discover, the example uses mss_dflt set to the currently used MTU size

(1492) in the system. Note also the possible usage of the length primitives to monitor packet sizes (see “Expressions” on page 590), such as 'ip and len < 1492'.

Example 30-36 Using tcpdump to monitor small packets through gateways

```
# tcpdump -tNi tr0 -c 4 "host 1.3.1.1 and ip[2:2] < 1492"
tcpdump: listening on tr0
1.3.1.1 > ospf-all: OSPFv2-hello 48: backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1.3.1.1 > ospf-all: OSPFv2-hello 48: backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1.3.1.1 > itsoaus: icmp: redirect itsopok to host 1.3.1.75]
1.3.1.1 > ospf-all: OSPFv2-hello 48: backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
33730 packets received by filter
0 packets dropped by kernel
```

The output shows that there are only routing packets that are shorter than our specified size (1492) and no data packets. If there were shorter data packets, we could investigate this further by checking the PMTU field in the route table with **netstat** (Chapter 31, “The netstat command” on page 619), the **mss_dflt** with the **no** command (Chapter 34, “The no command” on page 665), or the **lsattr** command.

Example 30-37 assumes that we have found a data packet with a suspiciously small size by using **tcpdump** sent from the local host to a destination on the **tr0** interface.

Example 30-37 Checking TCP MSS

```
# ifconfig tr0
tr0: flags=e0a0043<UP,BROADCAST,RUNNING,ALLCAST,MULTICAST,GROUPRT,64BIT>
    inet 1.3.1.164 netmask 0xfffff00 broadcast 1.3.1.255
    tcp_sendspace 16284 tcp_recvspace 16384 tcp_mssdflt 512 rfc1323 0

# lsattr -El tr0 -a tcp_mssdflt
tcp_mssdflt N/A True

# netstat -rn|head -2;netstat -rn|grep tr0
Routing tables
Destination      Gateway          Flags   Refs      Use  If      PMTU  Exp  Groups
default          1.3.1.1         UGc     0          0   tr0     -    -
1.3.1/24         1.3.1.164      U       39      213259  tr0     -    -
1.39.7.76        1.3.1.1        UGHW    0         6877  tr0     -    2
1.39.7.84        1.3.1.1        UGHW    0         6883  tr0     -    2
1.53.148.171    1.3.1.1        UGHW    1           44  tr0     -    -
1.184.194.78    1.3.1.1        UGHW    1          973  tr0     -    -
...(lines omitted)...
```

```
# no -o tcp_mssdflt
```

```
tcp_mssdf1t = 512
```

Note that the order of precedence for how the MSS is set is used for the commands in this output. To use these commands, we need to know which interface and which IP address we are looking for. In the example, the **ifconfig** and **no** commands show that the segment (packet) size to use when communicating with remote networks (if the **no** variable `subnetsarelocal` is set to zero) is 512 bytes.

Notes on subnetsarelocal

The `subnetsarelocal` tunable (refer to Chapter 34, “The `no` command” on page 665) determines which MSS size to use. When `subnetsarelocal` is set to one (1), all IP addresses are viewed as local network addresses except when the IP network class address part is different. In that case it is considered a remote IP address.

A class address	First byte (#.0.0.0)
B class address	First and second byte (##.0.0)
C class address	First, second, and third byte (###.0)

For example: the hosts 1.6.6.1 and 1.6.7.2 with netmask 255.255.255.0 are considered to be local but the hosts 1.6.6.1 and 2.6.7.2 with netmask 255.255.255.0 are considered to be remote when `subnetsarelocal` is set to one.

ICMP packets

To monitor all ICMP packets that are *not* echo requests or replies (not **ping** packets), specify the **tcpdump** command as in Example 30-38 (refer to Table 3 on page 534).

Example 30-38 Monitoring non-echo request/reply ICMP packets with tcpdump

```
# tcpdump -tNi tr0 -c 10 "icmp[0] != 8 and icmp[0] != 0"
tcpdump: listening on tr0
rtrgfr1 > alexaix: icmp: time exceeded in-transit [tos 0xc0]
1.3.1.1 > m781bf01: icmp: redirect 1.24.106.202 to host 1.3.1.75]
wlmhost > dhcp001: icmp: wlmhost udp port 33324 unreachable]
wlmhost > dhcp001: icmp: wlmhost udp port 33324 unreachable]
rtrgfr1 > alexaix: icmp: time exceeded in-transit [tos 0xc0]
wlmhost > dhcp001: icmp: wlmhost udp port 33325 unreachable]
wlmhost > dhcp001: icmp: wlmhost udp port 33325 unreachable]
rtrgfr1 > alexaix: icmp: time exceeded in-transit [tos 0xc0]
wlmhost > dhcp001: icmp: wlmhost udp port 33326 unreachable]
wlmhost > dhcp001: icmp: wlmhost udp port 33326 unreachable]
1699 packets received by filter
0 packets dropped by kernel
```

The marked line in the output above indicates an ICMP route redirect. Normally a route redirection is used by gateway hosts or routers to indicate, to the sender of a packet that it has forwarded, that another preferred route to the destination of the forwarded packet exists.

Other protocols from the IP header

To monitor other protocols it is possible to examine the IP headers protocol field directly (see “TCP/IP protocol and services tables” on page 533) as in Example 30-39 that only traces protocol number 89, which is the Open Shortest Path First (OSPF) routing protocol.

Example 30-39 Using tcpdump to monitor other protocols

```
# tcpdump -c 4 -qtNI 120 -i tr0 'ip[9] = 89'
tcpdump: listening on tr0
1.3.1.75 > ospf-all: OSPFv2-hello 48: rtrid 1.1.31.12 backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1.3.1.1 > ospf-all: OSPFv2-hello 48: backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1.3.1.75 > ospf-all: OSPFv2-hello 48: rtrid 1.1.31.12 backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1.3.1.1 > ospf-all: OSPFv2-hello 48: backbone dr 1.3.1.1 bdr 1.3.1.75] [ttl 1]
1087 packets received by filter
0 packets dropped by kernel
```

Verbosity

The **tcpdump** command can display four levels of verbosity. The *default*, *verbose* (-v), *quick* (-q), and *quick verbose* (-qv). (The other flags in the following example are not important here.) The following short samples illustrate some of the differences between these verbosity levels. Example 30-40 shows the default output.

Example 30-40 Using default output

```
# tcpdump -c 4 -tnIs 512 -i tr0 'tcp[13] & 7 != 0'
tcpdump: listening on tr0
1.14.3.69.1080 > 1.3.1.41.3357: FP 2600873504:2600874006(502) ack 5851629 win 16060]
1.3.1.41.3357 > 1.14.3.69.1080: F 5851629:5851629(0) ack 2600874007 win 16825 (DF)]
1.3.1.41.3361 > 1.14.3.69.1080: S 9308623:9308623(0) win 16384 <mss 4016,nop,nop,sackOK> (DF)]
1.14.3.69.1080 > 1.3.1.41.3361: S 3338953794:3338953794(0) ack 9308624 win 16060 <mss 1460>]
63 packets received by filter
0 packets dropped by kernel
```

Example 30-41 shows the output with the verbose flag (-v).

Example 30-41 Using verbose output

```
# tcpdump -c 4 -vtnIs 512 -i tr0 'tcp[13] & 7 != 0'
tcpdump: listening on tr0
```



```
1.14.3.69.1080 > 1.3.1.41.3382: S 983077529:983077529(0) ack 11986781 win 16060 <mss 1460>]
(ttl 54, id 2849)
1.14.3.69.1080 > 1.3.1.41.3382: F 983078259:983078259(0) ack 11987132 win 16060] (ttl 54, id
3696)
1.3.1.41.3382 > 1.14.3.69.1080: F 11987132:11987132(0) ack 983078260 win 16791 (DF)] (ttl 128,
id 65064)
1.3.1.41.3384 > 1.14.3.69.1080: S 12254776:12254776(0) win 16384 <mss 4016,nop,nop,sackOK>
(DF)] (ttl 128, id 65065)
43 packets received by filter
0 packets dropped by kernel
```

Example 30-42 shows the output with the quick flag (-q).

Example 30-42 Using quick output

```
# tcpdump -c 4 -qtnIs 512 -i tr0 'tcp[13] & 7 != 0'
tcpdump: listening on tr0
1.14.3.50.1080 > 1.3.1.130.1224: tcp 0]
1.3.1.130.1224 > 1.14.3.50.1080: tcp 0 (DF)]
1.3.1.41.3405 > 1.14.3.69.1080: tcp 0 (DF)]
1.3.1.41.3413 > 1.14.3.69.1080: tcp 0 (DF)]
38 packets received by filter
0 packets dropped by kernel
```

Example 30-43 shows the output with the quick verbose flags (-qv).

Example 30-43 Using quick verbose output

```
# tcpdump -c 4 -qvtnIs 512 -i tr0 'tcp[13] & 7 != 0'
tcpdump: listening on tr0
1.14.3.69.1080 > 1.3.1.41.3370: tcp 0] (ttl 54, id 24025)
1.3.1.41.3203 > 1.14.3.69.1080: tcp 0 (DF)] (ttl 128, id 65347)
1.3.1.41.3370 > 1.14.3.69.1080: tcp 0 (DF)] (ttl 128, id 65348)
1.3.1.41.3407 > 1.14.3.69.1080: tcp 0 (DF)] (ttl 128, id 65354)
223 packets received by filter
0 packets dropped by kernel
```

Name resolution

The `tcpdump` command can display host addresses in three different ways. The default is to display both the host and the domain part. With the `-N` flag only the host part is displayed, and with the `-n` flag only the IP address is displayed and no name resolution is performed at all. The other flags in the examples are not important here. The following short example illustrates the differences between these name resolution settings. The first example shows the default output. The address part in the output above has been marked (host and domain name) in Example 30-44 on page 612.

Example 30-44 Default name resolution

```
# tcpdump -c 1 -ftIqvs 512 -i tr0 tcp
tcpdump: listening on tr0
3b-043.1897 > wlmhost.wow.ibm.telnet: tcp 0 (DF)] (ttl 128, id 57620)
11 packets received by filter
0 packets dropped by kernel
```

The address part in the previous output has been marked (host name) in Example 30-45.

Example 30-45 Host names only

```
# tcpdump -c 1 -NftIqvs 512 -i tr0 tcp
tcpdump: listening on tr0
3b-043.1896 > wlmhost.telnet: tcp 0 (DF)] (ttl 128, id 57466)
12 packets received by filter
0 packets dropped by kernel
```

The address part in the previous output has been marked (IP address) in Example 30-46.

Example 30-46 IP addresses only

```
# tcpdump -c 1 -nftIqvs 512 -i tr0 tcp
tcpdump: listening on tr0
1.14.4.71.1080 > 1.3.1.115.1068: tcp 8] (ttl 54, id 35696)
6 packets received by filter
0 packets dropped by kernel
```

30.10 trpt

The syntax of the **trpt** command is:

```
trpt [ -a ] [ -f ] [ -j ] [ -pAddress ]... [ -s ] [ -t ]
```

Flags

- | | |
|------------------|--|
| -a | Prints the values of the source and destination addresses for each packet recorded in addition to the normal output. |
| -f | Follows the trace as it occurs, waiting briefly for additional records each time the end of the log is reached. |
| -j | Lists just the protocol control block addresses for which trace records exist. |
| -pAddress | Shows only trace records associated with the protocol control block specified in hexadecimal by the Address |

variable. You must repeat the -p flag with each Address variable specified.

- s Prints a detailed description of the packet-sequencing information in addition to the normal output.
- t Prints the values for all timers at each point in the trace in addition to the normal output.

Parameters

Address The Address variable is the hexadecimal address of the TCP protocol control block to query for trace data.

30.10.1 Information about measurement and sampling

The **trpt** command queries the protocol control block (PCB) for TCP trace records. This buffer is created when a socket is marked for debugging with the **setsockopt** subroutine. The **trpt** command then prints a description of these trace records.

In order for the **trpt** command to work, the TCP application that is to be monitored must be able to set the **S0_DEBUG** flag with the **setsockopt** subroutine. If this is not possible you can enable this option for all new sockets that are created by using the **no** command with the **sodebug** option set to one:

```
no -o sodebug=1
```

Note that the **S0_DEBUG** flag will not be turned off for sockets that have this set even when the **sodebug** option is set to zero.

For schematic information about frame and packet headers refer to of “Packet header formats” on page 535.

30.11 Examples for trpt

The following examples show the output of **trpt** command after **sodebug** has been set to one (1) with the **no** command, and a **telnet** session has been started immediately thereafter. Note that all **trpt** reports query the stored TCP trace records from the PCB. Only when **trpt** is used with the -f flag will it follow the trace as it occurs (after it has displayed the currently stored trace records), waiting briefly for additional records each time the end of the log is reached.

For a detailed description of the output fields of the **trpt** command, review the command in *AIX 5L Version 5.2 Commands Reference, Volume 5*.

To list the PCB addresses for which trace records exist, use the `-j` parameter with the `trpt` command as in Example 30-47.

Example 30-47 Using trpt -j

```
# trpt -j
7064fbe8
```

You can check the PCB record with the `netstat` command as in Example 30-48.

Example 30-48 Using netstat -aA

```
# netstat -aA|head -2;netstat -aA |grep 7064fbe8
Active Internet connections (including servers)
PCB/ADDR Proto Recv-Q Send-Q Local Address Foreign Address (state)
7064fbe8 tcp 0 0 wlmhost.32826 wlmhost.telnet ESTABLISHED
```

The report format of the `netstat -aA` column layout is:

```
PCB/ADDR Proto Recv-Q Send-Q Local Address Foreign Address (state)
```

The following are the explanation of the fields:

PCB/ADDR	The PCB address
Proto	Protocol
Recv-Q	Receive queue size (in bytes)
Send-Q	Send queue size (in bytes)
Local Address	Local address
Foreign Address	Remote address
(state)	Internal state of the protocol

30.11.1 Displaying all stored trace records

When no option is specified, the `trpt` command prints all of the trace records found in the system and groups them according to their TCP connection PCB. Note that in the following examples, there is only one PCB opened with `SO_DEBUG` (7064fbe8). Example 30-49 shows the output during initialization.

Example 30-49 Using trpt during Telnet initialization

```
# trpt
7064fbe8:
365 CLOSED:user ATTACH -> CLOSED
365 SYN_SENT:output [fcbaf1a5..fcbaf1a9]@0(win=4000)<SYN> -> SYN_SENT
365 CLOSED:user CONNECT -> SYN_SENT
365 SYN_SENT:input 4b96e888@fcbaf1a6(win=4410)<SYN,ACK> -> ESTABLISHED
365 ESTABLISHED:output fcbaf1a6@4b96e889(win=4410)<ACK> -> ESTABLISHED
```

```
365 ESTABLISHED:output [fcbaf1a6..fcbaf1b5]@4b96e889(win=4410)<ACK,PUSH> -> ESTABLISHED
365 ESTABLISHED:user SEND -> ESTABLISHED
...(lines omitted)...
```

Example 30-50 shows the result of the `trpt` command after the `telnet` session is closed.

Example 30-50 Using trpt during telnet termination

```
# trpt
...(lines omitted)...
591 ESTABLISHED:output fcbaf1d3@4b96e913(win=4410)<ACK> -> ESTABLISHED
591 ESTABLISHED:input 4b96e913@fcbaf1d3(win=4410)<ACK,FIN> -> CLOSE_WAIT
591 CLOSE_WAIT:output fcbaf1d3@4b96e914(win=4410)<ACK> -> CLOSE_WAIT
591 LAST_ACK:output fcbaf1d3@4b96e914(win=4410)<ACK,FIN> -> LAST_ACK
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
```

30.11.2 Displaying source and destination addresses

To print the values of the source and destination addresses for each packet recorded in addition to the normal output, use the `-a` parameter with the `trpt` command as in Example 30-51. The following example contains the same information as the two examples in Example 30-49 on page 614 and Example 30-50, but with additional details. The reason for showing the full report is that it can be correlated with the examples mentioned. Note that even though the `telnet` session has ended, the TCP trace buffer still contains the protocol trace information (it was just a short connection).

Example 30-51 Using trpt -a

```
# trpt -a
7064f8e8:
365 CLOSED:user ATTACH -> CLOSED
365 SYN_SENT:output (src=1.3.1.164,32821, dst=1.3.1.164,23)[fcbaf1a5..fcbaf1a9]@0(win=4000) <SYN> -> SYN_SENT
365 CLOSED:user CONNECT -> SYN_SENT
365 SYN_SENT:input (src=1.3.1.164,23, dst=1.3.1.164,32821)4b96e888@fcbaf1a6(win=4410)<SYN,ACK> -> ESTABLISHED
365 ESTABLISHED:output (src=1.3.1.164,32821, dst=1.3.1.164,23)fcbaf1a6@4b96e889(win=4410)<ACK> -> ESTABLISHED
365 ESTABLISHED:output (src=1.3.1.164,32821, dst=1.3.1.164,23)[fcbaf1a6..fcbaf1b5]@4b96e889(win=4410)<ACK,PUSH> -> ESTABLISHED
365 ESTABLISHED:user SEND -> ESTABLISHED
...(lines omitted)...
591 ESTABLISHED:output (src=1.3.1.164,32821, dst=1.3.1.164,23)fcbaf1d3@4b96e913(win=4410)<ACK> -> ESTABLISHED
591 ESTABLISHED:input (src=1.3.1.164,32821, dst=1.3.1.164,32821)4b96e913@fcbaf1d3(win=4410)<ACK,FIN> -> CLOSE_WAIT
591 CLOSE_WAIT:output (src=1.3.1.164,32821, dst=1.3.1.164,23)fcbaf1d3@4b96e914(win=4410)<ACK> -> CLOSE_WAIT
591 LAST_ACK:output (src=1.3.1.164,32821, dst=1.3.1.164,23)fcbaf1d3@4b96e914(win=4410)<ACK,FIN> -> LAST_ACK
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
```

30.11.3 Displaying packet-sequencing information

To print a detailed description of the packet-sequencing information in addition to the normal output, use the `-s` parameter with the `trpt` command as in the Example 30-52. The following example contains the same information as Example 30-49 on page 614 and Example 30-50 on page 615, but with additional details.

Example 30-52 Using trpt -s

```
# trpt -s

7064fbe8:
365 CLOSED:user ATTACH -> CLOSED
    rcv_nxt 0 rcv_wnd 0 snd_una 0 snd_nxt 0 snd_max 0
    snd_wl1 0 snd_wl2 0 snd_wnd 0
365 SYN_SENT:output [fcba1a5..fcba1a9)@0(win=4000)<SYN> -> SYN_SENT
    rcv_nxt 0 rcv_wnd 0 snd_una fcba1a5 snd_nxt fcba1a6 snd_max fcba1a6
    snd_wl1 0 snd_wl2 0 snd_wnd 0
365 CLOSED:user CONNECT -> SYN_SENT
    rcv_nxt 0 rcv_wnd 0 snd_una fcba1a5 snd_nxt fcba1a6 snd_max fcba1a6
    snd_wl1 0 snd_wl2 0 snd_wnd 0
365 SYN_SENT:input 4b96e888@fcba1a6(win=4410)<SYN,ACK> -> ESTABLISHED
    rcv_nxt 4b96e889 rcv_wnd 4410 snd_una fcba1a6 snd_nxt fcba1a6 snd_max fcba1a6
    snd_wl1 4b96e889 snd_wl2 fcba1a6 snd_wnd 4410
...(lines omitted)...
591 LAST_ACK:output fcba1d3@4b96e914(win=4410)<ACK,FIN> -> LAST_ACK
    rcv_nxt 4b96e914 rcv_wnd 4410 snd_una fcba1d3 snd_nxt fcba1d4 snd_max fcba1d4
    snd_wl1 4b96e913 snd_wl2 fcba1d3 snd_wnd 4410
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
    rcv_nxt 4b96e914 rcv_wnd 4410 snd_una fcba1d3 snd_nxt fcba1d4 snd_max fcba1d4
    snd_wl1 4b96e913 snd_wl2 fcba1d3 snd_wnd 4410
```

30.11.4 Displaying timers at each point in the trace

To print the values for all timers at each point in the trace in addition to the normal output, use the `-t` parameter with the `trpt` command as in Example 30-53. The following example contains the same information as Example 30-49 on page 614 and Example 30-50 on page 615, but with additional details.

Example 30-53 Using trpt -t

```
# trpt -t

7064fbe8:
365 CLOSED:user ATTACH -> CLOSED
365 SYN_SENT:output [fcba1a5..fcba1a9)@0(win=4000)<SYN> -> SYN_SENT
    REXMT=6 (t_rxtshft=0), KEEP=150
365 CLOSED:user CONNECT -> SYN_SENT
```

```

    REXMT=6 (t_rxtshft=0), KEEP=150
365 SYN_SENT:input 4b96e888@fcba1a6(win=4410)<SYN,ACK> -> ESTABLISHED
365 ESTABLISHED:output fcba1a6@4b96e889(win=4410)<ACK> -> ESTABLISHED
365 ESTABLISHED:output [fcba1a6..fcba1b5)@4b96e889(win=4410)<ACK,PUSH> -> ESTABLISHED
    REXMT=3 (t_rxtshft=0)
365 ESTABLISHED:user SEND -> ESTABLISHED
    REXMT=3 (t_rxtshft=0)
...(lines omitted)...

591 ESTABLISHED:output fcba1d3@4b96e913(win=4410)<ACK> -> ESTABLISHED
591 ESTABLISHED:input 4b96e913@fcba1d3(win=4410)<ACK,FIN> -> CLOSE_WAIT
591 CLOSE_WAIT:output fcba1d3@4b96e914(win=4410)<ACK> -> CLOSE_WAIT
591 LAST_ACK:output fcba1d3@4b96e914(win=4410)<ACK,FIN> -> LAST_ACK
    REXMT=3 (t_rxtshft=0), 2MSL=1200
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
    REXMT=3 (t_rxtshft=0), 2MSL=1200

```

30.11.5 Printing trace records for a single protocol control block

Example 30-54 shows the trace record for a single protocol control block.

Example 30-54 Display the trace record associated with a protocol control block

```

# trpt -j
7057d1f0, 7089b9f0, 714ae5f0
# trpt -p 7057d1f0

7057d1f0:
520 CLOSED:user ATTACH -> CLOSED
520 CLOSED:user SOCKADDR -> CLOSED
520 SYN_SENT:output [cd913597..cd91359b)@0(win=4000)<SYN> -> SYN_SENT
520 CLOSED:user CONNECT -> SYN_SENT
520 SYN_SENT:input dde23b65@cd913598(win=16d0)<SYN,ACK> -> ESTABLISHED
520 ESTABLISHED:output cd913598@dde23b66(win=4470)<ACK> -> ESTABLISHED
520 ESTABLISHED:output [cd913598..cd913660)@dde23b66(win=4470)<ACK,PUSH> ->
ESTABLISHED
520 ESTABLISHED:user SEND -> ESTABLISHED
520 ESTABLISHED:input dde23b66@cd913660(win=1920)<ACK> -> ESTABLISHED
521 ESTABLISHED:input [dde23b66..dde23bc6)@cd913660(win=1920)<ACK,PUSH> ->
ESTABLISHED
.... ( lines omitted).....

```

Archived

The netstat command

The **netstat** command is a monitoring tool that displays a wide range of network status information. The **netstat** command is a useful tool for determining network problems, and it can provide information about the network traffic, the amount of data send and received by each protocol, and memory usage for network buffers.

netstat resides in `/usr/sbin/netstat`, is linked to `/usr/bin`, and is part of the `bos.net.tcp.client` fileset, which is installable from the AIX base installation media.

31.1 netstat

The syntax of the **netstat** command is:

```
netstat [ -n ] [ {-A -a} | {-r -C -i -I Interface} ]  
        [ -f AddressFamily ] [ -p Protocol ] [ Interval ] [ System ]  
netstat [ -m | -s | -ss | -u | -v ] [ -f AddressFamily ]  
        [ -p Protocol ] [ Interval ] [ System ]  
netstat -D  
netstat -c  
netstat -P  
netstat [ -Zc | -Zi | -Zm | -Zs ]
```

Flags

- A** Shows the address of any protocol control blocks associated with the sockets. This flag acts with the default display and is used for debugging purposes.
- a** Shows the state of all sockets. Without this flag, sockets used by server processes are not shown.
- c** Shows the statistics of the Network Buffer Cache (NBC). The NBC is a list of network buffers that contains data objects that can be transmitted to networks. The NBC grows dynamically as data objects are added to or removed from it. The network buffer cache is used by some network kernel interfaces for performance enhancement on the network I/O. Currently the `send_file()` system call, the `Frca*` family of system calls, and the `frctr1` command use the NBC.
- C** Shows the routing tables, including the user-configured and current costs of each route. The user-configured cost is set using the `-hopcount` flag of the `route` command. The current cost may be different from the user-configured cost if dead gateway detection has changed the cost of the route.
- D** Shows the number of packets received, transmitted, and dropped in the communications subsystem.
- f AddressFamily** Limits reports of statistics or address control blocks to those items specified by the `AddressFamily` variable.
- i** Shows the state of all configured interfaces.
- I Interface** Shows the state of the configured interface specified by the `Interface` variable.

- m** Shows statistics recorded by the memory management routines.
- n** Shows network addresses as numbers. When this flag is not specified, the **netstat** command interprets addresses where possible and displays them symbolically. This flag can be used with any of the display formats.
- p Protocol** Shows statistics about the value specified for the Protocol variable, which is either a well-known name for a protocol or an alias for it. Some protocol names and aliases are listed in the `/etc/protocols` file. A null response means that there are no numbers to report. The program report of the value specified for the Protocol variable is unknown if there is no statistics routine for it.
- P** Shows the statistics of the Data Link Provider Interface (DLPI). If DLPI is not loaded, it displays the message: `can't find symbol: dl_stats`
- r** Shows the routing tables. When used with the **-s** flag, the **-r** flag shows routing statistics.
- s** Shows statistics for each protocol.
- ss** Displays all the non-zero protocol statistics and provides a concise display.
- u** Displays information about domain sockets.
- v** Shows statistics for CDLI-based communications adapters. This flag causes the **netstat** command to run the **entstat**, **tokstat**, **fddistat**, and **atmstat** commands. No flags are issued to these commands. Refer to Chapter 29, “atmstat, entstat, estat, fddistat, and tokstat commands” on page 539 for more information about these commands.
- Zc** Clears network buffer cache statistics.
- Zi** Clears interface statistics.
- Zm** Clears network memory allocator statistics.
- Zs** Clears protocol statistics. To clear statistics for a specific protocol, use **-p Protocol**. For example, to clear TCP statistics, enter **netstat -Zs -p tcp**.

Parameters

Interface

The network interface, for example tok0.

AddressFamily	The address family. The following address families are recognized: inet Indicates the AF_INET address family. inet6 Indicates the AF_INET6 address family. ns Indicates the AF_NS address family. unix Indicates the AF_UNIX address family.
Protocol	Limits the output to statistics for this protocol. The protocol names and aliases are listed in the /etc/protocols file.
Interval	The interval in seconds the netstat command is run. The data reported following the header line shows the summary values collected since the last reset of these counters. The other lines show the data for the time interval only.
System	The memory used by the current kernel. This is /unix unless you are looking into a dump file.

31.1.1 Information about measurement and sampling

The **netstat** command reads from kernel memory. This is done during execution time. The **netstat -v** command calls **atmstat**, **entstat**, **fddistat**, and **tokstat** without parameters to display the adapter device-driver statistics. This includes:

- ▶ The display of active socket connections for each protocol. The local and remote addresses, send and receive queue sizes, protocol, and state of the protocol are displayed.
- ▶ The display of routing table information. The available routes and the status of these routes are shown. Each route consists of a destination host or network and the gateway to use to forward packets.
- ▶ The display of contents of a network data structure. Statistics recorded by the memory management subroutines that show the usage of communications memory buffers (mbufs) in the system and statistics for each protocol can be displayed.
- ▶ The display of packet counts throughout the communications subsystem. This shows the number of packets received, sent, and dropped in the communication subsystem.
- ▶ The display of network buffer cache statistics. The NBC is currently used by two pieces of code. One is the `send_file()` system call, which uses the NBC if the `SF_SYNC_CACHE` flag is set. The second is the Fast Response Cache Accelerator (FRCA), which is implemented by the FRCA kernel extension.

This kernel extension is basically an HTTP get engine that runs in the kernel. An API is provided as well as the `frctr1` command to control and use FRCA.

- ▶ The display of data link provider interface (DLPI) statistics. These statistics are available only if DLPI is loaded.
- ▶ The reset of statistics. The interface, network buffer cache, mbufs, and protocol statistics can be cleared.

Among the many outputs `netstat` can provide, only a few of them monitor the system for performance. These are:

netstat -v Refer to Chapter 29, “`atmstat`, `entstat`, `estat`, `fdistat`, and `tokstat` commands” on page 539 for detailed information about the outputs of these commands.

netstat -in Lists the network interfaces including the Maximum Transmission Unit (MTU), packets received and transmitted, and receive and transmit errors for each interface.

netstat -rn The output of this command shows the current routing table used by the system including the used Path Maximum Transfer Unit (PMTU). For two hosts communicating across a path of multiple networks, a transmitted packet becomes fragmented if its size is greater than the smallest MTU of any network in the path. Because packet fragmentation can result in reduced network performance, it is desirable to avoid fragmentation by transmitting packets with a size no greater than the smallest MTU in the network path.

netstat -m Displays statistics for the communications memory buffer (mbuf) usage. Each processor has its own mbuf pool. If the network option `extendednetstats` is set to 1, a summary of all processors is collected and displayed. For performance reasons `extendednetstats` is set to 0 (zero) in `/etc/rc.net`. Refer to Chapter 34, “The `no` command” on page 665 for more information about the `no` command. In a multiprocessor system only one processor can update these summary values at a time, and this will block the other processors trying to update these summary values.

netstat -s The output of this command shows detailed statistics for each network protocol used. This includes packets sent and received, packets dropped, and error counters. The `netstat -p Protocol` command can be used to display the data only for this one protocol. This is useful if you are only interested in the statistics for one protocol, for example User Datagram Protocol (UDP). Using the `netstat -p udp` command shows only the statistics for UDP.

netstat -D This command shows the count of packets transmitted and received as well as the count for dropped packets for each layer in the communications subsystem.

netstat -an The output of this command shows the state of all sockets including the current sizes for their receive and send queues.

netstat -c This command provides statistics about the NBC usage.

For a detailed description of all other flags of the **netstat** command, refer to the *AIX 5L Version 5.2 Commands Reference*.

31.1.2 Examples for netstat

In this section we show the outputs of the **netstat** commands useful for performance monitoring and show which parts of these outputs should be inspected first.

The network interfaces

First, the state of the configured network interfaces should be observed using the **netstat -in** command. Example 31-1 shows the **netstat -in** command output.

Example 31-1 Output of the netstat -in command

```
# netstat -in
```

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
lo0	16896	link#1		182	0	188	0	0
lo0	16896	127	127.0.0.1	182	0	188	0	0
lo0	16896	::1		182	0	188	0	0
en0	1500	link#2	2.60.8c.f5.1c.fc	956	0	994	0	0
en0	1500	9.49.59.128	9.49.59.163	956	0	994	0	0
tr0	1492	link#3	8.0.5a.d.97.16	5925	0	240	0	0
tr0	1492	9.49	9.49.7.84	5925	0	240	0	0
css0	65520	link#4		47517	0	93533	0	0
css0	65520	9.49.59.64	9.49.59.99	47517	0	93533	0	0

The following lists the columns in Example 31-1:

Name The name of the network interface. In this case there are three network interfaces providing connections to three different networks; en0, tr0, and css0. An interface marked with a star, for example css0*, indicates that this network interface is currently down and cannot be used until it is set to the up state again using the **ifconfig Interface up** command.

Mtu The MTU size set for each network interface.

Network	The network address and the adapter hardware address for each network interface.
Address	IP address of the network interfaces.
Ipkts	Packets received on the network interface.
Ierrs	Receive errors on this network interface.
Opkts	Packets sent to the interface.
Oerrs	Transmit errors on the interface.
Coll	The number of collisions on the interface. The collision count for Ethernet interfaces is not supported.

The network address and the IP address for each interface should be correct. The MTU sizes for the network interfaces should be set to valid values. For example, all systems connected to one local network should use the same MTU on the network interface connecting them to this local network. The values for Ierrs and Oerrs should be zero, and if they are not zero the value should be very low and should not increase. The ratio between received packets and sent packets shows how the network interfaces are used. The previous example shows more packets received on the tr0 interface than this system sent. In case this system is a client system and it connects to its server through the tr0 interface, the numbers in the sample are acceptable. However, it is a good idea to take a closer look and find the reason why we received this much more data on tr0 than we sent.

The numbers of the packets received and sent over all network interfaces shows which interface gets used most. Some network traffic may be redirected to other network interfaces to balance the load. This can be done by changing the routing table by, for example, adding static host routes. However, you should always be careful in changing routing information on a system, and be aware of any changes on the other systems.

There are different ways to learn why the number of packets received on tr0 is 10 times higher than the number sent to this network interface. One is to run **netstat -s**, but for that command all traffic over all interfaces is taken into account. Another way is to start at the adapter and device driver layer using **netstat -v** or the **tokstat** command and see if there are any unusual numbers. Example 31-2 uses **tokstat** to check the token-ring device-driver statistics first.

Example 31-2 Output of tokstat tok0 command

```
# tokstat tok0
-----
TOKEN-RING STATISTICS (tok0) :
Device Type: Token-Ring High-Performance Adapter (8fa2)
Hardware Address: 08:00:5a:0d:97:16
```

Elapsed Time: 0 days 3 hours 28 minutes 9 seconds

Transmit Statistics:

Packets: 37355
Bytes: 4042801
Interrupts: 37353
Transmit Errors: 0
Packets Dropped: 0

Max Packets on S/W Transmit Queue: 27
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 0

Broadcast Packets: 11
Multicast Packets: 2
Timeout Errors: 0
Current SW Transmit Queue Length: 0
Current HW Transmit Queue Length: 0

General Statistics:

No mbuf Errors: 0
Abort Errors: 0
Burst Errors: 0
Frequency Errors: 0
Internal Errors: 0
Lost Frame Errors: 0
Token Errors: 0
Ring Recovered: 0
Soft Errors: 0

Driver Flags: Up Broadcast Running

AlternateAddress 64BitSupport ReceiveFunctionalAddr
16 Mbps

Receive Statistics:

Packets: 302050
Bytes: 24739052
Interrupts: 300777
Receive Errors: 0
Packets Dropped: 0
Bad Packets: 0

Broadcast Packets: 257020
Multicast Packets: 6992
Receive Congestion Errors: 0

Lobe Wire Faults: 0
AC Errors: 0
Frame Copy Errors: 0
Hard Errors: 0
Line Errors: 0
Only Station: 0
Remove Received: 0
Signal Loss Errors: 0
Transmit Beacon Errors: 0

This output of the **tokstat** command shows a very high number of broadcasts received. Most of these broadcasts may not even be useful for this system. We can perform a **netstat -p udp** command to see if there are datagrams dropped due to no socket. Example 31-11 on page 637 shows this. Here, some personal computers running Windows are connected to the net and sending many broadcasts.

The commands **iptrace** and **tcpdump**, for example **tcpdump -i tr0 ip broadcast**, should be used now to find the source or sources for all broadcasts.

The network routing

After inspecting the network interfaces the systems routing information should be validated. Wrong routing can result in poor performance. It is necessary to know the network topology to understand the current route settings. Example 31-3 shows an example output of the `netstat -rn` command.

Example 31-3 Output of `netstat -rn` command

```
# netstat -rn
Routing tables

Destination      Gateway          Flags   Refs      Use If      PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          9.49.8.1        UGc     0          0 tr0      -    -
9.3.9.165       9.49.8.1        UGHW    1         2344 tr0      -    -
9.49/20          9.49.7.84      U       7         1809 tr0      -    -
9.49.59.64/26   9.49.59.99    U       2        869707 css0     -    -
9.49.59.128/26 9.49.59.163  U       0         5545 en0      -    -
9.23.123.33    9.49.8.1        UGHW    0          4 tr0      -    1
9.23.123.34    9.49.8.1        UGHW    1          31 tr0      -    -
127/8           127.0.0.1       U       0          155 lo0     -    -

Route Tree for Protocol Family 24 (Internet v6):
::1              ::1              UH      0          0 lo0 16896  -
```

Destination The address of the destination network or host. The host addresses are highlighted in the sample. The routes to these hosts are cloned routes, which is indicated by the W in the Flags column. All traffic to and from these hosts goes through the tr0 interface and the default gateway 9.49.8.1.

Gateway The gateway used to reach the destination. The highlighted gateway addresses are the addresses of the network interfaces of this system. In our example all network traffic to the destination networks uses the interface connected to that network.

Use The number of packets sent using that route. In our example the most packets are sent to the css0 interface and the destination was on that network. No gateway was used.

PMTU Path Maximum Transfer Unit used for that route. For two hosts communicating across a path of multiple networks, a transmitted packet becomes fragmented if its size is greater than the smallest MTU of any network in the path. Because packet fragmentation can result in reduced network performance, it is desirable to avoid fragmentation by transmitting packets with a size no greater than the smallest MTU in the network path. In the example all data to the highlighted destinations is sent

through network interface tr0 to the default gateway 9.49.8.1. The MTU size for tr0 is 1492. (Refer to Example 31-1 on page 624.) To set a PMTU value, use either PMTU discovery, by using the **no -o tcp_pmtu_discover** and **no -o udp_pmtu_discover** commands, or add a static route including a PMTU value, for example **route add -host 9.3.9.165 9.49.8.1 -mtu 512**. The MTU set using the **route** command has no affect on MTU for applications using UDP. Refer to Chapter 34, “The no command” on page 665 for more details about the use of the **no** command.

The next sample (Example 31-4) shows a **netstat -rn** output after the host route to destination 9.3.9.165, including a PMTU of 512, was set using the **route add -host 9.3.9.165 9.49.8.1 -mtu 512** command.

Example 31-4 Output of netstat -rn including PMTU set to a fixed value

```
netstat -rn
Routing tables
Destination      Gateway          Flags  Refs      Use  If      PMTU  Exp  Groups

Route Tree for Protocol Family 2 (Internet):
default          9.49.8.1         UGc    0          0  tr0     -    -
9.3.9.165       9.49.8.1        UGH    1       4980 tr0   512  -
9.49/20          9.49.7.84        U       7         2434  tr0     -    -
9.49.59.64/26    9.49.59.99       U       0        969800  css0   -    -
9.49.59.128/26   9.49.59.163      U       2         10561  en0    -    -
9.23.123.34      9.49.8.1         UGHW   1          5882  tr0     -    -
127/8            127.0.0.1        U       0          159   lo0     -    -

Route Tree for Protocol Family 24 (Internet v6):
::1              ::1              UH      0           0  lo0    16896  -
```

Now all TCP traffic to destination 9.3.9.165 is broken up in packets of 512 bytes or less.

Note: In the example, the PMTU size of 512 used for destination 9.3.9.165 does not reflect the optimum PMTU size to use. To set the PMTU size it is necessary to know all MTU sizes on the networks your data travels to reach the destination. Use the **-mtu** option to the **route** command with care.

Kernel malloc statistics

The various layers of the communication subsystem share common buffer pools, the communications memory buffers (mbufs). The mbuf management facility controls buffer sizes. The buffer pools consists of pinned kernel memory. Pointers to mbufs passed from one layer of the communication subsystem to another reduces mbuf management overhead and avoids copying of data.

The maximum amount of memory the system can use for mbufs is defined in the system configuration. Use the command `lsattr -E1 sys0 -a maxmbuf` to control the current value set, and `lsattr -R1 sys0 -a maxmbuf` to see the possible values. The `maxmbuf` value can be changed by using the `chdev -l sys0 -a maxmbuf=NewValue` command. A change requires a reboot of the system to become activated.

If `maxmbuf` in the system configuration is zero, then the network option `thewall` defines the maximum amount of memory to be used. Use the `no` command to control and change `thewall`. (Refer to Chapter 34, “The `no` command” on page 665 for more information.) The `thewall` value is a runtime parameter and can be changed at any time.

On a multiprocessor system each processor manages its own mbuf pool. This is done to avoid unnecessary waits for locks that may occur if all processors are using the same mbuf pool. The `netstat -m` command is used to observe the system’s mbuf usage. Example 31-5 shows an example for the `netstat -m` output on a multiprocessor system with the network option `extendednetstats` set to zero.

Example 31-5 netstat -m output with extendednetstats=0

```
# netstat -m

Kernel malloc statistics:

***** CPU 0 *****
By size      inuse      calls failed  delayed    free    hiwat    freed
32           122        3425    0          0        134     1013     0
64           29         329     0          0         35     506      0
128          23        101446  0          0        233     253      11
256          147       591632  84         0        589     608      74
512          28         4942   0          0         28     63       0
1024         4          1714   0          0         4      158      0
2048         15         3940  149        0        153     158     104
4096         127       17943   0          0         88     190      0
8192         1          163    0          0         7      15       0
16384        0          253    0          0         38     38       2

***** CPU 1 *****
...lines omitted...statistics for other CPUs removed

By type      inuse      calls failed delayed    memuse    memmax    mapb

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
```

The columns of the mbuf statistics per CPU are:

By size	The size of the mbufs. Each processor's mbuf pool is split up into buckets between 32 and 16384 bytes.
inuse	The number of mbufs in use for the given mbuf size.
calls	The usage summary for the given mbuf size.
failed	The failed requests to acquire an mbuf. This value should be zero. If it is not zero then this number of requests for mbufs fails, causing incoming packets to be dropped. Tuning the maxmbuf system parameter or thewall network option is required.
delayed	The number of delayed requests for mbufs. The requester of an mbuf can specify the M_WAIT flag to get put to sleep if no mbuf is available. The requester will be woken up if mbuf space becomes available again. A user may notice a performance loss if the application is waiting for a delayed mbuf request. This value should stay zero.
free	An application can free a previous requested mbuf. The mbuf stays pinned in memory and can be used again. This avoids some overhead in managing mbufs, which includes unpinning and freeing the memory for general system usage.
hiwat	If the number of buffers on the free list reaches this high water mark, buffers from the free list are given back to the system. The high water mark is scaled by the system based on the amount of installed memory.
freed	A mbuf given back to the system increments the freed count. If these values consistently increase, the high water mark is too low, which causes unnecessary memory management overhead. The high water mark value cannot be changed.

Setting the network option `extendednetstats` to a value of one using the `no -o extendednetstats=1` command will enable `netstat -m` to provide additional information. However, this will cost performance on a multiprocessor system and should only be used to aid problem determination. Example 31-6 shows `netstat -m` output on a multiprocessor system with `extendednetstats` enabled. When enabling extended netstat a reboot is required for it to take effect.

Example 31-6 netstat -m output with extendednetstats=1

```
# netstat -m

521 mbufs in use:
512 mbuf cluster pages in use
```

```

2178 Kbytes allocated to mbufs
0 requests for mbufs denied
0 calls to protocol drain routines
0 sockets not created because sockthresh was reached

```

```
Kernel malloc statistics:
```

```
***** CPU 0 *****
```

```
By size          inuse      calls failed  delayed   free  hiwat  freed
... statistics for each CPU removed ...
```

By type	inuse	calls failed	delayed	memuse	memmax	mapb
mbuf	521	6048	0	0	133376	305408
mcluster	512	28	0	0	2097152	3149824
socket	20	28	0	0	19648	35392
pcb	6	9	0	0	416	1600
routetbl	24	0	0	0	2944	3584
ifaddr	16	0	0	0	1792	1792
mblk	7	422	0	0	896	119040
mbldata	30	1	0	0	30720	151808
strhead	12	0	0	0	2304	5376
strqueue	11	0	0	0	5632	13312
strmodsw	16	0	0	0	1024	1024
strosr	0	0	0	0	0	512
strsyncq	17	0	0	0	1728	3712
streams	60	0	0	0	12224	17600
devbuf	0	0	0	0	0	528384
kernel tablemoun	17	0	0	0	86368	89440
spec buf	1	0	0	0	128	128
locking	2	0	0	0	256	256
temp	10	0	0	0	8640	16928
mcast opts	0	0	0	0	0	128
mcast addr	3	0	0	0	192	192

```
Streams mblk statistic failures:
```

```
0 high priority mblk failures
```

```
0 medium priority mblk failures
```

```
0 low priority mblk failures
```

The summary statistic is displayed in the first part of the output. This includes information about the current memory usage for mbufs. The value for requests for mbufs denied should be zero. The value for sockets not created because sockthresh was reached should also be zero. It indicates that allocation of mbufs for a new socket connection failed. The network option sockthresh, which defaults to 85, allows mbuf allocation to new sockets only if less than 85 percent of the maximum mbuf space is in use. The value for sockthresh can be changed using the **no -o sockthresh=NewValue** command.

At the end of the statistics output a detailed usage of mbufs per service is displayed.

By type	This column names the service, such as mbuf or socket.
inuse	Shows the number of mbufs used by the specific service.
calls	Show the usage count for mbufs by the specific service.
failed	The failed mbuf requests for the specific services. These values should be zero. If they are not zero, then tuning of the maxmbuf system parameter or the thewall network option is necessary.
delayed	The number of delayed mbuf requests. The requester of an mbuf can specify the M_WAIT flag to get put to sleep if no mbuf is available. The requester will be woken up if mbuf space becomes available again. An user may notice a degradation of performance if the application is waiting for a delayed mbuf request. These values should stay at zero. If they are not zero, tuning of the maxmbuf system parameter or the thewall network option is necessary.

Statistics for each protocol

These statistics show detailed information including packet counts and error counts for each protocol used on the system. The **netstat -s** command shows the statistics for all protocols configured on the system as shown in Example 31-7. Using **netstat -p Protocol** shows the statistics only for this one protocol. We will show only the **netstat -p Protocol** outputs to keep the examples smaller.

The **netstat -p ip** command displays statistics for the IP protocol.

Example 31-7 Output from the netstat -p ip command

```
# netstat -p ip
ip:
 2935539 total packets received
 0 bad header checksums
 0 with size smaller than minimum
 0 with data size < data length
 0 with header length < data size
 0 with data length < header length
 0 with bad options
 0 with incorrect version number
 0 fragments received
 0 fragments dropped (dup or out of space)
 0 fragments dropped after timeout
 0 packets reassembled ok
2769505 packets for this host
 38 packets for unknown/unsupported protocol
```

```
0 packets forwarded
0 packets not forwardable
0 redirects sent
1624868 packets sent from this host
0 packets sent with fabricated ip header
0 output packets dropped due to no bufs, etc.
0 output packets discarded due to no route
1076 output datagrams fragmented
0 fragments created
1076 datagrams that can't be fragmented
165994 IP Multicast packets dropped due to no receiver
147 successful path MTU discovery cycles
46 path MTU rediscovery cycles attempted
1 path MTU discovery no-response estimate
2 path MTU discovery response timeouts
0 path MTU discovery decreases detected
152 path MTU discovery packets sent
0 path MTU discovery memory allocation failures
0 ipintrq overflows
0 with illegal source
0 packets processed by threads
0 packets dropped by threads
0 packets dropped due to the full socket receive buffer
0 dead gateway detection packets sent
0 dead gateway detection packet allocation failures
0 dead gateway detection gateway allocation failures
```

The IP protocol statistics show information about sent and received packets, error counters, statistics regarding forwarding of packets, and MTU discovery. The error counters should be zero or very low values compared with the received and sent packets counters. Any unusually high number of bad header checksums, packets with size smaller than minimum, packets with data size < data length, packets with header length < data size, packets with data length < header length, packets with bad options, or packets with incorrect version number indicates a problem on the network. Some other system may send such packets or there may be a hardware problem on the network. Tools such as `iptrace` or `tcpdump` can be used to identify the source of these invalid packets.

In case the system is set up to be a router, the fields `packets forwarded`, `packets not forwardable`, and `redirects sent` will show this. The number of packets not forwardable should be low. The `netstat -rn` should be used to see if all destination networks are still reachable. Refer to “The network routing” on page 627 for more information.

In Example 31-7 on page 632 there is a difference between the `packets received` and `packets for this host`. No packets were forwarded. However,

there are some packets for unknown/unsupported protocol and IP Multicast packets dropped due to no receiver, which means the systems received these packets but no listener on the system is running to use them. The **iptrace** or the **tcpdump** should show the source of these packets. Because such packets put additional load on the system and the network, check if the configuration on the source system for these packets is incorrect and, if so, correct it.

The **netstat -p icmp** command shows statistics for the Internet Control Message Protocol (ICMP) protocol as shown in Example 31-8.

Example 31-8 Output of the netstat -p icmp command

```
# netstat -p icmp
icmp:
  22 calls to icmp_error
  0 errors not generated because old message was icmp
  Output histogram:
    echo reply: 187807
    destination unreachable: 22
  0 messages with bad code fields
  0 messages < minimum length
  0 bad checksums
  0 messages with bad length
  Input histogram:
    echo reply: 149
    destination unreachable: 22
    echo: 187811
    time exceeded: 8
  187807 message responses generated
```

The error counters in this output should stay close to zero. A value greater than zero for the fields `messages with bad code fields`, `messages < minimum length`, `bad checksums`, and `messages with bad length` are an indication of a network problem. The **tcpdump** and **iptrace** command can be used for problem determination.

The output for the **netstat -p igmp** command is part of the **netstat -s** output and displays information for the Internet Group Multicast Protocol (IGMP) as shown in Example 31-9.

Example 31-9 Output of netstat -p igmp command

```
# netstat -p igmp
igmp:
  8 messages received
  0 messages received with too few bytes
  0 messages received with bad checksum
  0 membership queries received
  0 membership queries received with invalid field(s)
```



```
8 membership reports received
0 membership reports received with invalid field(s)
8 membership reports received for groups to which we belong
2 membership reports sent
```

This output is not very useful because there was not much traffic for the IGMP protocol on this system. However, messages received with too few bytes and messages received with bad checksum with values greater than zero point to network problems. The `tcpdump` and `iptrace` commands can be used for further problem determination.

TCP is the most widely used protocol on AIX systems. The `netstat -p tcp` command shows the statistics for this network protocol, as shown in Example 31-10.

Example 31-10 Output of netstat -p tcp command

```
# netstat -p tcp
tcp:
  1426798 packets sent
    380151 data packets (115215459 bytes)
    12 data packets (7860 bytes) retransmitted
    1003361 ack-only packets (22553 delayed)
    0 URG only packets
    1 window probe packet
    32137 window update packets
    11136 control packets
  2475620 packets received
    314523 acks (for 115227991 bytes)
    3966 duplicate acks
    0 acks for unsent data
    2343535 packets (3092077757 bytes) received in-sequence
    195 completely duplicate packets (125439 bytes)
    0 old duplicate packets
    7 packets with some dup. data (6692 bytes duped)
    4101 out-of-order packets (560796 bytes)
    0 packets (0 bytes) of data after window
    0 window probes
    113 window update packets
    2 packets received after close
    0 packets with bad hardware assisted checksum
    0 discarded for bad checksums
    0 discarded for bad header offset fields
    0 discarded because packet too short
    7 discarded by listeners
    23333 ack packet headers correctly predicted
    2140489 data packet headers correctly predicted
  3741 connection requests
  3707 connection accepts
```

```
7444 connections established (including accepts)
9245 connections closed (including 29 drops)
0 connections with ECN capability
0 times responded to ECN
4 embryonic connections dropped
271572 segments updated rtt (of 271586 attempts)
0 segments with congestion window reduced bit set
0 segments with congestion experienced bit set
0 resends due to path MTU discovery
1 path MTU discovery termination due to retransmits
13 retransmit timeouts
    0 connections dropped by retransmit timeout
4 fast retransmits
    0 when congestion window less than 4 segments
0 newreno retransmits
0 times avoided false fast retransmits
1 persist timeout
    0 connections dropped due to persist timeout
136 keepalive timeouts
    135 keepalive probes sent
    1 connection dropped by keepalive
0 times SACK blocks array is extended
0 times SACK holes array is extended
0 packets dropped due to memory allocation failure
0 connections in timewait reused
0 delayed ACKs for SYN
0 delayed ACKs for FIN
0 send_and_disconnects
0 spliced connections
0 spliced connections closed
0 spliced connections reset
0 spliced connections timeout
0 spliced connections persist timeout
0 spliced connections keepalive timeout
```

There are different performance areas to look at in this statistics output:

- ▶ The number of retransmits
Packets are dropped and a retransmission is performed. The cause for dropped packets could be CRC errors, poor or noisy cables, not enough receive buffers, remote node responding not in time, or that switches or routers along the route are dropping packets. Retransmission of packets could result in poor performance. The number of retransmissions should stay low compared to the number of packets sent. The **tcpdump** and **iptrace**, as well as performance monitoring of the receiving system, can be used to find the cause of the retransmits. The receiving system may be low on mbufs and drops the packets. Refer to Chapter 30, “TCP/IP packet tracing tools” on page 567 for more information about the **tcpdump** and **iptrace** commands.

- ▶ The number of delayed packets
This points to possible `tcp_nodelay` problems. `tcp_nodelay` specifies whether TCP should follow the Nagle algorithm for deciding when to send data. By default TCP will follow the Nagle algorithm. To disable this behavior, applications can enable `tcp_nodelay` to force TCP to always send data immediately. This can be done by the application. The Interface Specific Network Options (ISNO) and the `ifconfig` and `chdev` commands can be used to enable `tcp_nodelay` for each network interface. To use ISNO the network option `use_isno` must be set to a value of one. This is done using the `no -o use_isno=1` command. To set `tcp_nodelay` for one network interface the `ifconfig` and `chdev` commands are used, for example `ifconfig en0 tcp_nodelay 1`. The network option `tcp_nagle_limit` can be set to 1 (one) using the command `no -o tcp_nagle_limit=1` to disable the Nagle algorithm. Refer to Chapter 34, “The `no` command” on page 665 for more details about the `no` command.
- ▶ Packets received out of order
The sender transmits the packets in order, so there must be a reason why the system receives the packets out of order. One reason could be dropped packets due to our system is running out of mbufs. Problems with routing, for example the incoming packets using different routes, can cause out of order packets, or a router on the network path could be dropping packets. If the number of out of order packets received reaches an unusually high number compared with the total packets received, then further investigation is necessary. The `tcpdump` and `iptrace` commands can be used, as well as the `ping -R` and `traceroute` commands.
- ▶ The window probe
If the TCP window size of the receiving side of a connection is zero, then the sending side stops transmitting data and waits for an update of the receiver’s TCP window size. If the sender does not get this update it gets a timeout and sends a window probe packet. This always has a negative impact on network performance. The window probe packet value should remain at zero. The `windows probes` field in the receive section of the output is the probes for the TCP window size that the systems received. The consequences are the same as on the sent side. Tuning the `tcp_recvspace` using the `no` command is necessary if the window probe count gets too high. Refer to Chapter 34, “The `no` command” on page 665 for more information.

The `netstat -p udp` command is used to display UDP protocol statistics, as shown in Example 31-11.

Example 31-11 Output of netstat -p udp command

```
# netstat -p udp
udp:
    105925 datagrams received
    0 incomplete headers
```

```

0 bad data length fields
0 bad checksums
22 dropped due to no socket
92472 broadcast/multicast datagrams dropped due to no socket
0 socket buffer overflows
13431 delivered
9930 datagrams output

```

The values for incomplete headers, bad data length fields, bad checksums, and socket buffer overflows should stay at zero. Errors in the first three fields point to network problems so further investigation is necessary using the **tcpdump** and **iptrace** commands. For more information about these commands, refer to Chapter 30, “TCP/IP packet tracing tools” on page 567. In case of socket buffer overflows the network options **sb_max**, **upd_sendspace**, and **udp_recvspace** should be checked using the **no** command. Refer to Chapter 34, “The no command” on page 665 for more details. A high number of datagrams in the broadcast/multicast datagrams dropped due to no socket field compared to the total number of received datagrams points to a large number of broadcasts or multicast datagrams on the network. Our system has no listener running for the datagrams and the packets are dropped. The **tcpdump -i Interface ip broadcast** will show the source of these broadcasts.

Communications subsystems statistics

The **netstat -D** command provides information about packets sent and received, as well as sent and received packets dropped, as shown in Example 31-12. This information is provided for the adapter (hardware), the device driver, the demuxer, the protocols, and the network interfaces.

Example 31-12 Output of netstat -d command

```

# netstat -D

```

Source	Ipkts	Opkts	Idrops	Odrops
ent_dev0	127506	92275	0	0
fddi_dev0	0	0	0	0
tok_dev0	1943877	182098	0	0

Devices Total	2071383	274373	0	0

ent_dd0	127506	92275	0	0
fddi_dd0	0	0	0	0
tok_dd0	1943877	182098	0	0

Drivers Total	2071383	274373	0	0

ascsi_dmx0	0	N/A	0	N/A

ent_dmx0	127515	N/A	0	N/A
fddi_dmx0	0	N/A	0	N/A
tok_dmx0	1613875	N/A	330002	N/A

Demuxer Total	1741390	N/A	330002	N/A

IP	1948847	1866539	23107	23087
TCP	1084942	969481	0	0
UDP	559382	178335	44	0

Protocols Total	3593171	3014355	23151	23087

lo_if0	262	348	86	0
en_if0	127515	92283	0	0
tr_if0	1996638	228176	0	0
css_if0	1153430	1711417	0	3

Net IF Total	3277845	2032224	86	3

NFS/RPC Total	N/A	909	0	0

(Note: N/A -> Not Applicable)				

The **netstat -D** command provides an overview of the received packets in the `Ipkts` column and sent packets in the `Opkts` column for the different network layers. These can be used to get an idea about the usage of each network layer. Balancing the load on different adapters may improve performance. The previous example shows that the FDDI Adapter is not used at all. Moving some of the load currently on the token-ring adapter to this FDDI adapter would be a good idea.

The `Idrops` and `Odrops` columns show the dropped packets. There is always a reason for packets to be dropped. On the device level a shortage of mbufs can cause these drops. Drops on the demux level indicate packets of an unsupported protocol, such as IPX, are sent to the system. They cannot be processed and are discarded. However, these packets will cost performance because they are received by the adapter and passed to the device driver using mbufs and CPU time. The **iptrace** command can be used to identify the source of such packets. Further actions can be taken on the source system sending these packets to reduce their number or stop them from being sent.

Dropped packets on the protocol layer should be taken care of by using the **netstat -p Protocol** command to get more information about these dropped packets. Refer to “Statistics for each protocol” on page 632.

To display the state of all sockets

The `netstat -an` command provides information about all active connections including the protocol, local and foreign address, state of the connection, and size of the receive and send queues. Example 31-13 shows an example for the `netstat -an` command.

Example 31-13 Output of netstat -an command

```
# netstat -an
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address      Foreign Address    (state)
tcp4   0      2 9.49.7.84.23      9.3.9.165.36291   ESTABLISHED
tcp4   0      0 127.0.0.1.199     127.0.0.1.32860   ESTABLISHED
tcp4   0      0 127.0.0.1.32860   127.0.0.1.199     ESTABLISHED
tcp4   0      0 *.6680            *.*               LISTEN
tcp4   0      0 *.2401            *.*               LISTEN
tcp4   0      0 *.32803           *.*               LISTEN
tcp4   0      0 127.0.0.1.199     127.0.0.1.32802   ESTABLISHED
tcp4   0      0 127.0.0.1.32802   127.0.0.1.199     ESTABLISHED
tcp4   0      0 *.199             *.*               LISTEN
tcp4   0      0 *.12865           *.*               LISTEN
tcp4   0      0 *.6681            *.*               LISTEN
tcp4   0 17424 9.3.9.165.37773   9.3.9.165.20      ESTABLISHED
tcp  14520 0 9.3.9.165.20      9.3.9.165.37773   ESTABLISHED
tcp4   0 0 9.3.9.165.21      9.3.9.165.37772   ESTABLISHED
tcp   0 0 9.3.9.165.37772   9.3.9.165.21      ESTABLISHED
tcp4   0      0 *.37              *.*               LISTEN
tcp4   0      0 *.13              *.*               LISTEN
tcp4   0      0 *.19              *.*               LISTEN
tcp4   0      0 *.9               *.*               LISTEN
tcp4   0      0 *.7               *.*               LISTEN
tcp   0      0 *.512             *.*               LISTEN
tcp4   0      0 *.543             *.*               LISTEN
tcp   0      0 *.513             *.*               LISTEN
tcp4   0      0 *.544             *.*               LISTEN
tcp   0      0 *.514             *.*               LISTEN
tcp   0      0 *.23              *.*               LISTEN
tcp   0      0 *.21              *.*               LISTEN
tcp4   0      0 *.32772           *.*               LISTEN
tcp4   0      0 *.905             *.*               LISTEN
tcp4   0      0 *.904             *.*               LISTEN
tcp4   0      0 *.111             *.*               LISTEN
tcp4   0      0 *.25              *.*               LISTEN
udp4   0      0 *.514             *.*               LISTEN
udp4   0      0 *.10002           *.*               LISTEN
udp4   0      0 *.10001           *.*               LISTEN
udp4   0      0 *.10000           *.*               LISTEN
udp4   0      0 9.49.7.84.123    *.*               LISTEN
udp4   0      0 9.49.59.163.123  *.*               LISTEN
```

... lines omitted ...

Active UNIX domain sockets

SADR/PCB	Type	Recv-Q	Send-Q	Inode	Conn	Refs	Nextref	Addr
700edc00	dgram	0	0	137ac720	0	0	0	/dev/.SRC-unix/SRCgCedEo
7004d300								
7004e600	dgram	0	0	13af1d20	0	7004dbc0	0	/dev/log
7004d100								
70090c00	dgram	0	0	14d73d00	0	0	0	/dev/.SRC-unix/SRCb.edEb
7004df40								
70098000	dgram	0	0	1428cd20	0	0	0	/dev/.SRC-unix/SRCN1edEg
7004ddc0								
70098c00	dgram	0	0	0	0	0	0	
7004de00								
700d4800	stream	0	0	14919fa0	0	0	0	/var/ha/soc/em.c1srv.cws3et
7004d780								
70090e00	dgram	0	0	0	0	0	0	
7004df80								
700c1c00	dgram	0	0	14917660	0	0	0	/dev/.SRC-unix/SRCqQedEd
7004dd80								
7004e800	dgram	0	0	151d03a0	0	0	0	/dev/SRC
7007f040								
700c1e00	dgram	0	0	0	0	0	0	
7004de40								
700d4600	stream	0	0	0	7004d6c0	0	0	
7004d700								
700c1600	dgram	0	0	13231dc0	0	0	0	/dev/.SRC-unix/SRCvMedEe
7004dd00								

... lines omitted ...

The data provided by the `netstat -an` command is very useful for determining connection problems. For performance monitoring, the number of connections and the sizes of the receive and send queues are of interest. The number of established connections, for example to port 80 on a system running a WEB server, shows the current number of clients accessing this service.

The send and receive queue sizes are an indication of the current use of a connection. In Example 31-13 on page 640, the `ftp` command to our own token-ring adapter is running performing a `put "|dd if=/dev/zero bs=64k count=100000" /dev/null`. So both the sending and receiving side of the connection are on the system. The send and receive queues for the ftp data connection using port 20 are filled, data is sent out and received. Repeated runs of the `netstat -an` command may indicate stuck transmissions on connections if the send queue stays at the same size. The receiving system should be inspected to check the state of the connection there.

The network buffer cache

The **netstat -c** command provides statistics about the network buffer cache (NBC) usage. Example 31-14 shows the output of the **netstat -c** command.

Example 31-14 Output of netstat -c command

```
Network Buffer Cache Statistics:
-----
Current total cache buffer size: 756389056
Maximum total cache buffer size: 756389056
Current total cache data size: 636761915
Maximum total cache data size: 636761915
Current number of cache: 100016
Maximum number of cache: 100016
Number of cache with data: 100016
Number of searches in cache: 400113
Number of cache hit: 16
Number of cache miss: 200038
Number of cache newly added: 100016
Number of cache updated: 0
Number of cache removed: 0
Number of successful cache accesses: 100032
Number of unsuccessful cache accesses: 100022
Number of cache validation: 0
Current total cache data size in private segments: 1438760235
Maximum total cache data size in private segments: 1438760235
Current total number of private segments: 20000
Maximum total number of private segments: 20000
Current number of free private segments: 0
Current total NBC_NAMED_FILE entries: 100022
Maximum total NBC_NAMED_FILE entries: 100022
```

This example shows a NBC that is mostly written to, without many cache hits reported. The number of newly added files to the cache are equal to the number of total files in the cache. The reason could be an application just started using the NBC. However, the cache hit count should go up soon. Or the cache is too small for the application. The NBC is used by the `send_file()` system call if the `SF_SYNC_CACHE` flag is set. It is also used by the FRCA. If neither of these is used on a system, the values in the **netstat -c** output are 0 (zero).

The network options to control the NBC are:

nbc_limit Specifies the total maximum amount of memory in kilobytes that can be used for the NBC. The default value is derived from thewall. When the cache grows to this limit, the least-used cache objects are flushed out of cache to make room for the new ones.

- nbc_max_cache** Specifies the maximum size of the cache object allowed in the NBC without using the private segments in number of bytes, the default being 131,072 (128K) bytes. A data object bigger than this size is either cached in a private segment or is not cached at all.
- nbc_min_cache** Specifies the minimum size of the cache object allowed in the NBC in number of bytes, the default being one byte. A data object smaller than this size is not put into the NBC.
- nbc_pseg** Specifies the maximum number of private segments that can be created for the NBC. The default value is 0. When this option is set at a non-zero value, a data object between the size specified in `nbc_max_cache` and the segment size (256 MB) is cached in a private segment. A data object bigger than the segment size is not cached at all. When the maximum number of private segments exist, cache data in private segments may be flushed for new cache data so that the number of private segments do not exceed the limit. When `nbc_pseg` is set to zero, all caches in private segments are flushed.
- nbc_pseg_limit** Specifies the maximum amount of cached data allowed in private segments in the NBC in kilobytes. The default value is half of the total real memory size on the running system. Because data cached in private segments are pinned by the NBC, `nbc_pseg_limit` controls the amount of pinned memory used for the NBC in addition to the network buffers in global segments. When the amount of cached data reaches this limit, cache data in private segments may be flushed for new cache data so that the total pinned memory size does not exceed the limit. When `nbc_pseg_limit` is set to zero, all caches in private segments are flushed.

The `nfso` command

The `nfso` command enables the configuration of Network File System (NFS) variables and removal of file locks from NFS client systems on the server. Prior to changing NFS variables to tune NFS performance, monitor the load on the system using the `nfsstat`, `netstat`, `vmstat`, and `iostat` commands.

The `nfso` command resides in `/usr/sbin` and is part of the `bos.net.nfs.client` fileset, which is installable from the AIX base installation media.

32.1 nfsso

The syntax of the **nfsso** command is:

```
nfsso [ -p | -r ] [ -c ] { -o Tunable[ =Newvalue ] }  
nfsso [ -p | -r ] { -d Tunable }  
nfsso [ -p | -r ] -D  
nfsso [ -p | -r ] -a [ -c ]  
nfsso -?  
nfsso -h Tunable  
nfsso -l Hostname  
nfsso [ -c ]  
nfsso -L [ Tunable ]  
nfsso -x [ Tunable ]
```

Multiple flags -o, -d, and -L are allowed.

Flags

- a** Displays the current, reboot (when used in conjunction with -r) or permanent (when used in conjunction with -p) value for all tunable parameters, one perline in pairs Tunable = Value. For the permanent options, a value is only displayed for a parameter if its reboot and current values are equal. Otherwise NONE displays as the value.
- c** Changes the output format of the nfsso command to colon-delimited format.
- d Tunable** Sets the Tunable variable back to its default value. If a tunable needs to be changed (that is, it is not set to its default value) and is of type Bosboot or Reboot, or if it is of type Incremental and has been changed from its default value, and -r is not used in combination, it will not be changed, and a warning displays instead.
- D** Sets all tunable variables back to their default value. If tunables needing to be changed are of type Bosboot or Reboot, or are of type Incremental and have been changed from their default value, and -r is not used in combination, they will not be changed but warnings display instead.
- h Tunable** Displays help about the Tunable parameter.
- l HostName** Enables a system administrator to release NFS file locks on an NFS server. The HostName variable specifies the host name of the NFS client that has file locks held at the NFS server. The nfsso -l command makes a remote

procedure call to the NFS server's rpc.lockd network lock manager to request the release of the file locks held by the HostName NFS client.

If an NFS client has file locks held at the NFS server and has been disconnected from the network and cannot be recovered, the `nfso -l` command can be used to release those locks so that other NFS clients can obtain similar file locks for their purposes. Note that the `nfso` command can be used to release locks only on the local NFS server.

-o Tunable[=NewValue]

Displays the value or sets Tunable to NewValue. If a tunable needs to be changed (the specified value is different from the current value), and is of type Bosboot or Reboot, or if it is of type Incremental and its current value is bigger than the specified value, and `-r` is not used in combination, it will not be changed but a warning displays instead. When `-r` is used in combination without a new value, the nextboot value for the tunable displays. When `-p` is used in combination without a NewValue, a value displays only if the current and next boot values for the tunable are the same. Otherwise NONE displays as the value.

-p

Makes changes apply to both current and reboot values, when used in combination with `-o`, `-d`, or `-D`; that is, it turns on the updating of the `/etc/tunables/nextboot` file in addition to the updating of the current value. These combinations cannot be used on Reboot and Bosboot type parameters because their current value cannot be changed.

When used with `-a` or `-o` without specifying a new value, values are displayed only if the current and next boot values for a parameter are the same. Otherwise NONE displays as the value.

-r

Makes changes apply to reboot values when used with `-o`, `-d`, or `-D`; that is, it turns on the updating of the `/etc/tunables/nextboot` file. If any parameter of type Bosboot is changed, the user is prompted to run `bosboot`.

When used with `-a` or `-o` without specifying a new value, next boot values for tunables display instead of current values.

-L [Tunable]

Lists the characteristics of one or all tunables, one per line.

- x [tunable] Generates tunable characteristics in a comma-separated format for loading into a spreadsheet.
- ? Displays the usage statement.

Pre 5.2 compatibility mode considerations.

Pre 5.2 compatibility mode is controlled by the `pre520tune` attribute of `sys0`; see Introduction to AIX 5L Version 5.2 tunable parameter setting in the *AIX 5L Version 5.2 Performance Management Guide*. When running in pre 5.2 compatibility mode, reboot values for parameters, except those of type `Bosboot`, are not really meaningful because in this mode they are not applied at boot time.

In pre 5.2 compatibility mode, setting reboot values to tuning parameters continues to be achieved by imbedding calls to tuning commands in scripts called during the boot sequence. Parameters of type `Reboot` can therefore be set without the `-r` flag, so that existing scripts continue to work.

This mode is automatically turned ON when a machine is MIGRATED to AIX 5L Version 5.2. For complete installations, it is turned OFF and the reboot values for parameters are set by applying the content of the `/etc/tunables/nextboot` file during the reboot sequence. Only in that mode are the `-r` and `-p` flags fully functional. See AIX 5L Version 5.2 kernel tuning in the Performance Tools Guide and Reference for details about the new 5.2 mode.

32.1.1 Information about measurement and sampling

The `nfso` command reads the NFS network variables from kernel memory and writes changes to kernel memory of the running system. The values not equal to the default values must be set after each system start. This can be done by adding the necessary `nfso` variable values into the `/etc/tunables/nextboot` file. Most changes performed by `nfso` take effect immediately.

32.2 Examples for nfso

This section shows some examples of the `nfso` command.

32.2.1 Listing all of the tunables and their current values

Example 32-1 on page 649 uses the `nfso -a` command to display the current NFS network variables. This command should always be used to display and store the current setting prior changing them.

Example 32-1 Display and store in a file the current NFS network variables

```
# nfsd -a
    portcheck = 0
    udpchecksum = 1
    nfs_socketsize = 600000
    nfs_tcp_socketsize = 600000
    nfs_setattr_error = 0
    nfs_gather_threshold = 4096
    nfs_repeat_messages = 0
nfs_udp_duplicate_cache_size = 5000
nfs_tcp_duplicate_cache_size = 5000
    nfs_server_base_priority = 0
    nfs_dynamic_retrans = 1
    nfs_iopace_pages = 0
    nfs_max_connections = 0
    nfs_max_threads = 3891
    nfs_use_reserved_ports = 0
nfs_device_specific_bufs = 1
    nfs_server_clread = 1
    nfs_rfc1323 = 0
    nfs_max_write_size = 32768
    nfs_max_read_size = 32768
nfs_allow_all_signals = 0
    nfs_v2_pdts = 1
    nfs_v3_pdts = 1
    nfs_v2_vm_bufs = 1000
    nfs_v3_vm_bufs = 1000
nfs_securenfs_authtimeout = 0
nfs_v3_server_readdirplus = 1
: nfsd: 1831-731 cannot contact local lockd.
    lockd_debug_level = -1
: nfsd: 1831-730 cannot contact local statd.
    statd_debug_level = -1
: nfsd: 1831-730 cannot contact local statd.
    statd_max_threads = -1
```

32.2.2 Displaying characteristics of all tunables

Example 32-2 displays the output when using the `nfsd` command with the `-L` flag to display all of the variables and their characteristics.

Example 32-2 Listing of nfsd tunables

```
# nfsd -L
NAME                                CUR  DEF  BOOT  MIN  MAX  UNIT  TYPE
DEPENDENCIES
-----
portcheck                            0    0    0    0    1   On/Off  D
```

udpchecksum	1	1	1	0	1	On/Off	D
nfs_socketsize	600000	600000	600000	40000	1M	Bytes	D
nfs_tcp_socketsize	600000	600000	600000	40000	1M	Bytes	D
nfs_setattr_error	0	0	0	0	1	On/Off	D
nfs_gather_threshold	4K	4K	4K	512	8193	Bytes	D
nfs_repeat_messages	0	0	0	0	1	On/Off	D
nfs_udp_duplicate_cache_size	5000	5000	5000	5000	100000	Req	I
nfs_tcp_duplicate_cache_size	5000	5000	5000	5000	100000	Req	I
nfs_server_base_priority	0	0	0	31	125	Pri	D
nfs_dynamic_retrans	1	1	1	0	1	On/Off	D
nfs_iopace_pages	0	0	0	0	65535	Pages	D
nfs_max_connections	0	0	0	0	10000	Number	D
nfs_max_threads	3891	3891	3891	5	3891	Threads	D
nfs_use_reserved_ports	0	0	0	0	1	On/Off	D
nfs_device_specific_bufs	1	1	1	0	1	On/Off	D
nfs_server_clread	1	1	1	0	1	On/Off	D
nfs_rfc1323	0	0	0	0	1	On/Off	D
nfs_max_write_size	32K	32K	32K	512	64K	Bytes	D
nfs_max_read_size	32K	32K	32K	512	64K	Bytes	D
nfs_allow_all_signals	0	0	0	0	1	On/Off	D
nfs_v2_pdots	1	1	1	1	8	PDTs	M
nfs_v3_pdots	1	1	1	1	8	PDTs	M
nfs_v2_vm_bufs	1000	1000	1000	512	5000	Bufs	I

nfs_v3_vm_bufs	1000	1000	1000	512	5000	Bufs	I
nfs_securenfs_authtimeout	0	0	0	0	60	Seconds	D
nfs_v3_server_readdirplus	1	1	1	0	1	On/Off	D
lockd_debug_level	-1	0	0	0	10	Level	D
statd_debug_level	-1	0	0	0	10	Level	D
statd_max_threads	-1	50	50	1	1000	Threads	D

n/a means parameter not supported by the current platform or kernel

Parameter types:

- S = Static: cannot be changed
- D = Dynamic: can be freely changed
- B = Bosboot: can only be changed using bosboot and reboot
- R = Reboot: can only be changed during reboot
- C = Connect: changes are only effective for future socket connections
- M = Mount: changes are only effective for future mountings
- I = Incremental: can only be incremented

Value conventions:

- K = Kilo: 2^{10}
- M = Mega: 2^{20}
- G = Giga: 2^{30}
- T = Tera: 2^{40}
- P = Peta: 2^{50}
- E = Exa: 2^{60}

Any change (with -o, -d, or -D) to a Mount parameter results in a message warning the user that the change is only effective for future mountings. Any attempt to change (with -o, -d, or -D but without -r) the current value of a parameter of type Incremental with a new value smaller than the current value results in an error message.

32.2.3 Displaying and changing a tunable with the nfs command

Example 32-3 displays the value of the `nfs_dynamic_retrans` variable by using the `-o` flag, which can also be used to change a variable by assigning it to a specific value.

Example 32-3 Displaying and changing a tunable

```
# nfs -o nfs_dynamic_retrans
nfs_dynamic_retrans= 1
# nfs -o nfs_dynamic_retrans=0
Setting nfs_dynamic_retrans to 0
```

```
# nfsd -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0
```

32.2.4 Resetting a tunable value to its default

Example 32-4 shows that a value was changed in Example 32-3 on page 651 can be reset to the default by using the `-d` flag.

Example 32-4 Restoring default tunable value

```
# nfsd -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0
# nfsd -d nfs_dynamic_retrans
Setting nfs_dynamic_retrans to 1
# nfsd -o nfs_dynamic_retrans
nfs_dynamic_retrans= 1
```

32.2.5 Displaying help information about a tunable

Using the `-h` flag with the `nfsd` command displays information about that specific variable, as shown in Example 32-5.

Example 32-5 Getting information about a tunable

```
# nfsd -h nfs_dynamic_retrans
nfs_dynamic_retrans: Specifies whether the NFS client should use a dynamic
retransmission algorithm to decide when to resend NFS requests to the server.
Default : 1; Range: 0 or 1. If this function is turned on, the timeout parameter
is only used in the first retransmission. With this parameter set to 1, the NFS
client will attempt to adjust its timeout behavior based on past NFS server
response. This allows for a floating timeout value along with adjusting the
transfer sizes used. All of this is done based on an accumulative history of
the NFS server's response time. In most cases, this parameter does not need to
be adjusted. There are some instances where the straightforward timeout
behavior is desired for the NFS client. In these cases, the value should be set
to 0 before mounting file systems.
```

32.2.6 Permanently changing an nfsd tunable

When using the `-p` flag, permanent changes are made to a variable. It changes the current value and makes an entry into the `/etc/tunables/nextboot` file.

Example 32-6 on page 653 displays the contents of the `/etc/tunables/nextboot` file with no information about the `nfs_dynamic_retrans` variable. Then by executing `nfsd -p` with the `-o` flag to change the `nfs_dynamic_retrans` variable, a line was added to `/etc/tunables/nextboot` file. This ensures that the variable is defined for each reboot. It also changed the current value of the variable.

Example 32-6 Permanently changing the nextboot file

```
# nfsso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 1
# cat /etc/tunables/nextboot
ioo:
nfsso:
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"
vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"

# nfsso -p -o nfs_dynamic_retrans=0
# cat /etc/tunables/nextboot
ioo:
nfsso:
    nfs_dynamic_retrans = "0"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"
vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"
# nfsso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0
```

32.2.7 Changing a tunable after reboot

By using the `-r` flag the change to a variable will only take effect after a reboot. In Example 32-7 we used the `nfsso` command with the `-r` flag to have the variable change after the reboot. First we displayed the value of the `nfs_dyanmic_retrans` variable, which is set to 0, as it is in `/etc/tunables/nextboot`. We then ran `nfsso` with the `-r` flag. The current value of the variable has not changed, but the contents of the `/etc/tunables/nextboot` have been updated.

Example 32-7 Changing a parameter after next reboot

```
# nfsso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0

# cat /etc/tunables/nextboot
ioo:
nfsso:
    nfs_dynamic_retrans = "0"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"
vmo:
```

```
maxperm% = "50"
maxclient% = "50"
spec_dataseg_int = "0"

# nfsd -ro nfs_dynamic_rtrans=1

# nfsd -o nfs_dynamic_rtrans
nfs_dynamic_rtrans=0

# cat /etc/tunables/nextboot
ioo:
nfsd:
    nfs_dynamic_retrans = "1"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"
vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"
```

The `nfsstat` command

The `nfsstat` command displays statistics about the Network File System (NFS) and the Remote Procedure Call (RPC) interface to the kernel. You can also use the `nfsstat` command to reinitialize this information.

The `nfsstat` command is a monitoring tool. Its output data can be used for problem determination and performance tuning. The `nfsstat` command resides in `/usr/sbin` and is part of the `bos.net.nfs.client` fileset, which is installable from the AIX base installation media.

33.1 nfsstat

The syntax of the **nfsstat** command is:

```
/usr/sbin/nfsstat [ -c ] [ -s ] [ -n ] [ -r ] [ -z ] [ -m ]
```

Flags

- c** Displays client information. Only the client side NFS and RPC information is printed. Enables the user to limit the report to client data. The **nfsstat** command provides information about the number of RPC and NFS calls sent and rejected by the client. To print only client NFS or RPC information, combine this flag with the **-n** or **-r** option.
- m** Displays statistics for each NFS file system mounted along with the server name and address, mount flags, current read and write sizes, retransmission count, and the timers used for dynamic retransmission.
- n** Displays NFS information. Prints NFS information for both the client and server. To print only the NFS client or server information, combine this flag with the **-c** and **-s** options.
- r** Displays RPC information.
- s** Displays server information.
- z** Resets statistics. This flag is for use by the root user only and can be combined with any of the flags above to zero-out particular sets of statistics after printing them.

33.1.1 Information about measurement and sampling

The **nfsstat** command reads out statistic information collected by the NFS client and the NFS server kernel extensions. This read is done at **nfsstat** command execution time. The **nfsstat -z** command is used to reset the statistics. For details about the data structures used, refer to the system header files `/usr/include/nfs/nfs_fsctl.h` and `/usr/include/rpc/svc.h`.

The **nfsstat** command displays server and client statistics for both RPC and NFS. The **-s** (server), **-c** (client), **-r** (RPC), and **-n** (NFS) flags can be used to display only a subset of all data.

The RPC statistics output consists of two parts: the first shows the statistics for connection-oriented TCP RPC, the second shows the statistics for connectionless User Datagram Protocol (UDP) RPC. The NFS statistics output is also divided into two parts: the first shows the NFS version 2 statistics, and the second shows the NFS version 3 statistics. The RPC statistics are useful for detecting performance problems caused by time-outs and retransmissions. The

NFS statistics show the usage count of file system operations, such as read(), write(), and getattr(). These values show how the file system is used. This can help to decide which tuning actions to perform to improve performance. The `nfsstat` command can display information about each mounted file system.

33.2 Examples for nfsstat

In this section we take a closer look at each of the statistics `nfsstat` can provide:

- ▶ NFS server RPC statistics - the `nfsstat -sr` command.
- ▶ NFS server NFS statistics - the `nfsstat -sn` command.
- ▶ NFS client RPC statistics - the `netstat -cr` command.
- ▶ NFS client NFS statistics - the `netstat -cn` command.
- ▶ Statistics on mounted file systems - the `nfsstat -m` command

33.2.1 NFS server RPC statistics

The output in Example 33-1 shows the server RPC statistics created using the `nfsstat -sr` command:

Example 33-1 Output of nfsstat -sr

```
# nfsstat -sr
```

Server rpc:

Connection oriented						
calls	badcalls	nullrecv	badlen	xdrcall	dupchecks	dupreqs
31197	0	0	0	0	10085	0

Connectionless						
calls	badcalls	nullrecv	badlen	xdrcall	dupchecks	dupreqs
0	0	0	0	0	0	0

The output shows statistics for both connection-oriented (TCP) and connectionless (UDP) RPC. In this example, NFS used TCP as the transport protocol. The fields in this output are:

- `calls` Total number of RPC calls received from clients.
- `badcalls` Total number of calls rejected by the RPC layer. The rejects happen because of failed authentication. The value should be zero.
- `nullrecv` Number of times a RPC call was not available when it was thought to be received.
- `badlen` Packets truncated or damaged (number of RPC calls with a length shorter than a minimum-sized RPC call). The value should stay at zero. An increasing value may be caused by network problems.

- `xdrca11` Number of RPC calls whose header could not be External Data Representation (XDR) decoded. The value should stay at zero. An increasing value may be caused by network problems.
- `dupchecks` Number of RPC calls that require a look-up in the duplicate request cache. Duplicate checks are performed for operations that cannot be performed twice with the same result. If the first command succeeds but the reply is lost, the client retransmits this request. This retransmitted command will fail. An example of an operation that cannot be performed twice with the same result is the `rm` command. We want duplicate requests like these to succeed, so the duplicate cache is consulted, and, if it is a duplicate request, the same (successful) result is returned on the duplicate request as was generated on the initial request.

These operations apply to duplicate checks: `setattr()`, `write()`, `create()`, `remove()`, `rename()`, `link()`, `symlink()`, `mkdir()`, and `rmdir()`. Any instance of these is stored in the duplicate request cache.

The size of the duplicate request cache is controlled by the NFS options `nfs_tcp_duplicate_cache_size` for the TCP network transport and `nfs_udp_duplicate_cache_size` for the UDP network transport. See Chapter 32, “The `nfso` command” on page 645 for information regarding the NFS options `nfs_tcp_duplicate_cache_size` and `nfs_udp_duplicate_cache_size`.

These NFS options need to be increased on a high volume NFS server. Calculating the NFS operations per second and using four times this value is a good starting point. The `nfsstat -z; sleep 60; nfsstat -sn` command can be used to capture the number of NFS operations per minute.

- `dupreqs` Number of duplicate RPC calls found. This value gets increased each time a duplicate RPC request, using the data from the duplicate request cache, is found. An increasing value for `dupreqs` indicates retransmissions of commands from clients. These retransmissions can be caused by time-outs (the server did not answer in time) or dropped packets on the client receiving side or server sending side. Use the `nfsstat -cr` command to check for time-outs on the NFS clients. Refer to 33.2.3, “NFS client RPC statistics” on page 660 for more information about the `nfsstat -cr` command. Use the `netstat -in`, `netstat -s`, `netstat -v`, and `netstat -m` commands to check for dropped packets on both NFS client and NFS server.

The `nfsstat -zsr; sleep 60; nfsstat -sr` can be used to get the server RPC statistics for one minute and to calculate the per-second values. Doing this on a well-performing NFS sever during normal operation and storing this data will help to verify NFS server load in case this server later shows an NFS performance

problem. The cause for bad performance may be a temporary increased load from one or more NFS clients.

33.2.2 NFS server NFS statistics

The NFS server NFS statistics can be used to determine the type of NFS operation used most on the server. This helps to decide which tuning can be performed to increase NFS server performance. For example, a high percentage of write() calls may require disk and LVM tuning to increase write performance. A high value of read() calls may require more RAM for file caching. There are no rules of thumb, as tuning the NFS server depends on many factors such as:

- ▶ The amount of RAM installed
- ▶ The disk subsystem used
- ▶ The number of CPUs installed
- ▶ The CPU speed of the installed CPUs
- ▶ The number of NFS clients
- ▶ The networks used

Example 33-2 shows the output of the `nfsstat -sn` command.

Example 33-2 Output of `nfsstat -sn` command

```
# nfsstat -sn

Server nfs:
calls      badcalls  public_v2  public_v3
809766     0         0          0
Version 2: (0 calls)
null      getattr   setattr   root      lookup    readlink  read
0 0%      0 0%      0 0%      0 0%      0 0%      0 0%      0 0%
wrcache   write     create    remove    rename    link      symlink
0 0%      0 0%      0 0%      0 0%      0 0%      0 0%      0 0%
mkdir     rmdir    readdir   statfs
0 0%      0 0%      0 0%      0 0%
Version 3: (809765 calls)
null      getattr   setattr   lookup    access    readlink  read
1 0%      133491 16% 558 0%    227155 28% 15397 1%  0 0%      56636 6%
write    create   mkdir     symlink   mknod     remove   rmdir
172511 21% 67425 8% 558 0%    0 0%      0 0%      67486 8% 558 0%
rename    link      readdir   readdir+  fsstat    fsinfo    pathconf
0 0%      0 0%      1023 0%    560 0%    2 0%      0 0%      0 0%
commit
66404 8%
```

This example shows a high usage of write. The reported 21 percent may still be low enough not to worry about. However, the values for create (67425) and remove (67486) are high and equal. This could be an indication of an NFS client

creating a high number of temporary files in the NFS file system. Creating these temporary files in a local file system on the NFS client will reduce the load on the NFS server. The NFS client performance (at least the performance of the application creating the temporary files) will increase as well.

33.2.3 NFS client RPC statistics

The output in Example 33-3 shows the client RPC statistics created using the command `nfsstat -cr`.

Example 33-3 Output of `nfsstat -cr` command

```
# nfsstat -cr

Client rpc:
Connection oriented
calls      badcalls  badxids   timeouts  newcreds  badverfs  timers
1392748    0         0         0         0         0         0
nomem     cantconn  interrupts
0         0         0
Connectionless
calls      badcalls  retrans   badxids   timeouts  newcreds  badverfs
188030    0         13        0         0         0         0
timers    nomem     cantsend
11        0         0
```

The fields in this output are:

- `calls` Total number of RPC calls made to NFS.
- `badcalls` Total number of calls rejected by the RPC layer. The value should be zero.
- `retrans` Number of times a call had to be retransmitted due to a time-out while waiting for a reply from the server. This is applicable only to RPC over connectionless (UDP) transports. The NFS client had to retransmit requests to the NFS server because the NFS server was not responding in time. This could indicate an overloaded server, dropped packets on the server, or dropped packets on the client. Running the `vmstat` and `iostat` commands on the server should show the load on the server. (Refer to Chapter 13, “The `vmstat` command” on page 211 and Chapter 4, “The `iostat` command” on page 81 for details on these commands.) Use the `netstat -in`, `netstat -s`, `netstat -v`, and `netstat -m` commands on the server and client to check for dropped packets. (Refer to Chapter 31, “The `netstat` command” on page 619 for more information.)

Dropped packets on the server could be caused by an overrun of the network adapter transmit queue or a UDP socket buffer overflow. Tuning the NFS option `nfs_socketsize` using the `nfs` command in case of socket buffer overflows is required. Refer to Chapter 32, “The `nfs` command” on page 645 for more information about the `nfs` command.

<code>badxid</code>	Number of times a reply from a server was received that did not correspond to any outstanding call. This means the server is taking too long to reply. Refer to the description for the <code>retrans</code> field.
<code>timeouts</code>	Number of times a call timed-out while waiting for a reply from the server. The same as for the <code>retrans</code> value applies. Refer to the description in for the <code>retrans</code> field. Increasing the NFS mount option <code>timeo</code> by using the <code>smitty chnfsmnt</code> command should reduce the NFS client requests that time out and are retransmitted. This reduces the load on the server because the number of retransmitted requests decreases. However, the performance improvement on the client is not very high. If dynamic retransmission is used, the <code>timeo</code> value is only used for the first retransmission timeout. Refer to 33.2.5, “Statistics on mounted file systems” on page 662 for more details.
<code>newcreds</code>	Number of times authentication information had to be refreshed.
<code>badverfs</code>	Number of times a call failed due to a bad verifier in the response.
<code>timers</code>	Number of times the calculated time-out value was greater than or equal to the minimum specified time-out value for a call.
<code>nomem</code>	Number of times a call failed due to a failure to allocate memory.
<code>cantconn</code>	Number of times a call failed due to a failure to make a connection to the server.
<code>interrupts</code>	Number of times a call was interrupted by a signal before completing.
<code>cantsend</code>	Number of times a send failed due to a failure to make a connection to the client.

33.2.4 NFS client NFS statistics

These statistics show the NFS clients’ usage for the various NFS calls. This information can help in deciding the next steps to perform to increase performance. Example 33-4 on page 662 was taken on the NFS client at the same time the NFS Server Example 33-2 on page 659 was produced.

Example 33-4 Output of `nfsstat -cn` command

```
# nfsstat -cn

Client nfs:
calls      badcalls  clgets    cltoomany
1584182    0          0          0
Version 2: (188425 calls)
null       getattr   setattr   root       lookup     readlink   read
0 0%      95392 50% 0 0%      0 0%      11740 6%  0 0%      81068 43%
wrcache    write     create    remove     rename     link       symlink
0 0%      0 0%     0 0%      0 0%      0 0%      0 0%      0 0%
mkdir      rmdir     readdir   statfs
0 0%      0 0%     223 0%    2 0%
Version 3: (1399306 calls)
null       getattr   setattr   lookup     access     readlink   read
0 0%      230820 16% 966 0%    393221 28% 26634 1%  0 0%      97536 6%
write     create   mkdir     symlink    mknod      remove    rmdir
296985 21% 116725 8% 966 0%    0 0%      0 0%      116786 8% 966 0%
rename     link      readdir   readdir+   fsstat     fsinfo     pathconf
0 0%      0 0%     1771 0%   968 0%    4 0%      0 0%      0 0%
commit
114958 8%
```

Refer to 33.2.2, “NFS server NFS statistics” on page 659 for more information and use of this statistic. The NFS clients `nfsstat -cn` example above shows the same high count for file create and file remove as the server side in Example 33-2 on page 659. There could be an application running, creating temporary files in a NFS mounted file system. Moving these temporary files off of NFS to a local file system will increase performance on this NFS client and reduce load on the NFS server.

33.2.5 Statistics on mounted file systems

The `nfsstat -m` command displays statistics for each NFS mounted file system on an NFS client system. This includes:

- ▶ Name of the file system
- ▶ Name of the server serving the file system
- ▶ Flags used to mount the file system
- ▶ Current timers used for dynamic retransmission

Example 33-5 is an example of the `nfsstat -m` output.

Example 33-5 Output of `nfsstat -m` command

```
# nfsstat -m

/server1 from /server1:server1.itso.ibm.com
```

Flags:

```
vers=2,proto=udp,auth=unix,hard,intr,dynamic,rsize=8192,wsiz=8192,retrans=5  
Lookups: srtt=7 (17ms), dev=3 (15ms), cur=2 (40ms)  
Reads:   srtt=47 (117ms), dev=4 (20ms), cur=7 (140ms)  
All:     srtt=10 (25ms), dev=7 (35ms), cur=4 (80ms)
```

This example shows one NFS file system mounted over /server1. The NFS server serving this file system is server1.itso.ibm.com, and the directory name on the server is /system1.

Flags	The flags used to mount the NFS file system. Refer to the mount command in <i>AIX 5L Version 5.2 Commands Reference</i> , SBOF-1877 for more information.
srtt	Smoothed round-trip time.
dev	Estimated deviation.
cur	Current backed-off time-out value.

The current timers used for dynamic retransmission are the numbers in parentheses in the example output. These are the actual times in milliseconds. Response times are shown for lookups, reads, writes, and a combination of all operations (All). There was no write to this NFS file system, and so no response time values are shown for this function.

The dynamic retransmission can be turned off using the NFS option `nfs_dynamic_retrans`. Refer to Chapter 32, "The `nfso` command" on page 645 for more information. The default in AIX is that dynamic retransmission is used.

Archived

The no command

The Network Options (**no**) command is used to set the network attributes. The **no** command can either display the network parameters or change them in the kernel. It can also set a parameter back to its default value.

The **no** command resides in `/usr/sbin` and is part of the `bos.net.tcp.client` fileset, which is installable from the AIX installation media.

Note: The **no** parameters are not saved, and so are lost on a reboot. To ensure that the changes are permanent, add them to the `/etc/tunables/nextboot` file.

34.1 no

The syntax of the **no** command is:

```
no [ -p | -r ] { -o Tunable[=NewValue] }
no [ -p | -r ] { -d Tunable }
no [ -p | -r ] { -D }
no -a
no -?
no -h [ Tunable ]
no -L [ Tunable ]
no -x [ Tunable ]
```

Flags

- a** Displays current, reboot (when used in conjunction with -r), or permanent (when used in conjunction with -p) value for all tunable parameters, one per line in pairs Tunable = Value. For the permanent options, a value only displays for a parameter if its reboot and current values are equal. Otherwise NONE displays as the value.
- d Tunable** Resets Tunable its to default value. If Tunable needs to be changed (that is, it is not set to its default value) and it is of type Bosboot or Reboot, or if it is of type Incremental and has been changed from its default value and -r is not used in combination, it is not changed but a warning displays instead.
- D** Resets all tunables to their default value. If a tunable needing to be changed are of type Bosboot or Reboot, or if they are of type Incremental and have been changed from their default value, and neither -p nor -r are used in combination, they will not be changed but a warning displayd instead.
- h** Tunable Displays help about Tunable parameters.
- o Tunable[=NewValue]** Displays the value or sets the Tunable to NewValue. If a tunable needs to be changed (the specified value is different from the current value), and is of type Bosboot or Reboot, or if it is of type Incremental and its current value is bigger than the specified value, and -r is not used in combination, it is not changed but a warning displays instead.
- When -r is used in combination without a new value, the nextboot value for Tunable is displayed. When -p is used

in combination without a new value, a value displays only if the current and next boot values for tunable are the same. Otherwise NONE displays as the value.

- p** Makes changes apply to both current and reboot values when used in combination with -o, -d, or -D (turns on updating of the /etc/tunables/nextboot file in addition to the updating of the current value). These combinations cannot be used on Reboot and Bosboot type parameters because their current value cannot be changed.

When used with -a or -o without specifying a new value, values displays only if the current and next boot values for a parameter are the same. Otherwise NONE displays as the value.
- r** Makes changes apply to reboot values when used in combination with -o, -d, or -D (turns on the updating of the /etc/tunables/nextboot file). If any parameter of type Bosboot is changed, the user is prompted to run bosboot. When used with -a or -o without specifying a new value, next boot values for tunables display instead of the current values.
- L** Lists the characteristics of one or all tunables, one per line.
- x [tunable]** Generates tunable characteristics in a comma-separated format for loading into a spreadsheet.

Note: When using the -o flag do not enter space characters before or after the equal sign. If you do, the command will fail.

34.2 Examples for no

The output from the **no -a** command displays all of the **no** parameters, as seen in Example 34-1.

Example 34-1 The no -a command displays the network tunables and their values

```
# no -a
arpqsize = 12
arpt_killc = 20
arptab_bsiz = 7
arptab_nb = 73
bcastping = 0
clean_partial_conns = 0
delayack = 0
```

```
    delayackports = {}
    dgd_packets_lost = 3
    dgd_ping_time = 5
    dgd_retry_time = 5
    directed_broadcast = 0
    extendednetstats = 0
    fasttimo = 200
    icmp6_errmsg_rate = 10
    icmpaddressmask = 0
    ie5_old_multicast_mapping = 0
    ifsize = 256
    inet_stack_size = 16
    ip6_defttl = 64
    ip6_prune = 2
    ip6forwarding = 0
    ip6srcrouteforward = 1
    ipforwarding = 0
    ipfragttl = 60
    ipignoreredirects = 0
    ipqmaxlen = 100
    ipsendredirects = 1
    ipsrcrouteforward = 1
    ipsrcrouterecv = 0
    ipsrcroutesend = 1
    llsleep_timeout = 3
    lowthresh = 90
    main_if6 = 0
    main_site6 = 0
    maxnip6q = 20
    maxttl = 255
    medthresh = 95
    multi_homed = 1
    nbc_limit = 0
    nbc_max_cache = 0
    nbc_min_cache = 0
    nbc_pseg = 0
    nbc_pseg_limit = 131072
    ndp_mmaxtries = 3
    ndp_umaxtries = 3
    ndpqsize = 50
    ndpt_down = 3
    ndpt_keep = 120
    ndpt_probe = 5
    ndpt_reachable = 30
    ndpt_retrans = 1
    net_malloc_police = 0
    nonlosrcroute = 0
    nstrpush = 8
    passive_dgd = 0
```

```
pmtu_default_age = 10
pmtu_rediscover_interval = 30
  psebufcalls = 20
  psecache = 1
  pseintrstack = 12288
  psetimers = 20
  rfc1122addrchk = 0
  rfc1323 = 0
  rfc2414 = 0
  route_expire = 1
  routerevalidate = 0
  rto_high = 64
  rto_length = 13
  rto_limit = 7
  rto_low = 1
  sack = 0
  sb_max = 1048576
send_file_duration = 300
  site6_index = 0
  sockthresh = 85
  sodebug = 0
  somaxconn = 1024
  strctlsz = 1024
  strmsgsz = 0
  strthresh = 85
  strturncnt = 15
  subnetsarelocal = 1
tcp_bad_port_limit = 0
  tcp_ecn = 0
tcp_ephemeral_high = 65535
tcp_ephemeral_low = 32768
  tcp_finwait2 = 1200
  tcp_init_window = 0
tcp_inpcb_hashtab_siz = 24499
  tcp_keepcnt = 8
  tcp_keepidle = 14400
  tcp_keepinit = 150
  tcp_keepintvl = 150
tcp_limited_transmit = 1
  tcp_maxburst = 0
  tcp_mssdf1t = 512
tcp_nagle_limit = 65535
  tcp_ndebug = 100
  tcp_newreno = 1
  tcp_nodelayack = 0
tcp_pmtu_discover = 1
  tcp_recvspace = 16384
  tcp_sendspace = 16384
  tcp_timewait = 1
```

```
tcp_ttl = 60
thewall = 131072
udp_bad_port_limit = 0
udp_ephemeral_high = 65535
udp_ephemeral_low = 32768
udp_inpcb_hashtab_siz = 24499
udp_pmtu_discover = 1
udp_recvspace = 42080
udp_sendspace = 9216
udp_ttl = 30
udpcksum = 1
use_isno = 1
```

You can use the command in Example 34-2 to change a network attribute with the **no** command.

Example 34-2 Using the no command to change network parameters

```
# no -o tcp_recvspace
tcp_recvspace = 16384
# no -o tcp_recvspace=32768
# no -o tcp_recvspace
tcp_recvspace = 32768
```

First the value of `tcp_recvspace` is displayed as being 16386 bytes (16 KB) by using the **no** command with the `-o` flag. The value is then increased to 32768 bytes (32 KB). Notice that there is *no* space on either side of the equal sign in the command to set the value of `tcp_recvspace`. If a space is inserted, the command will fail with the error message shown in Example 34-3.

Example 34-3 Error message from the no command

```
# no -o tcp_recvspace = 16384
Some parameters were not parsed.
Usage: no -h [tunable] | {-L [tunable]} | {-x [tunable]}
       no [-p|-r] (-a | {-o tunable})
       no [-p|-r] (-D | ({-d tunable} {-o tunable=value}))
NOTE:
```

1. The `no` commands can only be executed by root.
 2. `-r`, `-p` and `-C` flags must be placed at the beginning of the command line.
 3. `-r` and `-p` are mutual exclusive.
 4. Display (`-a,-o` option) and modification (`-o` option=`value`, `-d` option, `-D`) flags cannot be mixed.
 5. `-D` flag can not mixed with other modification (`-o` option=`value`, `-d` option) flags.
-

Table 34-1 on page 671 shows a list of adapter types and the suggested minimum buffer and MTU sizes.

Table 34-1 Suggested minimum buffer and MTU sizes for adapters

Device	Speed	MTU	tcp_sendspace	tcp_recvspace	sb_max	rfc1323
Ethernet	10 Mbit	1500	16384	16384	32768	0
Ethernet	100 Mbit	1500	16384	16384	32768	0
Ethernet	Gigabit	1500	65535	16384	131072	0
Ethernet	Gigabit	9000	131072	65535	262144	0
Ethernet	Gigabit	9000	131072	92160	262144	1
ATM	155 Mbit	1500	16384	16384	131072	0
ATM	155 Mbit	9180	65535	65535 ^a	131072	0
ATM	155 Mbit	65527	655360	655360 ^b	1310720	1
FDDI	100 Mbit	4352	45056	45056	90012	0

a. Certain values of tcp_recvspace and tcp_sendspace will result in poor performance on ATM adapters. For example, an MTU size of 9180, a tcp_sendspace set to 16384, and a tcp_recvspace set to 32768 or 65535 results in poor performance. Setting the tcp_sendspace and tcp_recvspace both to 65535 results in a good performance. For best performance in this case, ensure that tcp_sendspace is equal to or larger than tcp_recvspace.

b. The TCP window is only a 16 bit size. With ATM adapters with large MTU sizes of 32 KB or 64 KB, streaming may be poor. To overcome this 16 bit limit, set the value of rfc1323 to 1 (one).

Permanently change a variable

When you change the value of a variable, it is lost after a reboot. To ensure that a change is permanent, use the **no** command with the **-r** or **-p** flags.

When using the **-r** flag to change the value of a variable, the variable will only be changed after the next reboot. The reason for this is that the currently variable value is not changed, but an entry is made into the `/etc/tunables/nextboot` file. In Example 34-4 we displayed the contents of the `/etc/tunables/nextboot` file, and there are no entries in the file related to the **no** command. We then displayed the value of the `ipforwarding` variable; this is set to its default value of 0. After executing the `no -ro ipforwarding` command, the variable value of the `ipforwarding` is unchanged, but an entry is made in the `/etc/tunables/nextboot` file.

Example 34-4 Using no -r command to make variable changes

```
# cat /etc/tunables/nextboot
i00:

nfso:
    nfs_dynamic_retrans = "1"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"
```

```

vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"
# no -o ipforwarding
ipforwarding = 0
# no -ro ipforwarding=1
# no -o ipforwarding
ipforwarding = 0
# cat /etc/tunables/nextboot
ioo:

nfs0:
    nfs_dynamic_retrans = "1"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"

vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"

no:
    ipforwarding = "1"

```

When using the `-p` flag to change a variable, the current value is updated and an entry is made in the `/etc/tunables/nextboot` file. In Example 34-5 we use the `no` command with the `-p` flag to permanently change the value of a variable. In this example we reset all of the `no` options to the defaults. The contents of the `/etc/tunables/nextboot` file are displayed, and there are no entries for any `no` variables. When running the `no -o ipforwarding` command, notice that the value has been set to 0. When executing the `no -po ipforwarding` command, notice that the `/etc/tunables/nextboot` file and the `ipforwarding` variable have been updated.

Example 34-5 Using no -p to make variable changes

```

# cat /etc/tunables/nextboot
ioo:

nfs0:
    nfs_dynamic_retrans = "1"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"

vmo:
    maxperm% = "50"

```

```
        maxclient% = "50"
        spec_dataseg_int = "0"
# no -o ipforwarding
ipforwarding = 0
# no -po ipforwarding=1
# cat /etc/tunables/nextboot
ioo:

nfso:
    nfs_dynamic_retrans = "1"
    nfs_v3_vm_bufs = "5000"
    nfs_v2_vm_bufs = "5000"

vmo:
    maxperm% = "50"
    maxclient% = "50"
    spec_dataseg_int = "0"

no:
    ipforwarding = "1"
# no -o ipforwarding
ipforwarding = 1
```

Tracing performance problems

This part describes the use of the AIX **trace** command and the tools that support or post-process the output of it.

- ▶ The **trace** command, described in “trace” on page 760, is used to monitor user and kernel subsystems statistics in detail.
- ▶ The **trcrpt** command, described in 40.3, “trcrpt” on page 777, is used to format a raw trace file into a readable trace file.
- ▶ The **trcnm** command, described in 40.2, “trcnm” on page 775, is used to generate a list of all symbols with their addresses defined in the kernel.

- ▶ The **gennames** command, described in “gennames” on page 704, is used to gather address mapping information necessary for other commands.
- ▶ The **stripnm** command, described in “stripnm” on page 724, produces an output similar to the output generated by the **gennames** command, which is required for using the **tprof**, **filemon**, **netpmon**, and **pprof** commands in real-time mode.
- ▶ The **genkex** command, described in 36.5, “genkex” on page 713, extracts the list of kernel extensions currently loaded into the system and displays the address, size, and path name for each kernel extension in the list.
- ▶ The **genkld** command, described in 36.4, “genkld” on page 712, extracts the list of shared objects for all processes currently loaded into the shared segment and displays the address, size, and path name for each object on the list.
- ▶ The **genld** command, described in 36.3, “genld” on page 710, collects the list of all processes currently running on the system, and optionally reports the list of loaded objects corresponding to each process.
- ▶ The **curt** command, described in Chapter 35, “The curt command” on page 677, is a trace post processing tool that summarizes system utilization. Usually you would look at the output of the **curt** command to get an overview of the state of the system before analyzing the trace in detail.
- ▶ The **locktrace** command, described in Chapter 37, “The locktrace command” on page 719, is used to determine which kernel locks will be traced by the trace subsystem.
- ▶ The **splat** command, described in Chapter 39, “The splat command” on page 729, is a trace post processing tool that produces kernel limple_lock usage reports.

The **curt** command

The CPU Usage Reporting Tool (**curt**) takes an AIX trace file as input and produces a number of statistics related to CPU utilization and process/thread activity. These easy-to-read statistics enable quick and easy tracking of what a specific application is doing. For information about **trace**, refer to Chapter 40, “The trace, trcnm, and trcrpt commands” on page 759.

The **curt** command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset that is obtained from the AIX base installation media.

35.1 curt

The syntax for the **curt** command is:

```
curt -i inputfile [-o outputfile] [-n gennamesfile] [-m trcnmfile]
[-a pidnamefile] [-f timestamp] [-l timestamp] [-ehpstP] [-V]
```

Flags

-i inputfile	Specifies the input AIX trace file to be analyzed.
-o outputfile	Specifies an output file (default is stdout).
-n gennamesfile	Specifies a names file produced by gennames .
-m trcnmfile	Specifies a names file produced by trcnm .
-a pidnamefile	Specifies a PID-to-process name mapping file.
-f timestamp	Starts processing trace at time stamp seconds.
-l timestamp	Stops processing trace at time stamp seconds.
-e	Outputs elapsed time information for system calls.
-h	Displays usage text (this information).
-p	Shows ticks as trace processing progresses.
-s	Outputs information about errors returned by system calls.
-t	Outputs detailed thread by thread information.
-P	Outputs detailed pthread information.

Parameters

inputfile	The AIX trace file that should be processed by curt .
gennamesfile	The names file as produced by gennames .
trcnmfile	The names file as produced by trcnm .
outputfile	The names of the output file created by curt .
pidnamefile	If the trace process name table is not accurate, or if more descriptive names are desired, use the -a flag to specify a PID to process name mapping file. This is a file with lines consisting of a process ID (in decimal) followed by a space, then an ASCII string to use as the name for that process.
timestamp	The time in seconds at which to start and stop the trace file processing.

35.1.1 Information about measurement and sampling

A raw (unformatted) system trace from AIX 5L is read by **curt** to produce summaries on CPU utilization and either process or thread activity. This summary information is useful for determining which application, system call, or interrupt handler is using most of the CPU time and is a candidate to be optimized to improve system performance.

Table 35-1 lists the minimum trace hooks required for **curt**. Using only these trace hooks will limit the size of the trace file. However, other events on the system may not be captured in this case. This is significant if you intend to analyze the trace in more detail.

Table 35-1 Minimum trace hooks required for *curt*

HOOK ID	Event Name	Event Explanation
100	HKWD_KERN_FLIH	Occurrence of a first-level interrupt, such as an I/O interrupt, a data access page fault, or a timer interrupt (scheduler).
101	HKWD_KERN_SVC	A thread has issued a system call.
102	HKWD_KERN_SLIH	Occurrence of a second-level interrupt; that is, first-level I/O interrupts are being passed on to the second-level interrupt handler who then is working directly with the device driver.
103	HKWD_KERN_SLIHRET	Return from a second-level interrupt to the caller (usually a first-level interrupt handler).
104	HKWD_KERN_SYSCRET	Return from a system call to the caller (usually a thread).
106	HKWD_KERN_DISPATCH	A thread has been dispatched from the runqueue to a CPU.
10C	HKWD_KERN_IDLE	The idle process has been dispatched.
119	HKWD_KERN_PIDSIG	A signal has been sent to a process.
134	HKWD_SYSC_EXECVE	An exec SVC has been issued by a (forked) process.
135	HKWD_SYSC_EXIT	An exit SVC has been issued by a process.
139	HKWD_SYSC_FORK	A fork SVC has been issued by a process.

HOOK ID	Event Name	Event Explanation
200	HKWD_KERN_RESUME	A dispatched thread is being resumed on the CPU.
210	HKWD_KERN_INITP	A kernel process has been created.
38F	HKWD_DR	A processor has been added/removed.
465	HKWD_SYSC_CRTHREAD	A thread_create SVC has been issued by a process.

Trace hooks 119 and 135 are used to report on the time spent in the `exit()` system call. This is special because a process will enter it but will never return (because the calling process terminates). However a `SIGCHLD` signal is sent to the parent process of the exiting process, and this event is reflected in the trace by a `HKWD_KERN_PIDSIG` trace hook. `curt` will match this trace hook with the `exit()` system call trace hook (`HKWD_KERN_SVC`) and treat it as the system call return for the `exit()` system call.

35.2 Examples for `curt`

To generate a trace to be used in the following examples, we perform the following steps.

The first step is generate a system trace from the system. This can be done by using the `trace.sh` script as supplied by `perfpmr`. See Chapter 7, “The `perfpmr` command” on page 115 for details, or alternatively, you can run `trace` as shown in Example 35-1 on page 681 (see 40.1.3, “Ways to start and stop trace” on page 767 for details on the `trace` command).

Preparing to run `curt` is a four-stage process as follows:

1. Build the raw trace
This create the files listed in Example 35-1 on page 681, producing one raw trace file per CPU. The files are called `trace.raw-0`, `trace.raw-1`, and so on for each CPU. An additional raw trace file called `trace.raw` is also generated. This is a master file that has information that ties in the other CPU-specific traces.
2. Merge the trace files
To merge the trace files together to form one raw trace file, run the `trcrpt` command as shown in Example 35-1 on page 681.
3. Create the supporting files `gennamesfile` and `trcnmfile`
Neither the `gennamesfile` nor the `trcnmfile` file are necessary for `curt` to run. However, if you provide one or both of those files, `curt` will output names for system calls and interrupt handles instead of just addresses. The `gennames`

command output includes more information than the `trcnm` command output, and so, while the `trcnmfile` will contain most of the important address to name mapping data, a `gennamesfile` will enable `curlt` to output more names, especially interrupt handlers. `gennames` requires root authority to run. `trcnm` can be run by any user.

4. Generate the `curlt` output.

Example 35-1 Creating a trace file for `curlt` to analyze

```
# HOOKS="100,101,102,103,104,106,10C,119,134,135,139,200,210,38F,465"
# SIZE="1000000"
# export HOOKS SIZE
# trace -n -C all -d -j $HOOKS -L $SIZE -T $SIZE -afo trace.raw
# trcon ; sleep 5 ; trcstop
# unset HOOKS SIZE
# ls trace.raw*
trace.raw  trace.raw-0  trace.raw-1  trace.raw-2  trace.raw-3
# trcrpt -C all -r trace.raw > trace.r
# rm trace.raw*
# ls trace*
trace.r
# gennames > gennames.out
# trcnm > trace.nm
```

Alternatively, “-J `curlt`” can be used in place of “-j `$HOOKS`” for the `trace` command from Example 35-1.

Overview of the reports generated by `curlt`

The following is an overview of the reports that can be generated by the `curlt` command.

- ▶ A report header with the trace file name, trace size, and date and time the trace was taken. The header also includes the command used when the trace was run.
- ▶ For each CPU (and a summary of all of the CPUs), processing time expressed in milliseconds and as a percentage (idle and non-idle percentages are included) for various CPU usage categories.
- ▶ Average thread affinity across all CPUs and for each individual CPU.
- ▶ The total number of process dispatches for each individual CPU.
- ▶ Information about the amount of CPU time spent in application and system call (syscall) mode, expressed in milliseconds and as a percentage by thread, process, and process type. Also included are the number of threads per process and per process type.

- ▶ Information about the amount of CPU time spent executing each kernel process, including the idle process, expressed in milliseconds and as a percentage of the total CPU time.
- ▶ Information about completed system calls that includes the name and address of the system call, the number of times the system call was executed, and the total CPU time expressed in milliseconds and as a percentage with average, minimum, and maximum time the system call was running.
- ▶ Information about pending system calls (system calls for which the system call return has not occurred at the end of the trace). The information includes the name and address of the system call, the thread or process that made the system call, and the accumulated CPU time the system call was running, expressed in milliseconds.
- ▶ Information about the first level interrupt handlers (FLIHs) that includes the type of interrupt, the number of times the interrupt occurred, and the total CPU time spent handling the interrupt with average, minimum, and maximum time. This information is given for all CPUs and for each individual CPU. If there are any pending FLIHs (FLIHs for which the resume has not occurred at the end of the trace), for each CPU the accumulated time and the pending FLIH type is reported.
- ▶ Information about the second level interrupt handlers (SLIHs) that includes the interrupt handler name and address, the number of times the interrupt handler was called, and the total CPU time spent handling the interrupt with average, minimum, and maximum time. This information is given for all CPUs and for each individual CPU. If there are any pending SLIHs (SLIHs for which the return has not occurred at the end of the trace), for each CPU the accumulated time and the pending SLIH name and address is reported.

To create additional, specialized reports with **curt**, run the **curt** command using the flags described below:

- e** Produces a report that includes the statistics displayed in “The default report” on page 683 and includes additional information about the System Calls Summary Report. The additional information pertains to the total, average, maximum, and minimum elapsed times a system call was running. Refer to Example 35-13 on page 696 for this report.
- s** Produces a report that includes the statistics displayed in “The default report” on page 683, and includes a report on errors returned by system calls. Refer to Example 35-14 on page 697 for this report.
- t** Produces a report that includes the statistics displayed in “The default report” on page 683, and includes a detailed report on thread status that includes the amount of time the thread was in application and kernel mode, what system calls the thread made, processor affinity, the number of times the thread was dispatched, and to what CPU it was dispatched. The report

also includes dispatch wait times and details of interrupts. Refer to Example 35-15 on page 698 for this report.

- p Produces a report that includes a detailed report on process status that includes the amount of CPU time the process was in application and system call mode, which threads were in the process, and what system calls the process made. Refer to Example 35-16 on page 700.

The default report

This section explains the default report created by **curt**, using the following command:

```
curt -i trace.r -m trace.nm -n gennames.out -o curt.out
```

The **curt** output always includes this default report in its output. The default report includes the following sessions:

- ▶ General Information
- ▶ System Summary
- ▶ Processor Summary
- ▶ Application Summary by TID
- ▶ Application Summary by PID
- ▶ Application Summary by Process Type
- ▶ Kproc Summary
- ▶ System Calls Summary
- ▶ Pending System Calls Summary
- ▶ FLIH Summary
- ▶ SLIH Summary

General information

The first information in the report is the time and date when this particular **curt** command was run, including the syntax of the **curt** command line that produced the report.

The General Information section also contains some information about the AIX trace file that was processed by **curt**. This information consists of the trace file name, size, and creation date. The command used to invoke the AIX trace facility and gather the trace file is displayed at the end of the report.

A sample of this output is shown in Example 35-2.

Example 35-2 General information from curt.out

```
Run on Mon Apr 14 17:26:06 2003
Command line was:
curt -i trace.r -m trace.nm -n gennames.out -o curt.out
----
AIX trace file name = trace.r
```

AIX trace file size = 3525612
 AIX trace file created = Mon Apr 14 17:12:14 2003

Command used to gather AIX trace was:
 trace -n -C all -d -j 100,101,102,103,104,106,10C,119,134,135,139,200,210,38F,465 -L 1000000
 -T 1000000 -afo trace.raw

System summary

The next part of the default output is the System Summary, shown in Example 35-3.

Example 35-3 The System Summary report from curt.out

System Summary			
processing total time (msec)	percent total time (incl. idle)	percent busy time (excl. idle)	processing category
=====	=====	=====	=====
14998.65	73.46	92.98	APPLICATION
591.59	2.90	3.66	SYSCALL
48.33	0.24	0.30	KPROC
486.19	2.38	3.00	FLIH
49.10	0.24	0.30	SLIH
8.83	0.04	0.05	DISPATCH (all procs. incl. IDLE)
1.04	0.01	0.01	IDLE DISPATCH (only IDLE proc.)
-----	-----	-----	
16182.69	79.26	100.00	CPU(s) busy time
4234.76	20.74		IDLE
-----	-----	-----	
20417.45			TOTAL
Avg. Thread Affinity =		0.99	

This portion of the report describes the time spent by the system as a whole (all CPUs) in various execution modes.

The System Summary has the following fields:

- Processing total This column gives the total time in milliseconds for the corresponding processing category.
- Percent total time This column gives the time from the first column as a percentage of the sum of total trace elapsed time for all processors. This includes whatever amount of time each processor spent running the IDLE process.

Percent busy	This column gives the time from the first column as a percentage of the sum of total trace elapsed time for all processors without including the time each processor spent executing the IDLE process.
Avg. Thread Affinity	The Avg. Thread Affinity is the probability that a thread was dispatched to the same processor that it last executed on.
The possible execution modes or processing categories translate as follows:	
APPLICATION	The sum of times spent by all processors in User (that is, non-supervisory or non-privileged) mode.
SYSCALL	The sum of times spent by all processors doing System Calls. This is the portion of time that a processor spends executing in the kernel code providing services directly requested by a user process.
FLIH	The sum of times spent by all processors in FLIHs (first level interrupt handlers). The FLIH time consists of the time from when the FLIH is entered until the SLIH is entered, then from when the SLIH returns back into the FLIH until either dispatch or resume is called.
SLIH	The sum of times spent by all processors in SLIHs (second level interrupt handlers). The SLIH time consists of the time from when a SLIH is entered until it returns. Note nested interrupts may occur inside an SLIH. These FLIH times are not counted as SLIH time but rather as FLIH time as described above.
DISPATCH	The sum of times spent by all processors in the AIX dispatch code. The time starts when the dispatch code is entered and ends when the resume code is entered. The dispatch code corresponds to the OS, deciding which thread will run next and doing the necessary bookkeeping. This time includes the time spent dispatching all threads (that is, includes the dispatch of the IDLE process).
IDLE DISPATCH	The sum of times spent by all processors in the AIX dispatch code where the process being dispatched was the IDLE process. Because it is the IDLE process being dispatched, the overhead spent in dispatching is less critical than other dispatch times where there is useful work being dispatched. Because the Dispatch category already includes the IDLE Dispatch category's time, the

IDLE Dispatch category's time will not be included in either of the total categories CPU busy time or TOTAL.

CPU(s) busy time	The sum of times spent by all processors executing in application, kernel, FLIH, SLIH, and dispatch modes.
IDLE	The sum of times spent by all processors executing the IDLE process.
TOTAL	The sum of CPU(s) busy time and WAIT.

The System Summary in Example 35-3 on page 684 shows that the CPU spends most of its time in application mode. We still have 4234.76 ms of idle time so we know that we have enough CPU to run our applications. The Kproc Summary, which can be seen in Example 35-8 on page 690, reports similar values. If there was insufficient CPU power then we would not expect to see any wait time. The Avg. Thread Affinity value is 0.99, showing good processor affinity (threads returning to the same processor when they are ready to be re-run).

Processor summary

This part of the **curt** output follows the System Summary and is essentially the same information but broken down on a processor-by processor basis. The same description that was given for the System Summary applies here, except that the phrase "sum of times spent by all processors" can be replaced by "time spent by this processor". A sample of processor summary output is shown in Example 35-4.

Example 35-4 The Processor Summary from *curt.out*

```

Processor Summary processor number 0
-----
processing      percent      percent
total time      total time      busy time
(msec)          (incl. idle)    (excl. idle)    processing category
-----
45.07           0.88           5.16  APPLICATION
591.39         11.58          67.71 SYSCALL
47.83          0.94           5.48  KPROC
173.78         3.40           19.90 FLIH
9.27           0.18           1.06  SLIH
6.07           0.12           0.70  DISPATCH (all procs. incl. IDLE)
1.04           0.02           0.12  IDLE DISPATCH (only IDLE proc.)
-----
873.42         17.10          100.00 CPU(s) busy time
4232.92        82.90
-----
5106.34                                TOTAL

Avg. Thread Affinity =          0.98

```

Total number of process dispatches = 1620
 Total number of idle dispatches = 782

```

Processor Summary processor number 1
-----
processing      percent      percent
total time      total time    busy time
(msec) (incl. idle) (excl. idle) processing category
=====
4985.81         97.70         97.70 APPLICATION
0.09            0.00          0.00 SYSCALL
0.00            0.00          0.00 KPROC
103.86          2.04          2.04 FLIH
12.54           0.25          0.25 SLIH
0.97            0.02          0.02 DISPATCH (all procs. incl. IDLE)
0.00            0.00          0.00 IDLE DISPATCH (only IDLE proc.)
-----
5103.26         100.00        100.00 CPU(s) busy time
0.00            0.00          IDLE
-----
5103.26                                     TOTAL
  
```

Avg. Thread Affinity = 0.99

Total number of process dispatches = 516
 Total number of idle dispatches = 0

Avg. Thread Affinity = 0.99

...(lines omitted)...

The Total number of process dispatches refers to how many times AIX dispatched any non-IDLE process on this processor.

Application Summary by Thread ID (TID)

The Application Summary by Thread ID shows an output of all threads that were running on the system during trace collection and their CPU consumption. The thread that consumed the most CPU time during the trace collection is at the top of the list. The report is shown in Example 35-5.

Example 35-5 Application Summary by Thread ID

```

Application Summary (by Tid)
-----
-- processing total (msec) -- -- percent of total processing time --
combined application syscall combined application syscall name (Pid Tid)
=====
  
```

4986.2355	4986.2355	0.0000	24.4214	24.4214	0.0000	cpu(18418 32437)
4985.8051	4985.8051	0.0000	24.4193	24.4193	0.0000	cpu(19128 33557)
4982.0331	4982.0331	0.0000	24.4009	24.4009	0.0000	cpu(18894 28671)
83.8436	2.5062	81.3374	0.4106	0.0123	0.3984	disp+work(20390 28397)
72.5809	2.7269	69.8540	0.3555	0.0134	0.3421	disp+work(18584 32777)
69.8023	2.5351	67.2672	0.3419	0.0124	0.3295	disp+work(19916 33033)
63.6399	2.5032	61.1368	0.3117	0.0123	0.2994	disp+work(17580 30199)
63.5906	2.2187	61.3719	0.3115	0.0109	0.3006	disp+work(20154 34321)
62.1134	3.3125	58.8009	0.3042	0.0162	0.2880	disp+work(21424 31493)
60.0789	2.0590	58.0199	0.2943	0.0101	0.2842	disp+work(21992 32539)

...(lines omitted)...

The output has two main sections, of which one shows the total processing time of the thread in milliseconds (processing total (msec)), and the other shows the CPU time the thread has consumed, expressed as a percentage of the total CPU time (percent of total processing time).

► Processing total (msec) section

combined The total amount of time, expressed in milliseconds, that the thread was running in either application or kernel mode.

application The amount of time, expressed in milliseconds, that the thread spent in application mode.

syscall The amount of CPU time, expressed in milliseconds, that the thread spent in system call mode.

► Percent of total processing time section

combined The amount of time the thread was running, expressed as percentage of the total processing time.

application The amount of time the thread spent in application mode, expressed as percentage of the total processing time.

syscall The amount of CPU time that the thread spent in system call mode, expressed as percentage of the total processing time.

name (Pid Tid) The name of the process associated with the thread, its process ID, and its thread ID.

The Application Summary by TID from **curt** shows an output of all threads that were running on the system during the time of trace collection and their CPU consumption as shown in Example 35-5 on page 687. The thread that consumed the most CPU time during the time of the trace collection is on top of the list.

We created a test program called *cpu* with CPU-intensive code. Example 35-5 on page 687 shows that the CPU spent most of its time in application mode running the *cpu* process. To learn more about this process, we could run the **gprof** command (see Chapter 19, “The gprof, pprof, prof, and tprof commands” on page 297) or other profiling tools to profile the process, or look directly at the formatted trace file from the **trcrpt** command. (See 40.3, “trcrpt” on page 777.)

Application Summary by Process ID (PID)

The Application Summary (by PID) has the same content as the Application Summary (by TID), except that the threads that belong to each process are consolidated, and the process that consumed the most CPU time during the monitoring period is at the beginning of the list.

In Example 35-6, the column name (PID) (Thread Count) shows the process name, its process ID, and the number of threads that belong to this process and that have been accumulated for this line of data.

Example 35-6 The Application and Kernel Summary (by PID) from curt.out

Application and Kernel Summary (by Pid)

```

-----
-- processing total (msec) --      -- percent of total processing time --
combined application syscall combined application syscall name (Pid)(Thread Count)
=====
4986.2355  4986.2355  0.0000  24.4214   24.4214  0.0000  cpu(18418) (1)
4985.8051  4985.8051  0.0000  24.4193   24.4193  0.0000  cpu(19128) (1)
4982.0331  4982.0331  0.0000  24.4009   24.4009  0.0000  cpu(18894) (1)
 83.8436   2.5062 81.3374  0.4106    0.0123  0.3984  disp+work(20390) (1)
 72.5809   2.7269 69.8540  0.3555    0.0134  0.3421  disp+work(18584) (1)
 69.8023   2.5351 67.2672  0.3419    0.0124  0.3295  disp+work(19916) (1)
 63.6399   2.5032 61.1368  0.3117    0.0123  0.2994  disp+work(17580) (1)
 63.5906   2.2187 61.3719  0.3115    0.0109  0.3006  disp+work(20154) (1)
 62.1134   3.3125 58.8009  0.3042    0.0162  0.2880  disp+work(21424) (1)
 60.0789   2.0590 58.0199  0.2943    0.0101  0.2842  disp+work(21992) (1)
...(lines omitted)...

```

Application Summary by process type

The Application Summary (by process type) consolidates all processes of the same name and sorts them in descending order of combined processing time.

The name (thread count) column shows the name of the process and the number of threads that belong to this process name (type) that were running on the system during the monitoring period. It is shown in Example 35-7 on page 690.

Example 35-7 The Application Summary (by process type) from curt.out

Application Summary (by process type)

```

-- processing total (msec) -- -- percent of total processing time --
combined  application  syscall  combined  application  syscall  name (thread count)
-----  -----  -----  -----  -----  -----  -----
14954.0738  14954.0738  0.0000  73.2416  73.2416  0.0000  cpu(3)
573.9466    21.2609    552.6857  2.8111   0.1041   2.7069  disp+work(9)
20.9568     5.5820    15.3748  0.1026   0.0273   0.0753  trcstop(1)
10.6151     2.4241     8.1909   0.0520   0.0119   0.0401  i4llmd(1)
8.7146      5.3062     3.4084   0.0427   0.0260   0.0167  dtgreet(1)
7.6063      1.4893     6.1171   0.0373   0.0073   0.0300  sleep(1)

```

...(lines omitted)...

Kproc Summary by Thread ID (TID)

The Kproc Summary (by TID) shows an output of all kernel process threads that were running on the system during the time of trace collection and their CPU consumption. The thread that consumed the most CPU time during the time of the trace collection is at the beginning of the list shown in Example 35-8.

Example 35-8 Kproc summary by TID

Kproc Summary (by Tid)

```

-- processing total (msec) -- -- percent of total time --
combined  operation  kernel  combined  operation  kernel  name (Pid Tid Type)
-----  -----  -----  -----  -----  -----  -----
4232.9216  0.0000  4232.9216  20.7319  0.0000  20.7319  wait(516 517 W)
30.4374    0.0000  30.4374   0.1491  0.0000  0.1491  lrud(1548 1549 -)

```

...(lines omitted)...

Kproc Types

Type	Function	Operation
W	idle thread	-

The Kproc Summary has the following fields:

name (Pid Tid Type)	The name of the kernel process associated with the thread, its process ID, its thread ID, and its type. The kproc type is defined in the Kproc Types listing following the Kproc Summary.
---------------------	---

processing total (msec) section

combined	The total amount of CPU time, expressed in milliseconds, that the thread was running in either operation or kernel mode
operation	The amount of CPU time, expressed in milliseconds, that the thread spent in operation mode
kernel	The amount of CPU time, expressed in milliseconds, that the thread spent in kernel mode

percent of total time section

combined	The amount of CPU time that the thread was running, expressed as a percentage of the total processing time
operation	The amount of CPU time that the thread spent in operation mode, expressed as a percentage of the total processing time
kernel	The amount of CPU time that the thread spent in kernel mode, expressed as a percentage of the total processing time

Kproc Types section

Type	A single letter to be used as an index into this listing
Function	A description of the nominal function of this type of kernel process

System Calls Summary

The System Calls Summary provides a list of all system calls that were used on the system during the monitoring period, as shown in Example 35-9. The list is sorted by the total time in milliseconds consumed by each type of system call.

Example 35-9 The System Calls Summary from `curt.out`

System Calls Summary						
Count	Total Time (msec)	% sys time	Avg Time (msec)	Min Time (msec)	Max Time (msec)	SVC (Address)
605	355.4475	1.74%	0.5875	0.0482	4.5626	<code>kwrite(4259c4)</code>
733	196.3752	0.96%	0.2679	0.0042	2.9948	<code>kread(4259e8)</code>
3	9.2217	0.05%	3.0739	2.8888	3.3418	<code>execve(1c95d8)</code>
38	7.6013	0.04%	0.2000	0.0051	1.6137	<code>_loadx(1c9608)</code>
1244	4.4574	0.02%	0.0036	0.0010	0.0143	<code>lseek(425a60)</code>
45	4.3917	0.02%	0.0976	0.0248	0.1810	<code>access(507860)</code>
63	3.3929	0.02%	0.0539	0.0294	0.0719	<code>_select(4e0ee4)</code>
2	2.6761	0.01%	1.3380	1.3338	1.3423	<code>kfork(1c95c8)</code>
207	2.3958	0.01%	0.0116	0.0030	0.1135	<code>_poll(4e0ecc)</code>

228	1.1583	0.01%	0.0051	0.0011	0.2436	kioc1(4e07ac)
9	0.8136	0.00%	0.0904	0.0842	0.0988	.smtcheckinit(1b245a8)
5	0.5437	0.00%	0.1087	0.0696	0.1777	open(4e08d8)
15	0.3553	0.00%	0.0237	0.0120	0.0322	.smtcheckinit(1b245cc)
2	0.2692	0.00%	0.1346	0.1339	0.1353	statx(4e0950)
33	0.2350	0.00%	0.0071	0.0009	0.0210	_sigaction(1cada4)
1	0.1999	0.00%	0.1999	0.1999	0.1999	kwaitpid(1cab64)
102	0.1954	0.00%	0.0019	0.0013	0.0178	klseek(425a48)

...(lines omitted)...

The System Calls Summary has the following fields:

Count	The number of times a system call of a certain type (see SVC (Address)) has been used (called) during the monitoring period
Total Time (msec)	The total time the system spent processing these system calls, expressed in milliseconds
% sys time	The total time the system spent processing these system calls, expressed as a percentage of the total processing time
Avg Time (msec)	The average time the system spent processing one system call of this type, expressed in milliseconds
Min Time (msec)	The minimum time the system needed to process one system call of this type, expressed in milliseconds
Max Time (msec)	The maximum time the system needed to process one system call of this type, expressed in milliseconds
SVC (Address)	The name of the system call and its kernel address

Pending System Calls Summary

The Pending System Calls Summary provides a list of all system calls that have been executed on the system during the monitoring period but have not completed. The list is sorted by TID. Example 35-10 displays the pending system calls summary.

Example 35-10 Pending System Calls Summary from curt.out

Pending System Calls Summary		
Accumulated Time (msec)	SVC (Address)	Procname (Pid Tid)
-----	-----	-----
0.0656	_select(4e0ee4)	sendmail(7844 5001)
0.0452	_select(4e0ee4)	syslogd(7514 8591)
0.0712	_select(4e0ee4)	snmpd(5426 9293)
0.0156	kioc1(4e07ac)	trcstop(47210 18379)

```

0.0274 kwaitpid(1cab64)          ksh(20276 44359)
0.0567 kread4259e8)           ksh(23342 50873)
...(lines omitted)...

```

The Pending System Calls Summary has the following fields:

Accumulated Time(msec) The accumulated CPU time that the system spent processing the pending system call, expressed in milliseconds.

SVC (Address) The name of the system call and its kernel address.

Procname (Pid Tid) The name of the process associated with the thread that made the system call, its PID, and the TID.

FLIH Summary

The FLIH Summary lists all first level interrupt handlers that were called during the monitoring period, as shown in Example 35-11.

The Global Flih Summary lists the total of first level interrupts on the system, while the Per CPU Flih Summary lists the first level interrupts per CPU.

Example 35-11 The Flih summaries from curt.out

Global Flih Summary					
Count	Total Time (msec)	Avg Time (msec)	Min Time (msec)	Max Time (msec)	Flih Type
2183	203.5524	0.0932	0.0041	0.4576	31(DEC_R_INTR)
946	102.4195	0.1083	0.0063	0.6590	3(DATA_ACC_PG_FLT)
12	1.6720	0.1393	0.0828	0.3366	32(QUEUED_INTR)
1058	183.6655	0.1736	0.0039	0.7001	5(IO_INTR)

Per CPU Flih Summary					
CPU Number 0:					
Count	Total Time (msec)	Avg Time (msec)	Min Time (msec)	Max Time (msec)	Flih Type
635	39.8413	0.0627	0.0041	0.4576	31(DEC_R_INTR)
936	101.4960	0.1084	0.0063	0.6590	3(DATA_ACC_PG_FLT)
9	1.3946	0.1550	0.0851	0.3366	32(QUEUED_INTR)
266	33.4247	0.1257	0.0039	0.4319	5(IO_INTR)

CPU Number 1:					
Count	Total Time (msec)	Avg Time (msec)	Min Time (msec)	Max Time (msec)	Flih Type

```

=====
      4      0.2405      0.0601      0.0517      0.0735      3(DATA_ACC_PG_FLT)
     258     49.2098     0.1907     0.0060     0.5076     5(IO_INTR)
     515     55.3714     0.1075     0.0080     0.3696    31(DECR_INTR)
...(lines omitted)...

```

Pending Flih Summary

```

-----
Accumulated Time (msec)  Flih Type
=====
                        0.0123    5(IO_INTR)

```

...(lines omitted)...

The FLIH Summary report has the following fields:

Count	The number of times a first level interrupt of a certain type (see FLIH Type) occurred during the monitoring period.
Total Time (msec)	The total time the system spent processing these first level interrupts, expressed in milliseconds.
Avg Time (msec)	The average time the system spent processing one first level interrupt of this type, expressed in milliseconds.
Min Time (msec)	The minimum time the system needed to process one first level interrupt of this type, expressed in milliseconds.
Max Time (msec)	The maximum time the system needed to process one first level interrupt of this type, expressed in milliseconds.
Flih Type	The number and name of the first level interrupt.

In Example 35-11 on page 693, the following are the FLIH types:

DATA_ACC_PG_FLT	Data access page fault
QUEUED_INTR	Queued interrupt
DECR_INTR	Decrementer interrupt
IO_INTR	I/O interrupt

SLIH Summary

The SLIH Summary lists all second level interrupt handlers that were called during the monitoring period, as shown in Example 35-12.

The Global Slh Summary lists the total of second level interrupts on the system, while the Per CPU Slh Summary lists the second level interrupts per CPU.

Example 35-12 The Slh summaries from curt.out

Global Slh Summary

```

-----
Count  Total Time      Avg Time      Min Time      Max Time Slih Name(Address)
      (msec)         (msec)         (msec)         (msec)
=====
   43    7.0434         0.1638         0.0284         0.3763  .copyout(1a99104)
  1015   42.0601         0.0414         0.0096         0.0913  .i_mask(1990490)

```

Per CPU Slih Summary

CPU Number 0:

```

Count  Total Time      Avg Time      Min Time      Max Time Slih Name(Address)
      (msec)         (msec)         (msec)         (msec)
=====
     8    1.3500         0.1688         0.0289         0.3087  .copyout(1a99104)
    258    7.9232         0.0307         0.0096         0.0733  .i_mask(1990490)

```

CPU Number 1:

```

Count  Total Time      Avg Time      Min Time      Max Time Slih Name(Address)
      (msec)         (msec)         (msec)         (msec)
=====
    10    1.2685         0.1268         0.0579         0.2818  .copyout(1a99104)
   248   11.2759         0.0455         0.0138         0.0641  .i_mask(1990490)

```

...(lines omitted)...

The SLIH Summary report has the following fields:

- Count The number of times each SLIH was called during the monitoring period.
- Total Time (msec) The total time the system spent processing these second level interrupts, expressed in milliseconds.
- Avg Time (msec) The average time the system spent processing one second level interrupt of this type, expressed in milliseconds.
- Min Time (msec) The minimum time the system needed to process one second level interrupt of this type, expressed in milliseconds.
- Max Time (msec) The maximum time the system needed to process one second level interrupt of this type, expressed in milliseconds.
- Slih Name (Address) The name and kernel address of the second level interrupt.

Report generated with the -e flag

The report generated with the -e flag includes the reports shown in “The default report” on page 683, and also includes additional information in the System Calls Summary report as shown in Example 35-13. The additional information pertains to the total, average, maximum, and minimum elapsed times a system call was running.

Example 35-13 *curt output with the -e flag*

```
# curt -e -i trace.r -m trace.nm -n gennames.out -o curt.out
# cat curt.out
...(lines omitted)...

System Calls Summary
-----
Count Total    % sys  Avg    Min    Max    Tot    Avg    Min    Max    SVC
      Time    time  Time  Time  Time  ETime  ETime  Etime  ETime  (Address)
      (msec)
-----
605 355.4475 1.74% 0.5875 0.0482 4.5626 31172.7658 51.5252 0.0482 422.2323 write(4259c4)
733 196.3752 0.96% 0.2679 0.0042 2.9948 12967.9407 17.6916 0.0042 265.1204 kread(4259e8)
  3  9.2217 0.05% 3.0739 2.8888 3.3418  57.2051 19.0684 4.5475 40.0557 execve(1c95d8)
 38  7.6013 0.04% 0.2000 0.0051 1.6137  12.5002  0.3290 0.0051  3.3120 _loadx(1c9608)
1244 4.4574 0.02% 0.0036 0.0010 0.0143  4.4574  0.0036 0.0010  0.0143 lseek(425a60)
 45  4.3917 0.02% 0.0976 0.0248 0.1810  4.6636  0.1036 0.0248  0.3037 access(507860)
 63  3.3929 0.02% 0.0539 0.0294 0.0719  5006.0887 79.4617 0.0294 100.4802 _select(4e0ee4)
  2  2.6761 0.01% 1.3380 1.3338 1.3423  45.5026 22.7513 7.5745 37.9281 kfork(1c95c8)
207 2.3958 0.01% 0.0116 0.0030 0.1135 4494.9249 21.7146 0.0030 499.1363 _poll(4e0ecc)
228 1.1583 0.01% 0.0051 0.0011 0.2436  1.1583  0.0051 0.0011  0.2436 kiocntl(4e07ac)
  9  0.8136 0.00% 0.0904 0.0842 0.0988 4498.7472 499.8608 499.8052 499.8898 .smtcheckinit(1b245a8)
  5  0.5437 0.00% 0.1087 0.0696 0.1777  0.5437  0.1087 0.0696  0.1777 open(4e08d8)
 15  0.3553 0.00% 0.0237 0.0120 0.0322  0.3553  0.0237 0.0120  0.0322 .smtcheckinit(1b245cc)
  2  0.2692 0.00% 0.1346 0.1339 0.1353  0.2692  0.1346 0.1339  0.1353 statx(4e0950)
 33  0.2350 0.00% 0.0071 0.0009 0.0210  0.2350  0.0071 0.0009  0.0210 _sigaction(1cada4)
  1  0.1999 0.00% 0.1999 0.1999 0.1999 5019.0588 5019.0588 5019.0588 5019.0588 kwaitpid(1cab64)
102  0.1954 0.00% 0.0019 0.0013 0.0178  0.5427  0.0053 0.0013  0.3650 klseek(425a48)
...(lines omitted)...

Pending System Calls Summary
-----
Accumulated    Accumulated    SVC (Address)    Procname (Pid Tid)
Time (msec)    ETime (msec)
-----
0.0855        93.6498    kread(4259e8)    oracle(143984 48841)
...(lines omitted)...
```

The System Calls Summary in this example has the following fields in addition to the default System Calls Summary displayed in Example 35-9 on page 691:

Tot ETime (msec) The total amount of time from when the system call was started to its completion. This time will include any times

	spent servicing interrupts, running other processes, and so forth.
Avg ETime (msec)	The average amount of time from when the system call was started to when it completed. This includes any time spent servicing interrupts, running other processes, and so forth.
Min ETime (msec)	The minimum amount of time from when the system call was started to when it completed. This includes any time spent servicing interrupts, running other processes, and so forth.
Max ETime (msec)	The maximum amount of time from when the system call was started to when it completed. This includes any time spent servicing interrupts, running other processes, and so forth.

The preceding example report shows that the maximum elapsed time for the `kwrite` system call was 422.2323 msec, but the maximum CPU time was 4.5626 msec. If this amount of overhead time is unusual for the device being written to, further analysis is needed.

Sometimes comparing the average elapsed time to the average execution time shows that a certain system call is being delayed by something unexpected. Other debug measures should be used to investigate further.

Report generated with the `-s` flag

The report generated with the `-s` flag includes the reports shown in “The default report” on page 683 and includes reports on errors returned by system calls, as shown in Example 35-14.

Example 35-14 `curl` output with the `-s` flag

```
# curl -s -i trace.r -m trace.nm -n gennames.out -o curl.out
# cat curl.out
...(lines omitted)...
      Errors Returned by System Calls
      -----

Errors (errno : count : description) returned for System call:
socket_aio_dequeue(0x11e0d8)
  11 :      485 : "Resource temporarily unavailable"
      Errors (errno : count : description) returned for System call:
connext(0x11e24c)
  75 :         7 : "Socket is already connected"
...(lines omitted)...
```

If a large number of errors of a specific type or on a specific system call point to a system or application problem, other debug measures can be used to determine and fix the problem.

Report generated with the -t flag

The report generated with the -t flag includes the reports shown in “The default report” on page 683 as well as a detailed report on thread status that includes the amount of time the thread was in application and kernel mode, what system calls the thread made, processor affinity, the number of times the thread was dispatched, and to what CPU it was dispatched. The report also includes dispatch wait times and details of interrupts. It is shown in Example 35-15.

Example 35-15 curt output with the -t flag

...(lines omitted)...

Report for Thread Id: **48841** (hex bec9) Pid: 143984 (kex 23270)

Process Name: **oracle**

Total Application Time (ms): 70.324465
 Total Kernel Time (ms): 53.014910

Count	Total Time (msec)	Thread System Call Data			SVC (Address)
		Avg Time (msec)	Min Time (msec)	Max Time (msec)	
69	34.0819	0.4939	0.1666	1.2762	kwrite(169ff8)
77	12.0026	0.1559	0.0474	0.2889	kread(16a01c)
510	4.9743	0.0098	0.0029	0.0467	times(f1e14)
73	1.2045	0.0165	0.0105	0.0306	select(1d1704)
68	0.6000	0.0088	0.0023	0.0445	lseek(16a094)
12	0.1516	0.0126	0.0071	0.0241	getrusage(f1be0)

No Errors Returned by System Calls

Pending System Calls Summary

Accumulated Time (msec)	SVC (Address)
0.1420	kread(16a01c)

processor affinity: 0.583333

Dispatch Histogram for thread (CPUid : times_dispatched).

CPU 0 : 23
 CPU 1 : 23
 CPU 2 : 9

CPU 3 : 9
CPU 4 : 8
CPU 5 : 14
CPU 6 : 17
CPU 7 : 19
CPU 8 : 1
CPU 9 : 4
CPU 10 : 1
CPU 11 : 4

total number of dispatches: 131
total number of redispaches due to interupts being disabled: 1
avg. dispatch wait time (ms): 8.273515

Data on Interrupts that Occured while Thread was Running

Type of Interrupt	Count
-----	-----
Data Access Page Faults (DSI):	115
Instr. Fetch Page Faults (ISI):	0
Align. Error Interrupts:	0
IO (external) Interrupts:	0
Program Check Interrupts:	0
FP Unavailable Interrupts:	0
FP Imprecise Interrupts:	0
RunMode Interrupts:	0
Decrementer Interrupts:	18
Queued (Soft level) Interrupts:	15

...(lines omitted)...

The information in the threads summary includes:

Thread ID	The TID of the thread.
Process ID	The PID the thread belongs to.
Process Name	The process name, if known, that the thread belongs to.
Total Application Time (ms)	The amount of time, expressed in milliseconds, that the thread spent in application mode.
Total System Call Time (ms)	The amount of time, expressed in milliseconds, that the thread spent in system call mode.
Thread System Call Data	A system call summary for the thread; this has the same fields as the global System Call Summary. (See Example 35-9 on page 691.) It also includes elapsed

times if the -e flag is specified and error information if the -s flag is specified.

Pending System Calls Summary

If the thread was executing a system call at the end of the trace, a pending system call summary will be printed. This has the Accumulated Time and Supervisor Call (SVC Address) fields. It also includes elapsed time if the -e flag is specified.

processor affinity The process affinity, which is the probability that, for any dispatch of the thread, the thread was dispatched to the same processor that it last executed on.

Dispatch Histogram for thread

Shows the number of times the thread was dispatched to each CPU in the system.

total number of dispatches

The total number of times the thread was dispatched (not including redispatches described below).

total number of redispatches

The number of redispatches due to interrupts being disabled, which is when the dispatch disabled code is forced to dispatch the same thread that is currently running on that particular CPU because the thread had disabled some interrupts. This is only shown if non-zero.

avg. dispatch wait time (ms)

The average dispatch wait time is the average elapsed time for the thread from being undispached and its next dispatch.

Data on Interrupts This is a count of how many times each type of FLIH occurred while this thread was executing.

Report generated with the -p flag

When a report is generated using the -p flag, it gives detailed information about each process found in the trace. The following example shows the report generated for the router process (PID 129190). A sample output is given in Example 35-16.

Example 35-16 *curl output with -p flag*

...(lines omitted)...

```
Process Details for Pid: 129190
  Process Name: router
```

```

7 Tids for this Pid: 245889 245631 244599 82843 78701 75347
28941
Total Application Time (ms): 124.023749
Total System Call Time (ms): 8.948695

```

```

Process System Call Data
Count  Total Time  % sys  Avg Time  Min Time  Max Time  SVC (Address)
      (msec)   time   (msec)   (msec)   (msec)   (msec)
=====  =====  =====  =====  =====  =====  =====
93      3.6829  0.05%  0.0396   0.0060   0.3077  kread(19731c)
23      2.2395  0.03%  0.0974   0.0090   0.4537  kwrite(1972f8)
30      0.8885  0.01%  0.0296   0.0073   0.0460  select(208c5c)
1       0.5933  0.01%  0.5933   0.5933   0.5933  fsync(1972a4)
106     0.4902  0.01%  0.0046   0.0035   0.0105  klseek(19737c)
13      0.3285  0.00%  0.0253   0.0130   0.0387  semctl(2089e0)
6       0.2513  0.00%  0.0419   0.0238   0.0650  semop(2089c8)
3       0.1223  0.00%  0.0408   0.0127   0.0730  statx(2086d4)
1       0.0793  0.00%  0.0793   0.0793   0.0793  send(11e1ec)
9       0.0679  0.00%  0.0075   0.0053   0.0147  fstatx(2086c8)
4       0.0524  0.00%  0.0131   0.0023   0.0348  kfcntl(22aa14)
5       0.0448  0.00%  0.0090   0.0086   0.0096  yield(11dbec)
3       0.0444  0.00%  0.0148   0.0049   0.0219  recv(11e1b0)
1       0.0355  0.00%  0.0355   0.0355   0.0355  open(208674)
1       0.0281  0.00%  0.0281   0.0281   0.0281  close(19728c)

```

```

Pending System Calls Summary
-----
Accumulated  SVC (Address)  Tid
Time (msec)
-----
0.0452  select(208c5c)  245889
0.0425  select(208c5c)  78701
0.0285  select(208c5c)  82843
0.0284  select(208c5c)  245631
0.0274  select(208c5c)  244599
0.0179  select(208c5c)  75347

```

```

...(lines omitted)...

```

The `-p` flag process information includes the process ID and name, and a count and list of the TIDs belonging to the process. The total application and system call time for all the threads of the process is given. It also includes summary reports of all completed and pending system calls for the threads of the process.

The **gennames**, **genld**, **genkld**, **genkex**, and **gensyms** commands

The **gennames**, **genld**, **genkld**, **genkex**, and **gensyms** commands extract information from the running system for offline processing. They reside in `/usr/bin` and are part of the `bos.perf.tools` fileset, which can be installed from the AIX base installation media.

36.1 Offline generation tools

This chapter discusses various offline generation tools that extract information from the current running system for offline processing:

- ▶ The **gennames** command gathers name-to-address mapping information necessary for commands such as **tprof**, **filemon**, **netpmon**, **pprof**, and **curt** to work in offline mode. This is useful when it is necessary to post-process a trace file from a remote system or perform the trace data collection at one time and post-process it at another time.
- ▶ The **genld** command collects the list of all processes currently running on the system, and optionally reports the list of loaded objects corresponding to each process.
- ▶ The **genkld** command extracts the list of shared objects for all processes currently loaded into the shared segment and displays the virtual address, size, and path name for each object on the list.
- ▶ The **genkex** command extracts the list of kernel extensions currently loaded into the system and displays the address, size, and path name for each kernel extension in the list.
- ▶ The **gensyms** command extracts name-to-address mapping that is necessary for offline processing of other commands, such as **tprof** or **splat**.

36.2 gennames

The syntax of **gennames** is:

```
gennames [ -f ] filenames
```

Flags

-f

In addition to the **gennames** output without the **-f** flag, device information for logical and physical volumes and the virtual file system information are printed. This information is necessary for the **filemon** command to be run in offline mode.

Parameters

filenames

Optional list of program names for which the output of the **stripnm** command must be collected to allow the usage of the **tprof** command with the **-p** flag in offline mode.

The **gennames** command writes its output to standard output. For further use the output must be redirected into a file.

36.2.1 Information about measurement and sampling

The **gennames** command gathers the following information:

- ▶ The name-to-address mapping information for the currently running kernel (/unix). The output is similar to the output of the **stripnm /unix** command.
- ▶ A list of all loaded kernel extensions. This list is similar to the output of the **genkex** command. Refer to 36.5, “genkex” on page 713 for more information about this command.
- ▶ A list of all loaded shared libraries. This list is similar to the output of the **genkld** command. Refer to 36.4, “genkld” on page 712 for more information about this command.
- ▶ The output of the **stripnm** command for all kernel extensions and libraries. Refer to Chapter 38, “The stripnm command” on page 723 for more information about this command.
- ▶ The process ID and process name for all of the loaded processes. The data collected is similar to the data the **genld** command reports. Refer to 36.3, “genld” on page 710 for more information about this command.
- ▶ The symbols defined in the system header file /usr/include/sys/lockname.h.
- ▶ The name-to-address mapping for optionally specified programs. The output of the **stripnm** command is collected to allow subroutine breakdown with the **tprof** command used in offline mode with any of the specified programs. Refer to 38.1, “stripnm” on page 724 and 19.5, “tprof” on page 324 for more information about these commands.
- ▶ The device information for logical and physical volumes and the virtual file system information. This data is needed to run the **filemon** command in offline mode.

36.2.2 Examples for gennames

Example 36-1 shows the use of the **gennames** command to gather information for later use by commands such as **tprof**, **filemon**, **netpmon**, **pprof**, and **curt**.

Example 36-1 Gather gennames output

```
# gennames >gennames.out
```

Attention: The output produced by the **gennames** command can exceed 300,000 lines and 4 MB — the size depends on the number of loaded shared libraries and kernel extensions — so opening this file in an editor may take some time.

Next we show small examples of the major sections in the output of the **gennames** command.

The name-to-address mapping

The **gennames** command provides name-to-address mapping information for the currently used kernel, the loaded kernel extensions, the loaded shared libraries, and optionally the specified programs. Example 36-2 shows a part of the listing for the kernel (/unix).

Example 36-2 Output of gennames showing the name-to-address mapping

```
# gennames

gennames v1.1
 /unix | 00000000| 00000000| 0052b988| | initialize
Symbols from /unix

 (... lines omitted ...)

../../../../../src/bos/kernel/net/llsleep.c| | file | |
| | | | |
.llsleep | 1129384| unamex| | | .text
.llsleep | 1129384| extern| | | .text
.llsleep_timeout_rt | 1129800| unamex| | | .text
.llsleep_timeout_rt | 1129800| extern| | | .text
.llwakeup | 1129856| unamex| | | .text
.llwakeup | 1129856| extern| | | .text
_$STATIC | 1709080| unamex| | | .text
llsleep | 1878320| extern| | | .text
llsleep_timeout_rt | 1878332| extern| | | .text
llwakeup | 1878344| extern| | | .text

 (... lines omitted ...)
```

The first line for each module, in this example /unix, shows the file name followed by the load address, the text section offset within the module, and the module size. These values are hexadecimal numbers. The symbol address offsets, for example for the function `.llwakeup`, are decimal values that represent the offset values of the symbols in the object modules text segment. To calculate the address of a symbol, use this equation $\text{start address} + \text{text offset} + \text{symbol offset} = \text{address in memory}$. In the example above, the external function `.llwakeup` is at offset 1129856. The load address and the text section offset of /unix is zero. The resulting address for the `.llwakeup` symbol is $0 + 0 + 1129856 = 1129856$. To verify this address, use the **kdb** sub command `nm .llwakeup` on the same system. This returns the address for symbol `.llwakeup` in the running kernel, in our case 0x113d80, that equals the above value (1129856).

The list of loaded kernel extensions

The information in this output section can be compared to the output of the **genkex** command. The only difference is that the **gennames** command integrates the list of loaded kernel extensions information into the name-to-address mapping listing. Example 36-3 shows the part for one kernel extension.

Example 36-3 Output of gennames showing the loaded kernel extensions

```
# gennames

(... lines omitted ...)

/usr/lib/drivers/nfs.ext | 05988000 | 00000100 | 00077f58 | | initialize
Symbols from __nfs.ext
.__nfs.ext                |          | file |          |          |          |
.nfs_config               |          | 0 | extern |          |          | .text
.init_kacl                 |          | 340 | extern |          |          | .text
.init_serv                 |          | 376 | extern |          |          | .text
.init_clnt                 |          | 616 | extern |          |          | .text
.fill_nfs_syms            |          | 1460 | extern |          |          | .text
.init_nfs_syms            |          | 1524 | extern |          |          | .text
.init_krpc                 |          | 2008 | extern |          |          | .text
.nfs_ulimit_64            |          | 2856 | extern |          |          | .text

(... lines omitted ...)
```

In this example, the kernel extension `/usr/lib/drivers/nfs.ext` is loaded at address `0x5988000`. The size of the kernel extension is `0x77f58` bytes, and the text offset is `0x100`. This kernel extension file information is followed by the name-to-address mapping information for this kernel extension.

The list of loaded shared libraries

The information in this output section can be compared to the output of the **genkld** command. The only difference is that the **gennames** command integrates the list of loaded shared libraries information into the name-to-address mapping list. Example 36-4 shows part of the input for a shared library.

Example 36-4 Output of gennames showing the loaded shared libraries

```
# gennames
/usr/lib/libXm.a[shr_32.o] | d2640100 | 00000140 | 002548a1 | | initialize
Symbols from __shr_32.o
nl_langinfo               |          | 0 | extern |          |          |
setlocale                  |          | 0 | extern |          |          |

(... lines omitted ...)

dlsym                      |          | 0 | extern |          |          |
```

TOC	254084	unamex			.data
XtDisplayOfObject	254084	unamex			.data
XSetClipMask	254088	unamex			.data
XtWindowOfObject	254092	unamex			.data

(... lines omitted ...)

For this example, the same applies as for Example 36-3 on page 707. However, one small difference should be noted: The AIX operating system is using shared objects. In our example, the whole `/usr/lib/libXm.a` is not loaded here. Only the shared object module `shr_32.o` out of `/usr/lib/libXm.a` is loaded at the specific address. There are more shared objects in `/usr/lib/libXm.a`, and each of them is loaded separately as needed. The symbols with an offset value of zero, in the example above `n1_langinfo`, `setlocale` and `dlsym`, are references from `/usr/lib/libXm.a[shr_32.o]` to other shared objects. The `dump -nv /usr/lib/libXm.a | grep setlocale` command can be used to find the object file providing the `setlocale` function to `/usr/lib/libXm.a[shr_32.o]`.

The list of loaded processes

In this section of the `gennames` command output, the process IDs and the names of the processes currently running are listed. This section of the output of the `gennames` command can be compared with the output of the `genld` command. Example 36-5 shows a small section of this output.

Example 36-5 Output of `gennames` showing the loaded processes

```
# gennames

(... lines omitted ...)

Symbols from genld
Proc_pid:      0 Proc_name: swapper

Proc_pid:      1 Proc_name: init

Proc_pid:     516 Proc_name: wait

Proc_pid:     774 Proc_name: wait

Proc_pid:    1032 Proc_name: wait

Proc_pid:    1290 Proc_name: wait

(... lines omitted ...)
```

This example shows the list of processes in the output of the **gennames** command. For each process the process ID and the name are displayed. Note that kernel processes are included in this listing.

Physical and logical volume and file system information

The **gennames -f** command gathers additional data necessary for the offline processing by the **filemon** command. The following additional data is gathered (Example 36-6).

Example 36-6 Physical and logical volume and file system information gathered

Symbols from filesystems:

dev_id	path	mode	blocks	Description
a0000	/dev/___vg10	20600	1048576	
a0008	/dev/hd1	60660	5111808	/home Frag_Sz.= 512
a0009	/dev/hd10opt	60660	65536	/opt Frag_Sz.= 512
a0005	/dev/hd2	60660	1933312	/usr
a0007	/dev/hd3	60660	1114112	/tmp
a0004	/dev/hd4	60660	196608	/
a0001	/dev/hd5	60660	32768	boot
a0002	/dev/hd6	60660	2097152	paging
a0003	/dev/hd8	60660	32768	jfslog

(... lines omitted ...)

Symbols from vfs:

num	name	mount point
1	/dev/hd4	/
2	/dev/hd2	/usr
4	/dev/hd9var	/var
5	/dev/hd3	/tmp
6	/dev/hd1	/home
7	/proc	/proc
8	/dev/hd10opt	/opt
9	/dev/lv02	/audit
10	/dev/lv04	/work/fs1
11	/dev/lv05	/work/fs2
12	/dev/data1v	/data
13	/dev/lv00	/tools
14	/dev/lv01	/test0
15	/dev/lv08	/test1
16	/dev/lv09	/test2

The first part of the output shows the physical and logical volume information. The columns in this part are:

dev_id Contains the major and minor device number of the device. The values in the output are hexadecimal. The last

four digits are the minor number. The other digits are the major number. In the above example, dev_id a0003 belongs to the logical volume /dev/hd8 and has the major device number 10 (a) and the minor device number 3 (0003).

path	Shows the full path name of the device.
mode	Shows the file access mode bits of the device. The values are defined in the system header file /usr/include/sys/mode.h. In the example the 60660 for /dev/hd8 translates to: Block special device, read and write permission for owner and group.
blocks	Shows the size of the physical or logical volume in blocks of 512 bytes. The device /dev/hd8 in the example has the size 32768 * 512 = 16 MB.
Description	Shows the description for the physical or logical volume. The logical volume /dev/hd8 in the example is a jfs1og.

The second part of the output shown in Example 36-6 on page 709 contains the name of the logical volume and the mount point for each volume during execution time of the **gennames** command.

36.3 genld

The **genld** command has the following syntax:

```
genld [ -l ]
```

Flags

-l Reports the lists of loaded objects for each process running on the system.

36.3.1 Information about measurement and sampling

For each currently running process, the **genld** command prints a report consisting of the process ID and name. With the **-l** flag (in Version 5.2B), **genld** reports the loaded libraries.

36.3.2 Examples for genld

Example 36-7 shows output from **genld**.

Example 36-7 genld report

```
#genld
```

```

Proc_pid:      0      Proc_name: swapper
Proc_pid:      1      Proc_name: init
Proc_pid:     4644     Proc_name: reaper
Proc_pid:     4902     Proc_name: lrud
Proc_pid:     5160     Proc_name: xmgc
Proc_pid:     5418     Proc_name: netm
Proc_pid:     5676     Proc_name: gil
...(lines omitted)...

```

As a root user, you can also get the loaded libraries for each process. This feature requires you to install APAR IY43857 for AIX 5.1 or IY43859 for AIX 5.2. The result of **genld -l** is shown in Example 36-8.

Example 36-8 The output of genld -l

```

# genld -l
Proc_pid:      0      Proc_name: swapper

Proc_pid:      1      Proc_name: init
                   10000000  850e init
                   d0049000  2ba5d /usr/lib/libpthreads.a/shr_xpg5.o
                   d0045000  38c7 /usr/lib/libpthreads.a/shr_comm.o
                   d00750f8   846 /usr/lib/libcrypt.a/shr.o
                   d01d6e00 1f36e9 /usr/lib/libc.a/shr.o

Proc_pid:     77862     Proc_name: lrud
Proc_pid:     81960     Proc_name: xmdetd
Proc_pid:     86058     Proc_name: vmptacrt
Proc_pid:     90156     Proc_name: xmgc
Proc_pid:     94254     Proc_name: netm
Proc_pid:    127082     Proc_name: cron
                   10000000  988d cron
                   d0085100 13f01 /usr/lib/libiconv.a/shr4.o
                   d007c100  8a06 /usr/lib/libi18n.a/shr.o
                   d0076000  2d81 /usr/lib/nls/loc/en_US
                   d00750f8   846 /usr/lib/libcrypt.a/shr.o
                   d01d6e00 1f36e9 /usr/lib/libc.a/shr.o

```

```

Proc_pid: 135282 Proc_name: errdemon
1000000 15ff3 errdemon
d0134000 1c6f3 /usr/lib/boot/bin/libcfg_chrp
d0085100 13f01 /usr/lib/libiconv.a/shr4.o
d007c100 8a06 /usr/lib/libi18n.a/shr.o
d0076000 2d81 /usr/lib/nls/loc/en_US
d00ac100 f72c /usr/lib/libcfg.a/shr.o
d009a100 11da4 /usr/lib/libodm.a/shr.o
d00750f8 846 /usr/lib/libcrypt.a/shr.o
d01d6e00 1f36e9 /usr/lib/libc.a/shr.o

```

...(lines omitted)...

36.4 genkld

There is no flags or parameters for the **genkld** command.

36.4.1 Information about measurement and sampling

For shared objects loaded into the system, the kernel maintains a linked list consisting of data structures called loader entries. A loader entry contains the name of the object, its starting address, and its size. This information is gathered and reported by the **genkld** command.

36.4.2 Examples for genkld

Example 36-9 shows an example of running **genkld**.

Example 36-9 Using genkld

```

# genkld
Virtual Address          Size File
d23c6d60                2397c /usr/lib/libtrace.a/shr.o
d239b100                2a1e9 /usr/lib/libptools.a/shr.o
d00c0000                 284 /usr/lib/drivers/nfs.load/
d24583c0                287aa /usr/lib/libxcurses.a/shr4.o
d00e6000                28024 /usr/lib/libpthreads.a/shr.o
d00c50f8                 16ea /usr/lib/libpthreads_compat.a/shr.o
d242a100                2d1f4 /usr/lib/libSpmi.a/spmishr.o
d2410100                19961 /usr/lib/libwlm.a/shr.o
d05ce100                356f8 /usr/lib/liblvm.a/shr.o
d2398000                20c9 /usr/lib/nls/loc/C.im/
d238f100                869c /usr/lib/libi18n.a/shr.o
d00bd100                284a /usr/lib/libgair4.a/shr.o
d00bc0f8                 758 /usr/lib/libgaimisc.a/shr.o
d235a100                1e209 /usr/lib/libXext.a/shr.o

```

```

d00b2100          951e /usr/lib/libXi.a/shr.o
d2379100          1532c /usr/lib/libICE.a/shr.o
d00a8100          94d8 /usr/lib/libSM.a/shr.o
d0172100          4b18 /usr/lib/libIM.a/shr.o
d015e100          13ef1 /usr/lib/libiconv.a/shr4.o
d03b5100          f793b /usr/lib/libX11.a/shr4.o
...(lines omitted)...

```

The example contains the following columns:

Virtual Address	Start of the virtual address in memory (in hex) where the kernel extension resides. Use the svmon command for more detailed information about virtual addresses.
Size	Size in hex of the kernel extension.
File	File from which the kernel extension is loaded.

If path names end with / (slash), then they are shared objects (as in the output of the **file** command). If path names end without / (slash), then they are modules of shared libraries.

If some shared libraries and shared objects are loaded more than once, then there is an entry for each of them.

As kernel extensions may be loaded more than once, they may appear more than once with different virtual addresses and different sizes.

36.5 genkex

There are no flags or parameters for the **genkex** command.

36.5.1 Information about measurement and sampling

For kernel extensions loaded into the system, the kernel maintains a linked list consisting of data structures called loader entries. A loader entry contains the name of the extension, its starting address, and its size. This information is gathered and reported by the **genkex** command.

36.5.2 Examples for genkex

Example 36-10 shows the output from running **genkex**.

Example 36-10 genkex report

```

# genkex
Virtual Address          Size File

```

6a4000	24d870	/etc/drivers/zcmem_ke
216f300	6f40	/usr/lib/drivers/trcdd
67cb000	24d870	/etc/drivers/zcmem_ke
6557000	24d870	/etc/drivers/zcmem_ke
62e3000	24d870	/etc/drivers/zcmem_ke
216ba00	38f0	/usr/lib/drivers/bpf
2169c00	1de4	/usr/lib/drivers/netintf
2164c60	4f88	/usr/lib/drivers/perfvmmstat.mp
215b8a0	93b0	/usr/lib/drivers/random
618d000	7244c	/usr/lib/drivers/nfs.ext
215b550	334	/usr/lib/drivers/nfs_kdes.ext
215b298	2b0	/usr/lib/drivers/syscalls64.ext
5ecc000	24d870	/etc/drivers/zcmem_ke
21557e0	5ab0	/usr/lib/perf/perfstat
21534b0	2314	/usr/lib/drivers/aiopin
214c5a0	6f08	/usr/lib/drivers/aio.ext
214c3a8	1dc	/usr/lib/drivers/smt_loadpin
2148220	4180	/usr/lib/drivers/smt_load
2142c00	5614	/usr/lib/drivers/vconsdd
2132800	103e4	/usr/lib/drivers/ptydd
2120ca0	11b40	/usr/lib/drivers/ldterm
211ce20	3e70	/usr/lib/drivers/if_en
5ccf000	ed460	/usr/lib/drivers/netinet
211c680	78c	/etc/drivers/scsesddpin
211ab80	1ae4	/usr/lib/drivers/scsesdd
2116e60	3d14	/etc/drivers/tapepin
2111880	55c8	/usr/lib/drivers/tape
210f320	2548	/usr/lib/drivers/eth_demux
20f5a80	19898	/usr/lib/drivers/pci/scntdd
20f2940	311c	/etc/drivers/coreprobe.ext

... (lines omitted) ...

The columns in the example are:

Virtual Address	Start of the virtual address in memory (in hex) where the kernel extension resides.
Size	Size in hex of the kernel extension.
File	File from which the kernel extension is loaded.

As kernel extensions may be loaded more than once, they may appear more than once with different virtual addresses and different sizes, but only if the file has changed.

The **genkex** output is useful for determining who owns the extension. In Example 36-11, the `/usr/lib/drivers/pci/scntdd` kernel extension belongs to fileset `devices.pci.1410ff01.rte`. If you experience problems with this kernel extension, you can look at `devices.pci.1410ff01.rte` to determine the problem.

Example 36-11 Determining the owner of a kernel extension

```
# genkex | grep ent
          20f5a80          19898 /usr/lib/drivers/pci/scntdd
# lslpp -w /usr/lib/drivers/pci/scntdd
```

File	Fileset	Type
/usr/lib/drivers/pci/scntdd	devices.pci.1410ff01.rte	File

36.6 gensyms

The **gensyms** command has no parameters or flags.

36.6.1 Information about measurement and sampling

The **gensyms** command gathers name-to-address mapping information necessary for the **tprof** command to work in offline mode. The information gathered includes:

- ▶ the list of all loaded kernel extensions.
- ▶ the list of all loaded shared libraries.
- ▶ the list of all loaded processes.
- ▶ for `/unix`, all kernel extensions, libraries, and all object files corresponding to processes, the output of the **stripnm** command is collected.

Important: The output of **gensyms** can be very large. Redirect the output to a disk file.

36.6.2 Examples for gensyms

Example 36-12 shows output from **gensyms**.

Example 36-12 Sample output of gensyms

```
Symlib/AIX Version 5.1.1.1 -- Jan 24 2003 08:14:36 - gensyms format
Using load addresses for text symbols
Kernel: 00000000 0074698c 00000000 /unix
Kernel_data: 00746990 02483068 00000000
```

```

Kernel_validated:      1
Src: 64/low.s
00000000 t .low
00000000 T pin_obj_start
00000024 T start
00003000 T nonpriv_page
00003000 T g_copyr
00003100 T .mulh
00003180 T .mull
00003200 T .divss
00003280 T .divus
00003300 T .quoss
00003380 T .quous
00003400 T .clear_lock
00003410 T .clear_lock_mem
00003420 T .check_lock
00003460 T .cs
000034a0 T cs
000034a8 T .clear_lock
000034b0 T .check_lock
000034b8 T cmp_swap_index
000034c0 T g_ksrval
000034c8 T Trconflag
000034e0 T _system_configuration
000035dc T utrhook_sc_index
000035e0 T __extension_status
00003600 T _system_TB_config
00003608 T tod
. . . (lines omitted) . . .
Src: misc.s
00013818 T panic
Src: execexit.s
00013830 T execexit
Src: trchka64.s
00013848 T trchhook_on
Src: pinned_text_start.s
00014000 T pinned_text_start
Src: ../../../../src/bos/kernel/si/pin_si.c
00014000 T .si_bcopy
00014050 T .si_bzero
Src: ../../../../src/bos/kernel/proc/start.c
00014098 T .main_bs_proc
000141d8 T .strtwait_bs_proc
000142b4 T .init_sprgs
00014344 T .proc0init
0001453c T .proc0init_stack
000145f0 T .invoke_start_bs_proc
000148a8 T .init_cpu_flihs
00014988 T .init_flihs

```

```
00014a0c T .si_context_init
Src: misc_ppc.s
00014b48 T .mttb
Src: ../../../../src/bos/kernel/kdb/POWER/kdb_si.c
00014b5c T .trc_init
00014ba4 t .kdb_get_vtty
00014c04 t .kdb_free_dbg
00014c50 t .kdb_print_vmrmap
. . . (lines omitted) . . .
```

Archived

The locktrace command

The **locktrace** command determines which kernel locks will be traced by the trace subsystem. If a **bosboot -L** was run prior to rebooting the system, then kernel lock tracing can be enabled or disabled for one or more individual lock classes, or for all lock classes. If the **bosboot -L** was *not* run prior to a reboot, then lock class tracing can either be enabled or disabled for *all* locks. Where the **bosboot -aL** was not invoked, the trace events will not display the lock class names for taken, missed, and released locks. By default lock tracing performed by the trace subsystem is disabled.

The **locktrace** command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

Note: Before **locktrace** can be used, you must create, as root, a new boot image with the `-L` option to enable lock instrumentation. Run the command: **bosboot -a -d /dev/hdiskx -L**, where `x` is the number of the bootdisk.

37.1 locktrace

The syntax for the **locktrace** command is:

```
locktrace [ -r ClassName | -s ClassName | -S | -R | -l ]
```

Flags

- r classname** Turn off lock tracing for all kernel locks belonging to the specified class. This option always fails if **bosboot -aL** was not executed prior to a reboot.
- s classname** Turn on lock tracing for all kernel locks belonging to the specified class. This option always fails if **bosboot -aL** has not been executed prior to a reboot.
- R** Turn off all lock tracing.
- S** Turn on lock tracing for all locks regardless of their class membership.
- l** List kernel lock tracing current status.

37.1.1 Information about measurement and sampling

The tracing of locks can be extremely useful in providing crucial information about how locks are being used, which are hot, which are in contention, and which are degrading the system. However, trace hooks in already heavily used lock routines cause the system to slow down. The **locktrace** command allows lock trace hooks to be effectively inserted or completely removed dynamically from the lock routines. If the system has been rebooted after a **bosboot -L**, finer selectivity is provided by enabling and disabling lock trace hooks for specific classes. This results in less or at least controlled system degradation from trace hooks in a lock routine. The names of the lock classes can be found in the file `lockname.h`, which is located in the `/usr/include/sys` directory. Example 37-1 shows a list of some of the class names that can be found in this file.

Example 37-1 An extract from the lockname.h file

```
...(lines omitted)...
#define MSG_LOCK_CLASS      120    /* IPC */
#define SEM_LOCK_CLASS      121    /* IPC */
#define SHM_LOCK_CLASS      122    /* IPC */
#define DEVNODE_LOCK_CLASS  123    /* LFS */
#define FFREE_LOCK_CLASS    124    /* LFS */
#define FIFOBUF_LOCK_CLASS  125    /* LFS */
#define FILOCK_LOCK_CLASS   126    /* LFS */
#define FOFF_LOCK_CLASS     127    /* LFS */
```

```
#define FPTR_LOCK_CLASS      128    /* LFS */
#define GFS_LOCK_CLASS      129    /* LFS */
#define GPA_LOCK_CLASS      130    /* LFS */
#define PATH_LOCK_CLASS     131    /* LFS */
#define U_FD_CLASS          132    /* LFS */
#define U_FSO_CLASS         133    /* LFS */
#define VFS_LIST_LOCK_CLASS 134    /* LFS */
#define VFS_LOCK_CLASS      135    /* LFS */
#define VNODE_LOCK_CLASS    136    /* LFS */
...(lines omitted)...
```

37.1.2 Examples for locktrace

Example 37-2 shows the output when the **locktrace** command is run without the **bosboot -L** command being run first and without the system being rebooted.

Example 37-2 locktrace error message when bosboot -L has not been run

```
# locktrace -s MSG_LOCK_CLASS
locktrace: selective tracing not allowed without bosboot -L
```

Example 37-3 shows the use of the **locktrace** command to enable the trace subsystem to trace the **SEM_LOCK_CLASS** lock type. The **bosboot -aL** command has been run on the system and it has been rebooted prior to the **locktrace** commands being run. The **locktrace -l** option shows which locks are enabled for tracing. In the first case, all lock tracing is disabled. The **locktrace -s SEM_LOCK_CLASS** command was used to enable the tracing of the **SEM_LOCK_CLASS** lock class type. The **locktrace -l** command was used again to check which lock classes are enabled for tracing.

Example 37-3 Enabling tracing of the SEM_LOCK_CLASS type

```
# locktrace -l
lock tracing disabled for all classes

# locktrace -s SEM_LOCK_CLASS
lock tracing enabled for class SEM_LOCK_CLASS

# locktrace -l
lock tracing enabled for classes:
  SEM_LOCK_CLASS
```

To disable the tracing of this class, use the command in Example 37-4. The **locktrace -r SEM_LOCK_CLASS** command is used to disable the tracing of the SEM_LOCK_CLASS lock type. The **locktrace -R** command could have been used, but would disable tracing of all lock classes.

Example 37-4 Turning off tracing of the SEM_LOCK_CLASS

```
# locktrace -r SEM_LOCK_CLASS  
lock tracing disabled for class SEM_LOCK_CLASS
```

```
# locktrace -l  
lock tracing disabled for all classes
```

The stripnm command

The **stripnm** command extracts the symbol information from a specified object file, executable, or archive library and prints it to standard output. If the input file is an archive library, the command extracts the symbol information from each object file contained in the archive. It can also be used to search for symbol information in the /unix file. If the /unix file does not correspond to the currently running kernel, a warning message is displayed. The **stripnm** command produces an output similar to the output generated by the **gennames** command, which is required for using the **tprof**, **filemon**, **netpmon**, and **pprof** commands in real-time mode.

The **stripnm** command is similar to the **nm** command. However, it can extract symbol information from striped executables whereas **nm** cannot.

The **stripnm** command resides in /usr/bin and is part of the bos.perf.tools fileset, which is installable from the AIX base installation media.

38.1 stripnm

The syntax of **stripnm** is:

```
stripnm [-x|d] [ -s ] [ -z ] File
```

Flags

-x	Prints symbol address values in hexadecimal format.
-d	Prints symbol address values in decimal format. This is the default with -z .
-s	Forces to ignore symbol table.
-z	Use the old format.

Parameter

File The name of the object or archive library file.

38.1.1 Information about measurement and sampling

If an executable is produced with optimization (**-O**), some of the information used by **stripnm** is not included. In order for **stripnm** to work correctly with stripped optimized executables, the **-q ttable=full** compiler option should be used in addition to the **-O** option when compiling the executable.

The **stripnm** command (when run without the **-s** flag) prints the symbol table of a specified object file to standard output. The file specified by the **File** parameter can be a single object file or an archive library of object files. If the file specified by the **File** parameter is an archive, a listing for each object file in the archive is produced. If the symbol table has been stripped from the object file, the **stripnm** command extracts symbol names from the traceback tables (even if the **-s** flag is not specified) and the loader section of the object file(s). If the traceback tables do not exist, an error message is displayed.

Each symbol name is preceded by its address and one character representing the symbol type (similar to **nm** output). When used with **-z**, the output format is the same as it was before AIX 5.2; that is, each symbol name is followed by its address (a series of blanks if the address is undefined) and the type of class and section type. The address field can be displayed as a decimal (the default value with **-z**, or when **-d** is used) or hexadecimal (the default value without **-z**, or if the **-x** flag is used).

Source file names are also collected and reported by the **stripnm** command. All of the symbols following a source file name line belong to the same source file

until the next source file name line is encountered. For striped files, the source file name is reported as being the object file name.

When run using the `-s` flag, the `stripnm` command ignores the symbol table if present and always extracts routine names from the traceback tables and the loader section of the object file(s).

When no symbol table is present or the `-s` flag is used, the `stripnm` command also searches for glue code and pointer glue information. Both are sequences of instructions found in the text section of the object file.

38.2 Examples for stripnm

Example 38-1 shows the output of the `stripnm -sx /usr/bin/ls` command.

Example 38-1 The output of the stripnm -sx /usr/bin/ls command

```
# stripnm -sx /usr/bin/ls
....(lines omitted)....
Using load addresses for text symbols
File-32: 10000000 10003aa8 00000100 /usr/bin/ls
Src: /usr/bin/ls
10000100 t __start
100001c8 t __threads_init
Src: glink.s
100002e0 t __mod_init
Src: /usr/bin/ls
10000308 t .add_cache
10000448 t .cache_hit
10000500 t .ls_select
100005f4 t .getname
1000070c t .pmode
10000780 t .column
100009e0 t .pentry
100012c4 t .new_line
10001364 t .pprintf
10001650 t .pdirectory
1000192c t .pem
10001ac8 t .main
100023ec t .compar
10002530 t .makename
10002684 t .readdir
10002860 t .savestr
1000291c t .gstat
Src: glink.s
10002fc0 t .malloc
10002fe8 t .catgets
```

```

10003010 t .fprintf
10003038 t .exit
10003060 t .free
10003088 t .__flsbuf
100030b0 t .__getpwuid_shadow
100030d8 t .getgrgid
Src: /usr/bin/ls
10003100 t .strcpy
Src: glink.s
..... (lines omitted) .....

```

The example above uses the following fields:

- ▶ The first column shows the address in hexadecimal format.
- ▶ The second column shows the type of symbol, which can be:

A	Global absolute symbol
a	Local absolute symbol
B	Global bss symbol
b	Local bss symbol
D	Global data symbol
d	Local data symbol
T	Global text symbol
t	Local text symbol
U	Undefined symbol

- ▶ The third column shows the symbol name.

Example 38-2 shows the use of “-z” flag to display output in old format.

Example 38-2 Old format output using -z on the stripnm command

```

# stripnm -zsx /usr/bin/ls
Symbols from /usr/bin/ls

```

/usr/bin/ls		file			
._start	0x10000100	unamex			.text
._threads_init	0x100001c8	unamex			.text
glink.s		file			
._mod_init	0x100002e0	unamex			.text
/usr/bin/ls		file			
.add_cache	0x10000308	unamex			.text
.cache_hit	0x10000448	unamex			.text
.ls_select	0x10000500	unamex			.text
.getname	0x100005f4	unamex			.text
.pmode	0x1000070c	unamex			.text
.column	0x10000780	unamex			.text
.penry	0x100009e0	unamex			.text
.new_line	0x100012c4	unamex			.text

.pprintf	0x10001364	unamex		.text
.pdirectory	0x10001650	unamex		.text
.pem	0x1000192c	unamex		.text
.main	0x10001ac8	unamex		.text
.compar	0x100023ec	unamex		.text
.makename	0x10002530	unamex		.text
.readdirs	0x10002684	unamex		.text
.savestr	0x10002860	unamex		.text
.gstat	0x1000291c	unamex		.text
glink.s		file		
.malloc	0x10002fc0	unamex		.text
.catgets	0x10002fe8	unamex		.text
.fprintf	0x10003010	unamex		.text
.exit	0x10003038	unamex		.text
....(lines omitted).....				

This example has the following fields:

- ▶ The first column shows the name of the symbol.
- ▶ The second column lists the address of the symbol. No address is displayed if the reference of the symbol is `file`.
- ▶ The third column shows the symbol's reference. The possible values are:

<code>file</code>	The symbol references a file name.
<code>extern</code>	The symbol references a named external symbol.
<code>unamex</code>	The symbol references an unnamed external symbol.
- ▶ The last column shows the section of the object the symbol belongs to. The possible values are:

<code>.text</code>	The text (code) section.
<code>.data</code>	The data section.
<code>.bss</code>	The uninitialized data section.

For detailed information about the Extended Common Object File Format (XCOFF), refer to the *AIX 5L Version 5.2 Files Reference*.

Archived

The splat command

The Simple Performance Lock Analysis Tool (**splat**) is a software tool that generates reports on the use of synchronization locks. These include the simple and complex locks provided by the AIX kernel as well as user-level mutexes, read/write locks, and condition variables provided by the PThread library. **splat** is not currently equipped to analyze the behavior of the VMM- and PMAP- locks used in the AIX kernel.

The **splat** command resides in `/usr/bin` and is part of the `bos.perf.tools` fileset, which is installable from the AIX base installation media.

39.1 splat

The syntax for the **splat** command is:

```
splat -i file [-n file] [-o file] [-d [ bfta ] ] [-l address] [-c class]
      [-s [ aceImS ] ] [-C cpus] [-S count] [-t start] [-T stop]
splat -h [topic]
splat -j
```

Flags

-i inputfile	Specifies the AIX trace log file input.
-n namefile	Specifies the file containing output of gennames or gensyms command.
-o outputfile	Specifies an output file (default is stdout).
-d detail	Specifies the level of detail of the report.
-c class	Specifies class of locks to be reported.
-l address	Specifies the address for which activity on the lock will be reported.
-s criteria	Specifies the sort order of the lock, function, and thread.
-C CPUs	Specifies the number of CPUs on the MP system that the trace was drawn from. The default is one. This value is overridden if more CPUs are observed to be reported in the trace.
-S count	Specifies the number of items to report on for each section. The default is 10. This gives the number of locks to report in the Lock Summary and Lock Detail reports, as well as the number of functions to report in the Function Detail and threads to report in the Thread detail. (The -s option specifies how the most significant locks, threads, and functions are selected.)
-t starttime	Overrides the start time from the first event recorded in the trace. This flag forces the analysis to begin an event that occurs starttime seconds after the first event in the trace.
-T stoptime	Overrides the stop time from the last event recorded in the trace. This flag forces the analysis to end with an event that occurs stoptime seconds after the first event in the trace.
-j	Prints the list of IDs of the trace hooks used by splat .
-h topic	Prints a help message on usage or a specific topic.

Parameters

inputfile	The AIX trace log file input. This file can be a merge trace file generated using trcrpt -r .
namefile	File containing output of gennames or gensyms command.
outputfile	File to write reports to.
detail	The detail level of the report; can be either: basic lock summary plus lock detail (the default) function basic + function detail thread basic + thread detail all basic + function + thread detail
class	Activity classes, which is a decimal value found in the file <code>/usr/include/sys/lockname.h</code> .
address	The address to be reported, given in hexadecimal.
criteria	Order the lock, function, and thread reports by the following criteria: a Acquisitions c Percent CPU time held e Percent elapsed time held l Lock address, function address, or thread ID m Miss rate s Spin count S Percent CPU spin hold time (the default) w Percent real wait time W Average WaitQ depth
CPUs	The number of CPUs on the MP system that the trace was drawn from. The default is one. This value is overridden if more CPUs are observed to be reported in the trace.
count	The number of locks to report in the Lock Summary and Lock Detail reports, as well as the number of functions to report in the Function Detail and threads to report in the Thread detail. (The -s option specifies how the most significant locks, threads, and functions are selected).
starttime	The number of seconds after the first event recorded in the trace that the reporting starts.
stoptime	The number of seconds after the first event recorded in the trace that the reporting stops.

topic

Help topics, which are:

- ▶ all
- ▶ overview
- ▶ input
- ▶ names
- ▶ reports
- ▶ sorting

39.1.1 Information about measurement and sampling

The **splat** command takes as input AIX trace log file or a set of log files for an SMP trace, and preferably a names file produced by **gennames**. The procedure for generating these files is shown in Chapter 40, “The trace, trcnm, and trcrpt commands” on page 759. When you run **trace** you will usually use the flag **-J splat** to capture the events analyzed by **splat** (or no **-J** flag, to capture all events). The important trace hooks are shown in Table 39-1.

Table 39-1 Trace hooks required for splat

Hook ID	Event name	Event explanation
106	HKWD_KERN_DISPATCH	The thread is dispatched from the runqueue to a CPU.
10C	HKWD_KERN_IDLE	The idle process is been dispatched.
10E	HKWD_KERN_RELOCK	One thread is suspended while another is dispatched; the ownership of a RunQ lock is transferred from the first to the second.
112	HKWD_KERN_LOCK	The thread attempts to secure a kernel lock; the subhook shows what happened.
113	HKWD_KERN_UNLOCK	A kernel lock is released.
38F		Dynamic reconfiguration
46D	HKWD_KERN_WAITLOCK	The thread is enqueued to wait on a kernel lock.
600	HKWD_PTHREAD_SCHEDULER	Operations on a Scheduler Variable.
603	HKWD_PTHREAD_TIMER	Operations on a Timer Variable.
605	HKWD_PTHREAD_VPSLEEP	Operations on a Vpsleep Variable.
606	HKWD_PTHREAD_CONDS	Operations on a Condition Variable.

Hook ID	Event name	Event explanation
607	HKWD_PTHREAD_MUTEX	Operations on a Mutex.
608	HKWD_PTHREAD_RWLOCK	Operations on a Read/Write Lock.
609	HKWD_PTHREAD_GENERAL	Operations on a PThread.

The execution, trace, and analysis intervals

In some cases you can use **trace** to capture the entire execution of a workload, while other times you will only capture an interval of the execution. We distinguish these as the *execution interval* and the trace interval. The execution interval is the entire time that a workload runs. This interval is arbitrarily long for server workloads that run continuously. The trace interval is the time actually captured in the trace log file by **trace**. The length of this trace interval is limited by how large of a trace log file will fit on the filesystem.

In contrast, the *analysis interval* is the portion of time that is analyzed by **splat**. The **-t** and **-T** options tell **splat** to start and finish analysis some number of seconds after the first event in the trace. By default **splat** analyzes the entire trace, so this analysis interval is the same as the trace interval. Example 39-1 on page 735 shows the reporting of the trace and analysis intervals.

Note: As an optimization, **splat** stops reading the trace when it finishes its analysis, so it will report the trace and analysis intervals as ending at the same time even if they do not.

You will usually want to capture the longest trace interval you can and analyze the entire interval with **splat** in order to most accurately estimate the effect of lock activity on the computation. The **-t** and **-T** options are usually used for debugging purposes to study the behavior of **splat** across a few events in the trace.

As a rule, either use large buffers when collecting a trace, or limit the captured events to the ones needed to run **splat**.

Trace discontinuities

The **splat** command uses the events in the trace to reconstruct the activities of threads and locks in the original system. It will not be able to correctly analyze all of the events across the trace interval if part of the trace is missing because:

- ▶ Tracing was stopped at one point and restarted at a later point.
- ▶ One CPU fills its trace buffer and stops tracing, while other CPUs continue tracing.

- ▶ Event records in the trace buffer were overwritten before they could be copied into the trace log file.

The policy of **splat** is to finish its analysis at the first point of discontinuity in the trace, issue a warning message, and generate its report. In the first two cases the warning message is:

```
TRACE OFF record read at 0.567201 seconds. One or more of the CPU's has
stopped tracing. You may want to generate a longer trace using larger
buffers and re-run splat.
```

In the third case the warning message is:

```
TRACEBUFFER WRAPAROUND record read at 0.567201 seconds. The input trace has
some records missing; splat finishes analyzing at this point. You may want
to re-generate the trace using larger buffers and re-run splat.
```

Along the same lines, versions of the AIX kernel or PThread library that are still under development may be incompletely instrumented, and so the traces will be missing events. **splat** may not give correct results in this case.

Address-to-name resolution in splat

The lock instrumentation in the kernel and PThread library captures the information for each lock event. Data addresses are used to identify locks; instruction addresses are used to identify the point of execution. These addresses are captured in the event records in the trace and used by **splat** to identify the locks and the functions that operate on them.

However, these addresses are of little use for the programmer, who would rather know the *names* of the lock and function declarations so they can be located in the program source files. The conversion of names to addresses is determined by the compiler and loader and can be captured in a file using the **gennames** or **gensyms** utility. **gennames** also captures the contents of the file `/usr/include/sys/lockname.h`, which declares classes of kernel locks. **gensyms** captures the address to name translation of kernel and subroutines.

This **gennames** or **gensyms** output file is passed to **splat** with the `-n` option. When **splat** reports on a kernel lock, it provides the best identification it can. A **splat** lock summary is shown in Example 39-3 on page 737; the left column identifies each lock by name if it can be determined, otherwise by class if it can be determined, or by address if nothing better can be provided. The lock detail shown in Example 39-4 on page 740 identifies the lock by as much of this information as can be determined.

Kernel locks that are declared will be resolved by *name*. Locks that are created dynamically will be identified by class if their class name is given when they are created. Note that the `libpthreads.a` instrumentation is not equipped to capture

names or classes of PThread synchronizers, so they are always identified only by address.

39.2 Examples for splat

The report generated by **splat** consists of an execution summary, a gross lock summary, and a per-lock summary, followed by a list of lock detail reports that optionally includes a function detail and/or a thread detail report.

39.2.1 Execution summary

Example 39-1 shows a sample of the Execution summary. This report is generated by default when using **splat**.

Example 39-1 Execution summary report

splat Cmd: splat -sa -da -S100 -i trace.rpt -n gennames.out -o splat.out

Trace Cmd: trace -C all -aj 600,603,605,606,607,608,609 -T 2000000 -L 20000000
-o trace.bin

Trace Host: lpar05 (0021768A4C00) AIX 5.2

Trace Date: Mon Apr 21 09:55:22 2003

Elapsed Real Time:18.330873

Number of CPUs Traced: 1 (Observed):0

Cumulative CPU Time:18.330873

		start	stop
		-----	-----
trace interval	(absolute tics)	1799170309	2104623072
	(relative tics)	0	305452763
	(absolute secs)	107.972055	126.302928
	(relative secs)	0.000000	18.330873
analysis interval	(absolute tics)	1799170309	2104623072
	(trace-relative tics)	0	305452763
	(self-relative tics)	0	305452763
	(absolute secs)	107.972055	126.302928
	(trace-relative secs)	0.000000	18.330873
	(self-relative secs)	0.000000	18.330873

The execution summary consists of the following elements:

- ▶ The command used to run **splat**.
- ▶ The **trace** command used to collect the trace.

- ▶ The host that the trace was taken on.
- ▶ The date that the trace was taken on.
- ▶ The real-time duration of the trace in seconds.
- ▶ The maximum number of CPUs that were observed in the trace, the number specified in the trace conditions information, and the number specified on the **splat** command line. If the number specified in the header or command line is less, the entry (Indicated: <value>) is listed. If the number observed in the trace is less, the entry (Observed: <value>) is listed.
- ▶ The cumulative CPU time, equal to the duration of the trace in seconds times the number of CPUs that represents the total number of seconds of CPU time consumed.
- ▶ A table containing the start and stop times of the trace interval, measured in tics and seconds, as absolute time stamps from the trace records, as well as relative to the first event in the trace. This is followed by the start and stop times of the analysis interval, measured in tics and seconds, as absolute time stamps as well as relative to the beginning of the trace interval and the beginning of the analysis interval.

39.2.2 Gross lock summary

Example 39-2 shows a sample of the gross lock summary report. This report is generated by default when using **splat**.

Example 39-2 Gross lock summary

	Total	Unique Addresses	Acquisitions (or Passes)	Acq. or Passes per Second	% Total System 'spin' Time
AIX (all) Locks:	523	523	1323045	72175.7768	0.003986
RunQ:	2	2	487178	26576.9121	0.000000
Simple:	480	480	824898	45000.4754	0.003986
Complex:	41	41	10969	598.3894	0.000000
PThread CondVar:	7	6	160623	8762.4305	0.000000
Mutex:	128	116	1927771	105165.2585	10.280745 *
RWLock:	0	0	0	0.0000	0.000000

('spin' time goal <10%)

The gross lock summary report table consists of the following columns:

Total	The number of AIX Kernel locks, followed by the number of each type of AIX Kernel lock; RunQ, Simple, and Complex. Under some conditions this will be larger than the sum of the numbers of RunQ, Simple, and Complex
-------	---

locks because we may not observe enough activity on a lock to differentiate its type. This is followed by the number of PThread condition variables, the number of PThread Mutexes, and the number of PThread Read/Write Locks.

Unique Addresses	The number of unique addresses observed for each synchronizer type. Under some conditions a lock will be destroyed and re-created at the same address; splat produces a separate lock detail report for each instance because the usage may be quite different.
Acquisitions (or Passes)	For locks, the total number of times <i>acquired</i> during the analysis interval; for PThread condition-variables, the total number of times the condition <i>passed</i> during the analysis interval.
Acq. or Passes (per second)	Acquisitions or passes per second, which is the total number of acquisitions or passes divided by the elapsed real time of the trace.
% Total System 'spin' Time	The cumulative time spent spinning on each synchronizer type, divided by the <i>cumulative CPU time</i> , times 100 percent. The general goal is to spin for less than 10 percent of the CPU time; a message to this effect is printed at the bottom of the table. If any of the entries in this column exceed 10 percent, they are marked with an asterisk (*).

39.2.3 Per-lock summary

Example 39-3 shows a sample of the per-lock summary report. This report is generated by default when using **splat**.

Example 39-3 Per-lock summary report

100 max entries, Summary sorted by Acquisitions:

Lock Names, Class, or Address	Type	Acqui- sitions or Passes	Spins	Wait	%Miss	%Total	Locks or Passes / CSec	Real CPU	Percent Real Elapse	Holdtime Comb Spin	Kernel Symbol
*****	*	*****	****	****	*****	*****	*****	*****	*****	*****	*****
PROC_INT_CLASS.0003	Q	486490	0	0	0.0000	36.7705	26539.380	5.3532	100.000	0.0000	unix
THREAD_LOCK_CLASS.0012	S	323277	0	0	0.0000	24.4343	17635.658	6.8216	6.8216	0.0000	libc
THREAD_LOCK_CLASS.0118	S	323094	0	0	0.0000	24.4205	17625.674	6.7887	6.7887	0.0000	libc
ELIST_CLASS.003C	S	80453	0	0	0.0000	6.0809	4388.934	1.0564	1.0564	0.0000	unix
ELIST_CLASS.0044	S	80419	0	0	0.0000	6.0783	4387.080	1.1299	1.1299	0.0000	unix
tod_lock	C	10229	0	0	0.0000	0.7731	558.020	0.2212	0.2212	0.0000	unix
LDATA_CONTROL_LOCK.0000	S	1833	0	0	0.0000	0.1385	99.995	0.0204	0.0204	0.0000	unix
U_TIMER_CLASS.0014	S	1514	0	0	0.0000	0.1144	82.593	0.0536	0.0536	0.0000	netinet
(... lines omitted ...)											
00000002FF22B70	L	368838	0	N/A	0.0000	100.000	9622.964	99.9865	99.9865	0.0000	
0000000F00C3D74	M	160625	0	0	0.0000	14.2831	8762.540	99.7702	99.7702	0.0000	

00000000200017E8	M	160625	175	0	0.1088	14.2831	8762.540	42.9371	42.9371	0.1487
0000000020001820	V	160623	0	624	0.0000	100.000	1271.728	N/A	N/A	N/A
00000000F00C3750	M	37	0	0	0.0000	0.0033	2.018	0.0037	0.0037	0.0000
00000000F00C3800	M	30	0	0	0.0000	0.0027	1.637	0.0698	0.0698	0.0000

(... lines omitted ...)

The first line indicates the maximum number of locks to report (100 in this case, but we only show 13 of the entries here) as specified by the `-S 100` flag. It also indicates that the entries are sorted by the total number of acquisitions or passes, as specified by the `-sa` flag. Note that the various Kernel locks and PThread synchronizers are treated as *two separate lists* in this report, so you would get the top 100 Kernel locks sorted by acquisitions, followed by the top 100 PThread synchronizers sorted by acquisitions or passes.

The per-lock summary table consists of the following columns:

Lock Names, Class, or Address	The name, class, or address of the lock, depending on whether <code>splat</code> could map the address from a name file. See "Address-to-name resolution in <code>splat</code> " on page 734 for an explanation.
Type	The type of the lock, identified by one of the following letters: <ul style="list-style-type: none"> Q A RunQ lock S A simple kernel lock C A complex kernel lock M A Pthread mutex V A Pthread condition-variable L A Pthread read/write lock
Acquisitions or Passes	The number of times the lock was acquired or the condition passed during the analysis interval.
Spins	The number of times the lock (or condition-variable) was spun on during the analysis interval.
Wait	The number of times a thread was driven into a <i>wait</i> state for that lock or condition-variable during the analysis interval.
%Miss	The percentage of access attempts that resulted in a <i>spin</i> as opposed to a successful acquisition or pass.
%Total	The percentage of all acquisitions that were made to this lock, out of all acquisitions to all locks of this type. Note that all AIX locks (RunQ, simple, and complex) are treated as being the same type for this calculation. The PThread

synchronizers mutex, condition-variable, and read/write lock are all distinct types.

Locks or Passes / CSec	The number of times the lock (or condition-variable) was acquired (or passed) divided by the <i>cumulative CPU time</i> . This is a measure of the acquisition frequency of the lock.
Real CPU	The percentage of the <i>cumulative CPU time</i> that the lock was held by an executing thread. Note that this definition is not applicable to condition-variables because they are not held.
Real Elapse	The percentage of the <i>elapsed real time</i> that the lock was held by any thread at all, whether running or suspended. Note that this definition is not applicable to condition-variables because they are not held.
Comb Spin	The percentage of the <i>cumulative CPU time</i> that executing threads spent spinning on the lock. Note that the PThreads library currently uses <i>waiting</i> for condition-variables, so there is no time actually spent spinning.
Kernel Symbol	The name of the kernel-extension or library (or /unix for the kernel) that the lock was defined in. Note that this information is <i>not recoverable</i> for PThreads.

39.2.4 AIX kernel lock details

By default, `splat` prints out a lock detail report for each entry in the summary report. There are two types of AIX Kernel locks: *simple* and *complex*. We will start by examining the contents of the simple lock report, and follow this with an explanation of the additional information printed with a complex lock report.

The RunQ lock is a special case of the simple lock, although its pattern of usage differs markedly from other lock types. `splat` distinguishes it from the other simple locks to save you the trouble of figuring out why it behaves so uniquely.

Simple- and RunQ- lock details

Example 39-4 on page 740 shows a sample AIX SIMPLE lock report. The first line starts with either [AIX SIMPLE Lock] or [AIX RunQ lock]. Below this is the 16-digit hexadecimal ADDRESS of the lock. If the gennames output-file allows, the ADDRESS is also converted into a lock NAME and CLASS, and the containing kernel-extension (KEX) is identified as well. The CLASS is printed with an eight-hex-digit extension indicating how many locks of this class were allocated prior to it.

Example 39-4 AIX SIMPLE lock

```

[AIX SIMPLE Lock]                CLASS: NETISR_LOCK_FAMILY.FFFFFFFF
ADDRESS: 0000000000535378        KEX: unix
NAME:      netisr_slock
=====
Acqui- | Miss Spin  Wait  Busy |      Secs Held      | Percent Held ( 18.330873s )
sitions | Rate Count Count Count | CPU   Elapsed      | Real  Real  Comb Real
471    | 0.000 0    0    0    | 0.002584 0.002584 | CPU  Elapsed Spin Wait
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----
%Enabled 0.00 ( 0 )|SpinQ  Min  Max  Avg | WaitQ  Min  Max  Avg
%Disabled 100.00 ( 471)|Depth 0    0    0  | Depth 0    0    0
-----|-----|-----|-----|-----|-----|-----|-----|-----|-----

```

Lock Activity w/Interrupts Enabled (mSecs)

SIMPLE	Count	Minimum	Maximum	Average	Total
LOCK	0	0.000000	0.000000	0.000000	0.000000
SPIN	0	0.000000	0.000000	0.000000	0.000000
UNDISP	0	0.000000	0.000000	0.000000	0.000000
WAIT	0	0.000000	0.000000	0.000000	0.000000
PREEMPT	0	0.000000	0.000000	0.000000	0.000000

Lock Activity w/Interrupts Disabled (mSecs)

SIMPLE	Count	Minimum	Maximum	Average	Total
LOCK	471	0.001200	0.019684	0.005486	2.583943
SPIN	0	0.000000	0.000000	0.000000	0.000000
UNDISP	0	0.000000	0.000000	0.000000	0.000000
WAIT	0	0.000000	0.000000	0.000000	0.000000
PREEMPT	0	0.000000	0.000000	0.000000	0.000000

The statistics are:

- Acquisitions The number of times the lock was acquired in the analysis interval (this includes successful simple_lock_try() calls).
- Miss Rate The percentage of attempts that failed to acquire the lock.
- Spin Count The number of unsuccessful attempts to acquire the lock.
- Wait Count The number of times a thread was forced into suspended wait state waiting for the lock to come available.
- Busy Count The number of simple_lock_try() calls that returned busy.

Seconds Held	This field contains the following subfields:
CPU	The total number of CPU seconds that the lock was held by an executing thread.
Elapsed	The total number of elapsed seconds that the lock was held by any thread at all, whether running or suspended.
Percent Held	This field contains the following subfields:
Real CPU	The percentage of the cumulative CPU time that the lock was held by an executing thread.
Real Elapsed	The percentage of the elapsed real time that the lock was held by any thread at all, either running or suspended.
Comb(ined) Spin	The percentage of the cumulative CPU time that running threads spent spinning while trying to acquire this lock.
Real Wait	The percentage of elapsed real time that any thread waited to acquire this lock. Note that if two or more threads are waiting simultaneously, this wait time will only be charged once. If you want to know how many threads were waiting simultaneously, look at the WaitQ Depth statistics.
%Enabled	The percentage of acquisitions of this lock that occurred while interrupts were enabled. The total number of acquisitions made while interrupts were enabled is in parentheses.
%Disabled	The percentage of acquisitions of this lock that occurred while interrupts were disabled. In parentheses is the total number of acquisitions made while interrupts were disabled.
SpinQ	The minimum, maximum, and average number of threads <i>spinning</i> on the lock, whether executing or suspended, across the analysis interval.
WaitQ	The minimum, maximum, and average number of threads <i>waiting</i> on the lock, across the analysis interval.

The Lock Activity with Interrupts Enabled (mSecs) and Lock Activity with Interrupts Disabled (mSecs) sections contain information about the time each lock state is used by the locks.

Figure 39-1 shows the states that a thread can be in with respect to the given simple or complex lock.

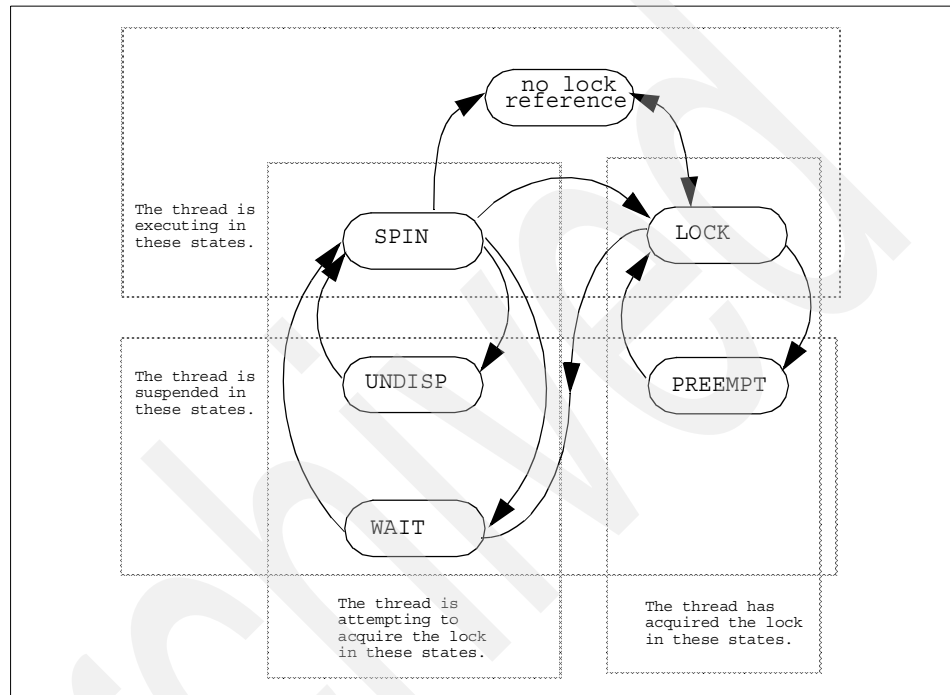


Figure 39-1 Lock states

The states are defined as follows:

- (no lock reference) The thread is running, does not hold this lock, and is not attempting to acquire this lock.
- LOCK The thread has successfully acquired the lock and is currently executing.
- SPIN The thread is executing and unsuccessfully attempting to acquire the lock.
- UNDISP The thread has become undispached while unsuccessfully attempting to acquire the lock.
- WAIT The thread has been suspended until the lock comes available. It does not necessarily acquire the lock at that time, instead going back to a SPIN state.

PREEMPT The thread is holding this lock and has become undispached.

The Lock Activity sections of the report measure the intervals of time (in milliseconds) that each thread spends in each of the states for this lock. The columns report the number of times that a thread entered the given state, followed by the maximum, minimum, and average time that a thread spent in the state once entered, followed by the total time all threads spent in that state. These sections distinguish whether interrupts were enabled or disabled at the time the thread was in the given state.

A thread can acquire a lock prior to the beginning of the analysis interval and release the lock during the analysis interval. When **splat** observes the lock being released, it recognizes that the lock had been held during the analysis interval up to that point and counts the time as part of the state-machine statistics. For this reason the state-machine statistics can report that the number of times that the LOCK state was entered may actually be larger than the number of acquisitions of the lock that were observed in the analysis interval.

RunQ locks are used to protect resources in the thread management logic. These locks are acquired a large number of times and are only held briefly each time. A thread does not necessarily need to be executing to acquire or release a RunQ lock. Further, a thread may spin on a RunQ lock, but it will not go into an UNDISP or WAIT state on the lock. You will see a dramatic difference between the statistics for RunQ versus other simple locks.

Function detail

Example 39-5 is an example of the function detail report. This report is obtained by using the **-df** or **-da** options of **splat**. Note that we have split the three right columns here and moved them below the table.

Example 39-5 Function detail report for the simple lock report

Function Name	Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Percent CPU	Held of Elapse	Total Spin	Time Wait
~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~~	~~~~~
._thread_unlock	80351	0.00	0	0	0	1.13	1.13	0.00	0.00
.thread_waitlock	68	0.00	0	0	0	0.00	0.00	0.00	0.00

Return Address	Start Address	Offset
~~~~~	~~~~~	~~~~~
00000000001AA54	000000000000000	0001AA54
00000000001A494	000000000000000	0001A494

The columns are defined as follows:

Function Name	The name of the function that acquired or attempted to acquire this lock (with a call to one of the functions <code>simple_lock</code> , <code>simple_lock_try</code> , <code>simple_unlock</code> , <code>disable_lock</code> , or <code>unlock_enable</code>), if it could be resolved.
Acquisitions	The number times the function was able to acquire this lock.
Miss Rate	The percentage of acquisition attempts that failed.
Spin Count	The number of unsuccessful attempts by the function to acquire this lock.
Wait Count	The number of times that any thread was forced to wait on the lock, using a call to this function to acquire the lock.
Busy Count	The number of times the function used tried to acquire the lock without success (that is, calls to <code>simple_lock_try()</code> that returned busy).

Percent Held of Total Time contains the following subfields:

CPU	The percentage of the cumulative CPU time that the lock was held by an executing thread that had acquired the lock through a call to this function.
EIapse(d)	The percentage of the elapsed real time that the lock was held by any thread at all, whether running or suspended, that had acquired the lock through a call to this function.
Spin	The percentage of cumulative cpu time that executing threads spent spinning on the lock while trying to acquire the lock through a call to this function.
Wait	The percentage of elapsed real time that executing threads spent waiting on the lock while trying to acquire the lock through a call to this function.
Return Address	The return address to this calling function, in hexadecimal.
Start Address	The start address of the calling function, in hexadecimal.
Offset	The offset from the function start address to the return address, in hexadecimal.

The functions are ordered by the same sorting criterion as the locks, controlled by the `-s` option of `splat`. Further, the number of functions listed is controlled by the `-S` parameter, with the default being the top 10 functions being listed.

Thread detail

Example 39-6 shows an example of the thread detail report. This report is obtained by using the `-dt` or `-da` options of `splat`.

Note that at any point in time, a single thread is either running or it is not, and when it runs, it only runs on one CPU. Some of the composite statistics are measured relative to the cumulative CPU time when they measure activities that can happen simultaneously on more than one CPU, and the magnitude of the measurements can be proportional to the number of CPUs in the system. In contrast, the thread statistics are generally measured relative to the elapsed real time, which is the amount of time a single CPU spends processing and the amount of time a single thread spends in an executing or suspended state.

Example 39-6 Thread detail report

ThreadID	Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Percent CPU	Held of Elapsed	Total Time Spin	Wait
517	1613	0.00	0	0	0	0.05	100.00	0.00	99.81
5423	1569	0.00	0	0	0	0.06	100.00	0.00	0.00
4877	504	0.00	0	0	0	0.01	100.00	0.00	0.00
4183	79	0.00	0	0	0	0.00	100.00	0.00	0.00
3	59	0.00	0	0	0	0.00	100.00	0.00	0.00
2065	36	0.00	0	0	0	0.00	100.00	0.00	0.00
2323	36	0.00	0	0	0	0.00	100.00	0.00	0.00
2839	33	0.00	0	0	0	0.00	100.00	0.00	0.00
2581	33	0.00	0	0	0	0.00	100.00	0.00	0.00
5425	8	0.00	0	0	0	0.00	100.00	0.00	0.00

The columns are defined as follows:

ThreadID	The <i>thread identifier</i> .
Acquisitions	The number of times this thread acquired the lock.
Miss Rate	The percentage of acquisition attempts by the thread that failed to secure the lock.
Spin Count	The number of unsuccessful attempts by this thread to secure the lock.
Wait Count	The number of times this thread was forced to <i>wait</i> until the lock came available.

Busy Count The number of times this thread used *try* to acquire the lock, without success (calls to `simple_lock_try()` that returned busy).

Percent Held of Total Time consists of the following subfields:

CPU The percentage of the *elapsed real time* that this thread executed while holding the lock.

Elapse(d) The percentage of the *elapsed real time* that this thread held the lock while running or suspended.

Spin The percentage of *elapsed real time* that this thread executed while spinning on the lock.

Wait The percentage of *elapsed real time* that this thread spent *waiting* on the lock.

Complex lock report

The AIX Complex lock supports *recursive* locking, where a thread can acquire the lock more than once before releasing it, as well as differentiating between *write-locking*, which is exclusive, from *read-locking*, which is not. The top of the complex lock report appears in Example 39-7.

Example 39-7 Complex lock report (top part)

```
[AIX COMPLEX Lock]                    CLASS:        TOD_LOCK_CLASS.FFFF
ADDRESS: 0000000000856C88            KEX: unix
NAME:            tod_lock
-----
```

Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Secs Held		Percent Held (15.710062s)				
					CPU	Elapsed	Real CPU	Real Elapsed	Comb Spin	Real Wait	
8763	0.000	0	0	0	0.044070	0.044070	0.28	0.28	0.00	0.00	

%Enabled	0.00 (0)	SpinQ	Min	Max	Avg	WaitQ	Min	Max	Avg		
%Disabled	100.00 (8763)	Depth	0	0	0	Depth	0	0	0		

		Readers	0	0	0	Readers	0	0	0		
		Writers	0	0	0	Writers	0	0	0		

Upgrade	0	0	0								
Dngrade	0	0	0	LockQ	Min	Max	Avg				
Recursion	0	1	0	Readers	0	1	0				

Note that this report begins with [AIX COMPLEX Lock]. Most of the entries are identical to the simple lock report, while some of them are differentiated by read/write/upgrade. For example, the SpinQ and WaitQ statistics include the minimum, maximum, and average number of threads spinning or waiting on the lock. They also include the minimum, maximum, and average number of threads attempting to acquire the lock for reading versus writing. Because an arbitrary number of threads can hold the lock for *reading*, the report includes the minimum, maximum, and average number of readers in the LockQ that holds the lock.

A thread may hold a lock for writing; this is *exclusive* and prevents any other thread from securing the lock for reading *or* for writing. The thread *downgrades* the lock by simultaneously releasing it for writing and acquiring it for reading; this enables other threads to acquire the lock for reading, as well. The reverse of this operation is an *upgrade*; if the thread holds the lock for reading and no other thread holds it as well, the thread simultaneously releases the lock for reading and acquires it for writing. The upgrade operation may require that the thread wait until other threads release their read-locks. The downgrade operation does not.

A thread may acquire the lock to some recursive depth; it must release the lock the same number of times to free it. This is useful in library code where a lock must be secured at each entry point to the library; a thread will secure the lock once as it enters the library, and internal calls to the library entry points simply re-secure the lock, and release it when returning from the call. The minimum, maximum, and average recursion depths of any thread holding this lock are reported in the table.

A thread holding a *recursive* write-lock is not allowed to downgrade it because the downgrade is intended to apply to only the last write-acquisition of the lock, and the prior acquisitions had a real reason to keep the acquisition exclusive. Instead, the lock is marked as being in the downgraded state, which is erased when the this latest acquisition is released or upgraded. A thread holding a recursive read-lock can only upgrade the latest acquisition of the lock, in which case the lock is marked as being upgraded. The thread will have to wait until the lock is released by any other threads holding it for reading. The minimum, maximum, and average recursion depths of any thread holding this lock in an upgraded or downgraded state are reported in the table.

The Lock Activity report also breaks down the time by whether the lock is being secured for reading, writing, or upgrading, as shown in Example 39-8.

Example 39-8 Complex lock report (lock activity)

Lock Activity w/Interrupts Enabled (mSecs)					
READ	Count	Minimum	Maximum	Average	Total
+++++++	++++++	+++++	+++++	+++++	+++++
LOCK	7179	0.001260	0.023825	0.005623	40.366684

SPIN	0	0.000000	0.000000	0.000000	0.000000
UNDISP	0	0.000000	0.000000	0.000000	0.000000
WAIT	0	0.000000	0.000000	0.000000	0.000000
PREEMPT	0	0.000000	0.000000	0.000000	0.000000

WRITE	Count	Minimum	Maximum	Average	Total
LOCK	1584	0.001380	0.008582	0.002338	3.703169
SPIN	0	0.000000	0.000000	0.000000	0.000000
UNDISP	0	0.000000	0.000000	0.000000	0.000000
WAIT	0	0.000000	0.000000	0.000000	0.000000
PREEMPT	0	0.000000	0.000000	0.000000	0.000000

UPGRADE	Count	Minimum	Maximum	Average	Total
LOCK	0	0.000000	0.000000	0.000000	0.000000
SPIN	0	0.000000	0.000000	0.000000	0.000000
UNDISP	0	0.000000	0.000000	0.000000	0.000000
WAIT	0	0.000000	0.000000	0.000000	0.000000
PREEMPT	0	0.000000	0.000000	0.000000	0.000000

Note that there is no time reported to perform a downgrade because this is performed without any contention. The upgrade state is only reported for the case where a recursive read-lock is upgraded; otherwise the thread activity is measured as releasing a read-lock and acquiring a write-lock.

The function- and thread- details also break down the acquisition, spin, and wait counts by whether the lock is to be acquired for reading or writing, as shown in Example 39-9.

Example 39-9 Complex lock report (function and thread detail)

Function Name	Acquisitions Write	Acquisitions Read	Miss Rate	Spin Count Write	Spin Count Read	Wait Count Write	Wait Count Read	Busy Count	Percent CPU	Percent Held Elapse	Total Time Spin	Total Time Wait
.tstart	0	1912	0.00	0	0	0	0	0	0.07	0.07	0.00	0.00
.clock	0	1911	0.00	0	0	0	0	0	0.05	0.05	0.00	0.00

Return Address	Start Address	Offset
000000000001AA54	0000000000000000	0001AA54
000000000001A494	0000000000000000	0001A494

ThreadID	Acquisitions		Miss Rate	Spin Count		Wait Count		Busy Count	Percent CPU	Held of Total Time		
	Write	Read		Write	Read	Write	Read			Elapse	Spin	Wait
5423	1206	5484	0.00	0	0	0	0	0	0.24	0.24	0.00	0.00
4877	300	1369	0.00	0	0	0	0	0	0.03	0.03	0.00	0.00
517	54	242	0.00	0	0	0	0	0	0.01	0.01	0.00	0.00
4183	5	27	0.00	0	0	0	0	0	0.00	0.00	0.00	0.00

39.2.5 PThread synchronizer reports

By default, **splat** prints out a detailed report for each PThread entry in the summary report. The PThread synchronizers come in three types; mutex, read/write lock, and condition-variable. The mutex and read/write lock are related to the AIX complex lock, so you will see similarities in the lock detail reports. The condition-variable differs significantly from a lock, and this is reflected in the report details.

The PThread library instrumentation does not provide names or classes of synchronizers, so the addresses are the only way we have to identify them. Under certain conditions the instrumentation is able to capture the return addresses of the function-call stack, and these addresses are used with the **gennames** output to identify the call-chains when these synchronizers are created. Sometimes the creation and deletion times of the synchronizer can be determined as well, along with the ID of the PThread that created them. Example 39-10 shows an example of the header.

Example 39-10 PThread synchronizer report header

```
[PThread MUTEX]   ADDRESS: 00000000F0049DE8
Parent Thread: 0000000000000001   creation time: 0.624240
Creation call-chain
=====
00000000D00D9414   .pthread_mutex_lock
00000000D00E0D48   .pthread_once
00000000D01EC30C   .__getgrent_tsd_callback
00000000D01D9574   .__libc_inline_callbacks
00000000D01D9500   .__libc_declare_data_functions
00000000D00EF400   .__pth_init_libc
00000000D00DCF78   .pthread_init
0000000010000318   .driver_addmulti
0000000010000234   .driver_addmulti
00000000D01D8E0C   .__modinit
0000000010000174   .driver_addmulti
```

Mutex reports

The PThread mutex is like an AIX simple lock in that only one thread can acquire the lock and is like an AIX complex lock in that it can be held recursively. A sample report is shown in Example 39-11.

Example 39-11 PThread mutex report

```
[PThread MUTEX] ADDRESS: 00000000F010A3C8
Parent Thread: 0000000000000001 creation time: 15.708728
Creation call-chain =====
00000000D00491BC .pthread_mutex_lock
00000000D0050DA0 .pthread_once
00000000D007417C ._odm_init
00000000D01D9600 ._libc_process_callbacks
00000000D01D8F28 ._modinit
000000001000014C .driver_addmulti
=====
```

Acqui- sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Secs Held		Percent Held (15.710062s)			
					CPU	Elapsed	Real CPU	Real Elapsed	Comb Spin	Real Wait
1	0.000	0	0	0	0.000000	0.000000	0.00	0.00	0.00	0.00

```
-----
Depth   Min  Max  Avg
SpinQ   0   0   0
WaitQ   0   0   0
Recursion 0   1   0
-----
```

Besides the common header information and the [PThread MUTEX] identifier, this report lists the following lock details:

- Acquisitions** The number of times the lock was acquired in the analysis interval.
- Miss Rate** The percentage of attempts that failed to acquire the lock.
- Spin Count** The number of unsuccessful attempts to acquire the lock.
- Wait Count** The number of times a thread was forced into a suspended *wait* state waiting for the lock to come available.
- Busy Count** The number of trylock() calls that returned *busy*.
- Seconds Held** This field contains the following subfields:
 - CPU** The total number of CPU seconds that the lock was held by an executing thread.

	Elapsed	The total number of elapsed seconds that the lock was held, whether the thread was running or suspended.
Percent Held	This field contains the following subfields:	
	Real CPU	The percentage of the <i>cumulative CPU time</i> that the lock was held by an executing thread.
	Real Elapsed	The percentage of the <i>elapsed real time</i> that the lock was held by any thread at all, either running or suspended.
	Comb(ined) Spin	The percentage of the <i>cumulative cpu time</i> that running threads spent spinning while trying to acquire this lock.
	Real Wait	The percentage of <i>elapsed real time</i> that any thread was waiting to acquire this lock. Note that if two or more threads are waiting simultaneously, this wait-time will only be charged <i>once</i> . If you want to know <i>how many</i> threads were waiting simultaneously, look at the WaitQ Depth statistics.
Depth	This field contains the following subfields:	
	SpinQ	The minimum, maximum, and average number of threads <i>spinning</i> on the lock, whether executing or suspended, across the analysis interval.
	WaitQ	The minimum, maximum, and average number of threads <i>waiting</i> on the lock, across the analysis interval.
	Recursion	The minimum, maximum, and average recursion depth to which each thread held the lock.

If the `-dt` or `-da` options are used, `splat` reports the thread detail as shown in Example 39-12.

Example 39-12 PThread mutex report (thread detail)

Acqui- PThreadID	Miss sitions	Spin Rate	Wait Count	Busy Count	Percent Held of Total Time Count	CPU	Elapse	Spin	Wait
1	1	0.0000	0	0	0	0.0001	0.0001	0.0000	0.0000

The columns are defined as follows:

PThreadID	The <i>PThread</i> identifier.
Acquisitions	The number of times this thread acquired the lock.
Miss Rate	The percentage of acquisition attempts by the thread that failed to secure the lock.
Spin Count	The number of unsuccessful attempts by this thread to secure the lock.
Wait Count	The number of times this thread was forced to <i>wait</i> until the lock came available.
Busy Count	The number of times this thread used <i>try</i> to acquire the lock without success (calls to <code>simple_lock_try()</code> that returned busy).

Percent Held of Total Time contains the following subfields:

CPU	The percentage of the <i>elapsed real time</i> that this thread executed while holding the lock.
Elapse(d)	The percentage of the <i>elapsed real time</i> that this thread held the lock while running or suspended.
Spin	The percentage of <i>elapsed real time</i> that this thread executed while spinning on the lock.
Wait	The percentage of <i>elapsed real time</i> that this thread spent <i>waiting</i> on the lock.

Read/Write lock reports

The PThread read/write lock is like an AIX complex lock in that it can be acquired for reading or writing; writing is exclusive in that a single thread can only acquire the lock for writing, and no other thread can hold the lock for reading or writing at that point. Reading is not exclusive, so more than one thread can hold the lock for reading. Reading is recursive in that a single thread can hold multiple read-acquisitions on the lock. Writing is not. A sample report is shown in Table 39-2.

Table 39-2 PThread read/write lock report

```
[PThread RWLock]   ADDRESS:   000000002FF22B70
Parent Thread: 0000000000000001   creation time: 0.051140
Creation call-chain =====
00000000100003D4   .driver_addmulti
```

00000000100001B4 .driver_addmulti

```
=====
Acqui- | Miss Spin Wait | Secs Held | Percent Held (383.290027s )
sitions | Rate Count Count | CPU Elapsed | Real Real Comb Real
3688386 | 0.000 0 0 | 383.2384 383.2384 | 99.99 99.99 0.00 0.00
=====
```

```
-----
                Readers                Writers                Total
Depth  Min  Max  Avg                Min  Max  Avg                Min  Max  Avg
LockQ  0   3688386 3216413                0   0   0                0   3688386 3216413
SpinQ  0   0   0                0   0   0                0   0   0
WaitQ  0   0   0  0  0                0   0   0                0   0   0
-----
```

Besides the common header information and the [PThread RWLock] identifier, this report lists the following lock details:

Acquisitions The number of times the lock was acquired in the analysis interval.

Miss Rate The percentage of attempts that failed to acquire the lock.

Spin Count The number of unsuccessful attempts to acquire the lock.

Wait Count The current PThread implementation does not force threads to wait on read/write locks. What is reported here is the number of times a thread, spinning on this lock, is undispatched.

Seconds Held This field contains the following subfields:

CPU The total number of CPU seconds that the lock was held by an executing thread. If the lock is held multiple times by the same thread, only one hold interval is counted.

Elapsed The total number of elapsed seconds that the lock was held by any thread, whether the thread was running or suspended.

Percent Held This field contains the following subfields:

Real CPU The percentage of the *cumulative CPU time* that the lock was held by any executing thread.

Real Elapsed The percentage of the *elapsed real time* that the lock was held by any thread at all, either running or suspended.

Comb(ined) Spin The percentage of the *cumulative cpu time* that running threads spent spinning while trying to acquire this lock.

Real Wait The percentage of *elapsed real time* that any thread was waiting to acquire this lock. Note that if two or more threads are waiting simultaneously, this wait-time will only be charged *once*. If you want to know *how many* threads were waiting simultaneously, look at the WaitQ Depth statistics.

Depth

This field contains the following subfields:

LockQ The minimum, maximum, and average number of threads *holding* the lock, whether executing or suspended, across the analysis interval. This is broken down by read-acquisitions, write-acquisitions, and all acquisitions together.

SpinQ The minimum, maximum, and average number of threads *spinning* on the lock, whether executing or suspended, across the analysis interval. This is broken down by read-acquisitions, write-acquisitions, and all acquisitions together.

WaitQ The minimum, maximum, and average number of threads in a timed-wait state for the lock, across the analysis interval. This is broken down by read-acquisitions, write-acquisitions, and all acquisitions together.

If the -dt or -da options are used, **sp1at** reports the thread detail as shown in Example 39-13.

Example 39-13 PThread read/write lock (thread detail)

ThreadID	Acquisitions		Miss Rate	Spin Count		Wait Count		Busy Count	Percent CPU Elapse	Held of Total Time	
	Write	Read		Write	Read	Write	Read			Spin	Wait
1 0	3688	3860	0.000	0	0	0	00.00	99.99	0.00	0.00	

The columns are defined as follows:

PThreadID	The <i>PThread</i> identifier.
Acquisitions	The number of times this thread acquired the lock, differentiated by write versus read.
Miss Rate	The percentage of acquisition attempts by the thread that failed to secure the lock.
Spin Count	The number of unsuccessful attempts by this thread to secure the lock, differentiated by write versus read.
Wait Count	The number of times this thread was forced to <i>wait</i> until the lock came available, differentiated by write versus read.
Busy Count	The number of times this thread used <i>try</i> to acquire the lock, without success (for example calls to <code>simple_lock_try()</code> that returned busy).

Percent Held of Total Time contains the following subfields:

CPU	The percentage of the <i>elapsed real time</i> that this thread executed while holding the lock.
Elapse(d)	The percentage of the <i>elapsed real time</i> that this thread held the lock while running or suspended.
Spin	The percentage of <i>elapsed real time</i> that this thread executed while spinning on the lock.
Wait	The percentage of <i>elapsed real time</i> that this thread spent <i>waiting</i> on the lock.

Condition-Variable report

The PThread condition-variable is a synchronizer but not a lock. A PThread is suspended until a signal indicates that the condition now holds. A sample report is shown in Example 39-14.

Example 39-14 PThread Condition-Variable Report

```

[PThread CondVar]  ADDRESS:  0000000020004858
Parent Thread:    0000000000000000    creation time: 18.316493
Creation call-chain =====
00000000D004E42C  ._free_pthread
00000000D004CE98  .pthread_init
00000000D01D8E40  .__modinit
000000001000014C  .driver_addmulti
=====

```

	Fail	Spin	Wait	Spin / Wait Time (18.330873s)	Comb	Comb
Passes	Rate	Count	Count	Spin	Wait	

```
0          | 0.00 0    0    | 0.00    0.00
```

```
Depth      Min   Max   Avg
SpinQ      0     0     0
WaitQ      0     0     0
```

Besides the common header information and the [PThread CondVar] identifier, this report lists the following details:

Passes The number of times the condition was signaled to hold during the analysis interval.

Fail Rate The percentage of times that the condition was tested and was not found to be true.

Spin Count The number of times that the condition was tested and was not found to be true.

Wait Count The number of times a thread was forced into a suspended *wait* state waiting for the condition to be signaled.

Spin / Wait Time This field contains the following subfields:

Comb Spin The total number of CPU seconds that threads spun while waiting for the condition.

Comb Wait The total number of elapsed seconds that threads spent in a wait state for the condition.

Depth This field contains the following subfields:

SpinQ The minimum, maximum, and average number of threads *spinning* while waiting for the condition, across the analysis interval.

WaitQ The minimum, maximum, and average number of threads *waiting* for the condition, across the analysis interval.

If the **-dt** or **-da** options are used, **sp1at** reports the thread detail as shown in Example 39-15.

Example 39-15 PThread Condition-Variable Report (thread detail)

PThreadID	Passes	Fail Rate	Spin Count	Wait Count	% Total Time	
					Spin	Wait
1	80312	0.0000	0	80312	0.0000	82.4531
258	80311	0.0000	0	80312	0.0000	82.4409

The columns are defined as follows:

PThreadID	The PThread identifier.
Passes	The number of times this thread was notified that the condition <i>passed</i> .
Fail Rate	The percentage of times the thread checked the condition and did not find it to be true.
Spin Count	The number of times the thread checked the condition and did not find it to be true.
Wait Count	The number of times this thread was forced to <i>wait</i> until the condition came true.
Percent Total Time	This field contains the following subfields: <ul style="list-style-type: none">Spin The percentage of <i>elapsed real time</i> that this thread spun while testing the condition.Wait The percentage of <i>elapsed real time</i> that this thread spent waiting for the condition to hold.

Archived

The trace, trcnm, and trcrpt commands

The **trace** command is a utility that monitors statistics of user and kernel subsystems in detail.

Many of the performance tools listed in this book, such as **curt** (see Chapter 35, “The curt command” on page 677), use **trace** to obtain their data, then format the data read from the raw trace report and present it to the user. The **trcrpt** command formats a report from the trace log.

Usually before analyzing the trace file, you would use other performance tools to obtain an overview of the system for potential or real performance problems. This give an indication of what to look for in the trace for resolving any performance bottlenecks. The commonly used methodology is to look at the **curt** output, then other performance command outputs, then the formatted trace file.

The **trcnm** command generates a list of all symbols with their addresses defined in the kernel. This data is used by the **trcrpt -n** command to interpret addresses when formatting a report from a trace log file.

The **trace** command resides in `/usr/sbin` and is linked from `/usr/bin`. The **trcnm** and **trcrpt** commands reside in `/usr/bin`. All of these commands are part of the `bos.sysmgt.trace` fileset, which is installable from the AIX base installation media.

40.1 trace

The following syntax applies to the **trace** command:

```
trace [ -a [ -g ] ] [ -f | -l ] [ -b | -B ] [ -c ] [ -C [ CPUList | all ] ] [ -d ]  
      [ -h ] [ -j Event [ ,Event ] ] [ -k Event [ ,Event ] ]  
      [ -J Event-group [ ,Event-group ] ] [ -K Event-group [ ,Event-group ] ]  
      [ -m Message ] [ -n ] [ -o Name ] [ -o- ] [ -p ] [ -s ] [ -L Size ]  
      [ -T Size ]
```

Flags

- a** Runs the trace daemon asynchronously (that is, as a background task). Once **trace** has been started this way, you can use the **trcon**, **trcoff**, and **trcstop** commands to respectively start tracing, stop tracing, or exit the trace session. These commands are implemented as links to **trace**.
- b** Allocates buffers from the kernel heap. If the requested buffer space cannot be obtained from the kernel heap, the command fails. This flag is only valid for a 32-bit kernel.
- B** Allocates buffers in separate segments. This flag is only valid for a 32-bit kernel.
- c** Saves the trace log file, adding **.old** to its name.
- C[CPUList | all]** Traces using one set of buffers per CPU in the CPUList. The CPUs can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks. To trace all CPUs, specify **all**. Because this flag uses one set of buffers per CPU, and produces one file per CPU, it can consume large amounts of memory and file space and should be used with care. The files produced are named **trcfile**, **trcfile-0**, **trcfile-1**, and so forth, where the numbers represent the CPU numbers. If **-T** or **-L** are specified, the sizes apply to each set of buffers and each file. On a uniprocessor system, you may specify **-C all**, but the **-C** flag with a list of CPU numbers is ignored. If the **-C** flag is used to specify more than one CPU, such as **-C all** or **-C "0 1"**, then the associated buffers are not put into the system dump.
- d** Disables the automatic start of trace data collection. Normally the collection of trace data starts automatically when you issue the trace daemon, but when you have specified the **trace** command using the **-d** flag, the trace will not start until the **trcon** command has been issued.

- f** Runs **trace** in a single mode. Causes the collection of trace data to stop as soon as the in-memory buffer is filled up. The trace data is then written to the trace log. Use the **trcon** command to restart trace data collection and capture another full buffer of data. If you issue the **trcoff** command before the buffer is full, trace data collection is stopped and the current contents of the buffer are written to the trace log.
- g** Starts a trace session on a generic trace channel (channels 1 through 7). This flag works only when **trace** is run asynchronously (-a). The return code of the command is the channel number; the channel number must subsequently be used in the generic trace subroutine calls. To stop the generic trace session, use the command **trcstop -<channel_number>**.
- h** Omits the header record from the trace log. Normally, the trace daemon writes a header record with the date and time (from the **date** command) at the beginning of the trace log; the system name, version and release, the node identification, and the machine identification (from the **uname -a** command); and a user-defined message. At the beginning of the trace log, the information from the header record is included in the output of the **trcrpt** command.

-j Event[,Event] See the description for the -k flag.

-k Event[,Event]

Specifies the user-defined events for which you want to collect (-j) or exclude (-k) trace data. The **Event** list items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.

The following events are used to determine the PID, the cpuid, and the exec path name in the **trcrpt** report:

106 DISPATCH
10C DISPATCH IDLE PROCESS
134 EXEC SYSTEM CALL
139 FORK SYSTEM CALL
465 KTHREAD CREATE

If any of these events is missing, the information reported by the **trcrpt** command will be incomplete. Consequently, when using the -j flag, you should include all of these events in the Event list. Conversely, when using the -k flag, you should not include these events in the Event list. If starting the trace with **smi t** or the -J flag, these events are in the tidhk group. Additional event hooks can be read in Appendix B, “Trace hooks” on page 973.

-J Event-group [, Event-group]

-K Event-group [, Event-group]

Specifies the event groups to be included (-J) or excluded (-K). The -J and -K flags work like -j and -k, except with event groups instead of individual hook IDs. All four flags, -j, -J, -k, and -K, may be specified. Some important event groups relate to trace hooks used by other commands, such as **curt** and **splat**. A list of these groups can be shown by the command **trcevrp -1**.

-l Runs **trace** in a circular mode. The **trace** daemon writes the trace data to the trace log when the collection of trace data is stopped. Only the last buffer of trace data is captured. When you stop trace data collection using the **trcoff** command, restart it using the **trcon** command.

-L Size Overrides the default trace log file size of 1 MB with the value stated. Specifying a file size of zero sets the trace log file size to the default size. For a multiple-CPU system, the size limit applies to each of the per-CPU logfiles that are generated, rather than their collective size.

Note: In the circular and alternate modes, the trace log file size must be at least twice the size of the trace buffer. In the single mode, the trace log file must be at least the size of the buffer. See the -T flag for information about controlling the trace buffer size.

-m Message Specifies text to be included in the message field of the trace log header record.

-n Adds information to the trace log header; lock information, hardware information, and, for each loader entry, the symbol name, address, and type.

-o Name Overrides the `/var/adm/ras/trcfile` default trace log file and writes trace data to a user-defined file.

-o - Overrides the default trace log name and writes trace data to standard output. The -c flag is ignored when using this flag. An error is produced if -o- and -C are specified.

-p Includes the cpuid of the current processor with each hook. This flag is only valid for 64-bit kernel traces. The **trcrpt** command can report the cpuid whether or not this option is specified.

-s Stops tracing when the trace log fills. The **trace** daemon normally wraps the trace log when it fills up and continues to collect trace data. During asynchronous operation, this flag causes the **trace** daemon to stop trace data collection. During

interactive operations, the **quit** subcommand must be used to stop **trace**.

-T Size

Overrides the default trace buffer size of 128 KB with the value stated. You must be root to request more than 1 MB of buffer space. The maximum possible size is 268,435,184 bytes (256 MB) unless the **-f** flag is used, in which case it is 536,870,368 bytes (512 MB). The smallest possible size is 8192 bytes, unless the **-f** flag is used, in which case it is 16,392 bytes. Sizes between 8,192 and 16,392 will be accepted when using the **-f** flag, but the actual size used will be 16,392 bytes. Note that with the **-C** option allocating one buffer per traced CPU, the size applies to each buffer rather than the collective size of all buffers.

Note: In the single mode, the trace log file must be at least the size of the buffer. See the **-L** flag for information about controlling the trace log file size. The trace buffers use pinned memory, which means they are not pageable. Therefore, the larger the trace buffers, the less physical memory is available to applications. In the circular and the alternate modes, the trace buffer size must be one-half or less the size of the trace log file.

Unless the **-b** or **-B** flags are specified, the system attempts to allocate the buffer space from the kernel heap. If this request cannot be satisfied, the system then attempts to allocate the buffers as separate segments.

The **-f** flag actually uses two buffers, which behave as a single buffer (except that a buffer wraparound trace hook will be recorded when the first buffer is filled).

Subcommands

When run interactively, **trace** recognizes the following subcommands:

trcon	Starts the collection of trace data.
trcoff	Stops the collection of trace data.
q or quit	Stops the collection of trace data and exits trace .
!	Runs the shell command specified by the Command parameter.
?	Displays the summary of trace subcommands.

Signals

The **INTERRUPT** signal acts as a toggle to start and stop the collection of trace data. Interruptions are set to **SIG_IGN** for the traced process.

Files

/usr/include/sys/trcmacos.h	Defines trchook and utrchook macros.
/var/adm/ras/trcfile	Contains the default trace log file.

40.1.1 Information about measurement and sampling

When **trace** is running, it will require a CPU overhead of less than 2%. When the trace buffers are full, **trace** will write its output to the trace log, which may require up to five percent of CPU resource. The **trace** command claims and pins buffer space. If a system is short of memory, then running **trace** could further degrade system performance.

Attention: Depending on what trace hooks you are tracing, the trace file can become very large.

The **trace** daemon configures a trace session and starts the collection of system events. The data collected by the trace function is recorded in the trace log. A report from the trace log is a raw file and can be formatted to a readable ASCII file with the **trcrpt** command.

When invoked with the **-a** flag, the trace daemon runs asynchronously (that is, as a background task). Otherwise, it is run interactively and prompts you for subcommands as is shown in Example 40-3 on page 768.

You can use the System Management Interface Tool (**smit**) to run the **trace** daemon. See “Using SMIT to stop and start trace” on page 767 for details.

Operation modes

There are three modes of trace data collection:

- ▶ Alternate (the default)
All trace events are captured in the trace log file.
- ▶ Circular
The trace events wrap within the in-memory buffers and are not captured in the trace log file until the trace data collection is stopped. To choose the Circular trace method, use the **-l** flag.
- ▶ Single
The collection of trace events stops when the in-memory trace buffer fills up and the contents of the buffer are captured in the trace log file. To choose the Single trace method, use the **-f** flag.

Buffer allocation

Trace buffers are either allocated from the kernel heap or put into separate segments. By default, buffers are allocated from the kernel heap unless the buffer size requested is too large for buffers to fit in the kernel heap, in which case they are allocated in separate segments.

Allocating buffers from separate segments hinders trace performance somewhat. However, buffers in separate segments will not take up paging space; just pinned memory. The type of buffer allocation can be specified with the optional `-b` or `-B` flags when using a 32-bit kernel.

40.1.2 Terminology used for trace

In order to understand how the trace facility (also called *trace program*) works, it is important to know the meaning of some terms.

Trace hooks

A *trace hook* is a specific event that is to be monitored. For example, if you wish to monitor Physical File System (PFS) events, include trace hook 10A in the trace. Trace hooks are defined by the kernel and can change with different releases of the operating system, but trace hooks can also be defined and used by an application. If a specific event in an application does not have a trace hook defined, then this event will never show up in a trace report.

Trace hooks can be displayed with `trcrpt -j`. The example in “AIX 5L trace hooks” on page 974 shows trace hooks that are applicable for AIX 5L Version 5.2.

It is recommended that you run `trcrpt -j` to check for any modifications to the trace hooks that IBM may make.

Hook ID

A unique number is assigned to a trace hook (for example, a certain event) called a *hook ID*. These hook IDs can either be called by a user application or by the kernel. The hook IDs can be found in the file `/usr/sys/include/trchkid.h`.

Trace daemon

The trace daemon (sometimes also called trace command or trace process) has to be activated in order to generate statistics about user processes and kernel subsystems. This is actually the process that can be monitored by the `ps` command.

Trace buffer

The data that is collected by the `trace` daemon is first written to the trace buffer. Only one trace buffer is transparent to the user, though it is internally divided into two parts, also referred to as a set of trace buffers. By using the `-C` option with the `trace` command, one set of trace buffers can be created for each CPU of an SMP system. This enhances the total trace buffer capacity.

Trace log file

Once one of the two internal trace buffers is full, its content is usually written to the *trace log file*. The trace log file does fill up quite quickly, so that in most cases only a few seconds are chosen to be monitored by **trace**.

The sequence followed by the trace facility is shown in Figure 40-1.

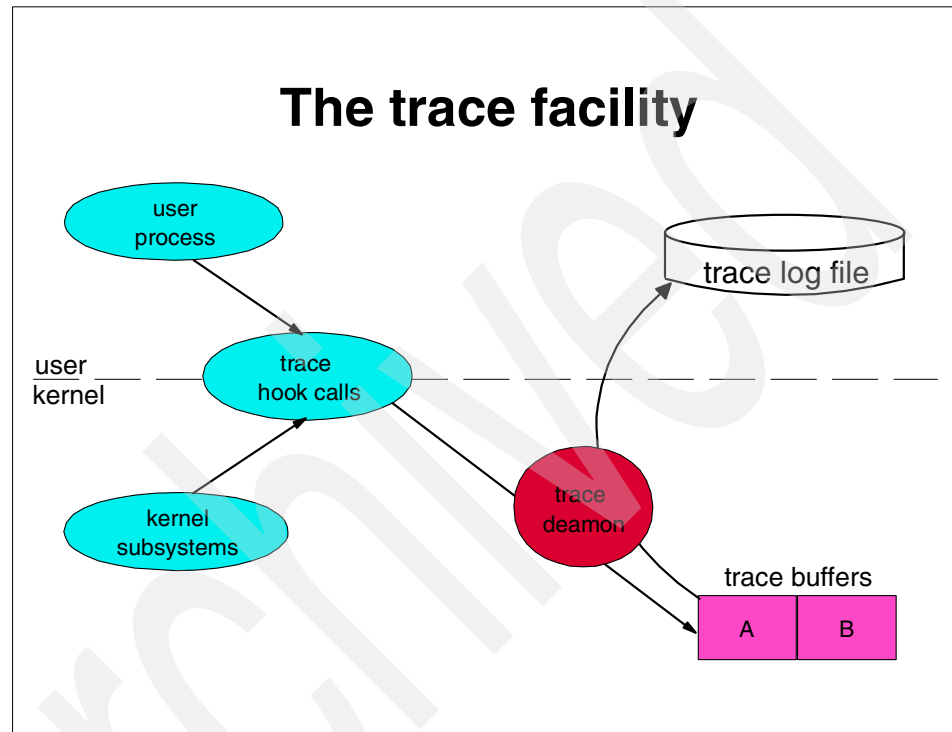


Figure 40-1 The trace facility

Either a user process or a kernel subsystem calls a *trace hook function* (by using the hook ID). These trace hook functions check whether the trace daemon is running and, if so, pass the data to the trace daemon that then takes the hook ID and the according event and writes them (together with a time stamp) sequentially to the *trace buffer*. Depending on the options that were chosen when the trace daemon was invoked (see “Operation modes” on page 764), the trace data is then written to the *trace log file*. A report from the trace log can be generated with the **trcrpt** command.

Just as important is to keep in mind that the trace log file can grow huge depending on the amount of data that is being collected. A trace on a fully loaded 24-way SMP can easily accumulate close to 100 MB of trace data in less than a

minute. Some sensibility is required to determine whether all that data is really needed. Often a few seconds is enough to catch all the important activities that need to be traced. An easy method of limiting the size of the trace log file is to run the trace in Single mode as discussed in "Operation modes" on page 764.

40.1.3 Ways to start and stop trace

There are several ways to stop and start trace.

Using SMIT to stop and start trace

A convenient way to stop and start trace is to use the `smitty trace` command. This is especially convenient if you are including or excluding specific trace hooks. Using the System Management Interface Tool (SMIT) enables you to view a trace hook list using the F4 key and choose the trace hook(s) to include or exclude.

To access the trace menus of SMIT, type `smitty trace`. The menu in Example 40-1 will appear.

Example 40-1 The SMIT trace menu

```
Trace

Move cursor to desired item and press Enter.

START Trace
STOP Trace
Generate a Trace Report
Manage Event Groups
```

Enter the START Trace menu and start the trace as shown in Example 40-2.

Example 40-2 Using SMIT to start the trace

```
# smitty trace

START Trace

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

EVENT GROUPS to trace       [Entry Fields]      +
ADDITIONAL event IDs to trace                 +
Event Groups to EXCLUDE from trace                 +
Event IDs to EXCLUDE from trace                 +
Trace MODE                  [alternate]          +
STOP when log file full?    [no]                 +
```

LOG FILE	[trace.raw]	
SAVE PREVIOUS log file?	[no]	+
Omit PS/NM/LOCK HEADER to log file?	[yes]	+
Omit DATE-SYSTEM HEADER to log file?	[no]	+
Run in INTERACTIVE mode?	[no]	+
Trace BUFFER SIZE in bytes	[10000000]	#
LOG FILE SIZE in bytes	[10000000]	#
Buffer Allocation	[automatic]	+

You can exit the menu, then select the `STOP Trace` option of the menu in Example 40-1 on page 767 to stop the trace. The trace `trace.raw` will reside in the current directory.

Running trace interactively

Example 40-3 shows how to run **trace** interactively, tracing the `ls` command as well as other processes running on the system from within the **trace** command. The raw trace file created by **trace** is called `/var/adm/ras/trcfile`.

Example 40-3 Running trace interactively

```
# trace
-> !ls
-> quit
# ls -l /var/adm/ras/trcfile*
-rw-rw-rw-  1 root      system      1338636 Apr 16 08:53 /var/adm/ras/trcfile
```

Running trace asynchronously

Example 40-4 shows how to run **trace** asynchronously, tracing the `ls` command as well as other processes running on the system. This method avoids delays when the command finishes. The raw trace file created by **trace** is called `/var/adm/ras/trcfile`.

Example 40-4 Running trace asynchronously

```
# trace -a ; ls ; trcstop
# ls -l /var/adm/ras/trcfile*
-rw-rw-rw-  1 root      system      208640 Apr 16 08:54 /var/adm/ras/trcfile
```

Note that by using this method, the trace file is considerably smaller than the interactive method shown in Example 40-3.

Running trace on an entire system for 10 seconds

Example 40-5 on page 769 shows how to run **trace** on the entire system for 10 seconds. This traces all system activity and includes all trace hooks. The raw trace file created by **trace** is called `/var/adm/ras/trcfile`.

Example 40-5 Running trace on an entire system for 10 seconds

```
# trace -a ; sleep 10 ; trcstop
# ls -l /var/adm/ras/trcfile*
-rw-rw-rw- 1 root system 1350792 Apr 16 08:56 /var/adm/ras/trcfile
```

Tracing to a specific log file

Example 40-6 shows how to run **trace** asynchronously, tracing the **ls** command and outputting the raw trace file to `/tmp/my_trace_log`.

Example 40-6 Tracing to a specific log file

```
# ls -l /tmp/my_trace_log
/tmp/my_trace_log not found
# trace -a -o /tmp/my_trace_log; ls; trcstop
# ls -l /tmp/my_trace_log*
-rw-rw-rw- 1 root system 206924 Apr 16 08:58 /tmp/my_trace_log
```

Tracing a command

The following section shows how to trace commands.

Tracing a command that is not already running on the system

Example 40-6 shows how to run **trace** on a command that you are about to start. It allows you to start **trace**, run the command, and then terminate **trace**. This ensures that all trace events are captured.

Tracing a command that is already running on the system

To trace a command that is already running, run a trace on the entire system as in Example 40-5, and use the **trcrpt** command with the `-p` flag to specify reporting of the specific process.

Tracing using one set of buffers per CPU

Normally, **trace** groups all CPU buffers into one trace file. Events that occurred on the individual CPUs may be separated into CPU-specific files as shown in Example 40-7. This increases the total buffered size capacity for collecting trace events.

Example 40-7 Tracing using one set of buffers per CPU

```
# trace -aC all ; sleep 10 ; trcstop
# ls -l /var/adm/ras/trcfile*
-rw-rw-rw- 1 root system 37996 Apr 16 08:59 /var/adm/ras/trcfile
-rw-rw-rw- 1 root system 1313400 Apr 16 09:00 /var/adm/ras/trcfile-0
-rw-rw-rw- 1 root system 94652 Apr 16 09:00 /var/adm/ras/trcfile-1
-rw-rw-rw- 1 root system 184 Apr 16 08:59 /var/adm/ras/trcfile-10
-rw-rw-rw- 1 root system 184 Apr 16 08:59 /var/adm/ras/trcfile-11
-rw-rw-rw- 1 root system 184 Apr 16 08:59 /var/adm/ras/trcfile-12
```

-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-13
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-14
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-15
-rw-rw-rw-	1	root	system	1313400	Apr 16 09:00	/var/adm/ras/trcfile-2
-rw-rw-rw-	1	root	system	1010096	Apr 16 09:00	/var/adm/ras/trcfile-3
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-4
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-5
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-6
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-7
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-8
-rw-rw-rw-	1	root	system	184	Apr 16 08:59	/var/adm/ras/trcfile-9

The example above has four individual files (one for each CPU) plus the master file `/var/adm/ras/trcfile`.

Running the `trace -aCa11 -o mylog` command would produce the files `mylog`, `mylog-0`, `mylog-1`, `mylog-2`, `mylog-3`, and so forth, one for each CPU.

40.1.4 Examples for trace

These are just two examples where `trace` can be used. The `trace` command is a powerful tool that can be used for many diagnostic purposes.

- ▶ Checking return times from called routines

If the system is running slow, then `trace` can be used to determine how long threads are taking to return from functions. Long return times could highlight a performance problem. An example of this shown in “Checking return times from trace” on page 770.

- ▶ Sequential reads and writes

If you are experiencing high disk I/O then you can determine how long the disk I/O is taking to perform and what sort of disk accesses are occurring. For example, a database may be performing a full table scan on an unindexed file to retrieve records. This would be inefficient and may point to problems with indexing, or there may not be an index at all. An example of this is shown in “Sequential reads and writes” on page 774.

Checking return times from trace

In this section we will check return times from the trace to see if there are any long delays.

First, we create a raw trace of all the processes running on the system as in Example 40-8 on page 771. Then the individual CPU traces are combined into the raw trace file (`trace.r`). We will then use `trcrpt` to create the file `trcrpt.out`.

Example 40-8 Running trace on an entire system for 10 seconds

```
# trace -aC all ; sleep 10 ; trcstop
# gennames > gennames.out
# trcnm > trace.nm
# cp /etc/trcfmt trace.fmt
# trcrpt -C all -r /var/adm/ras/trcfile > trace.r
# trcrpt -O exec=on,pid=on,cpuid=on -n trace.nm -t trace.fmt trace.r >
trcrpt.out
```

A useful part of the trace report (trcrp.out) is the return times from various functions that occurred during the trace. Use the **grep** command for only the usec times for an indication of which processes are using the most time. This can also be achieved by using the shell script in Example 40-9. The script greps for the usec times, and displays trace file lines of the top 20 highest return times. It excludes the trace hook ID 102 (wait).

Example 40-9 Script to check for return times in trace

```
# Extract the return times from the trace file

TMPFILE1=/tmp/usec1-$$
TMPFILE2=/tmp/usec2-$$

grep "ID PROCESS NAME" trcrpt.out

grep usec trcrpt.out | grep -vw '102 wait' | awk -F'[ ' '{ print $2 }' | \
awk '{ print $1 }' > $TMPFILE1

sort -rn $TMPFILE1| head -20 > $TMPFILE2

while read string
do
    grep "$string usec" trcrpt.out
done < $TMPFILE2
```

Example 40-10 shows the output from the script.

Example 40-10 Top 20 highest return times

ID	PROCESS NAME	CPU	PID	I	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
104	syncd	2	378		4.504329796	0.000216		return from sync		[472167 usec]
221	wait	0	516	1	4.882048580	0.002962		SCDISKDD iodone:	ipldevice	
	bp=30B47200 B_WRITE				[392401 usec]					
221	wait	0	516	1	4.875472073	0.003951		SCDISKDD iodone:	ipldevice	
	bp=309D2100 B_WRITE				[386128 usec]					
106	java	0	29944		1.924588685	0.000746		dispatch:	cmd=java pid=29944	
	tid=40263 priority=181				old_tid=517 old_priority=255			CPUID=0	[250117 usec]	
104	java	2	29944		9.930639660	0.001493		return from _select		[250117 usec]

```

106 java          0 29944 1.924588685 0.000746          dispatch: cmd=java pid=29944
tid=40263 priority=181 old_tid=517 old_priority=255 CPUID=0 [250117 usec]
104 java          2 29944 9.930639660 0.001493          return from _select [250117 usec]
104 java          0 29944 4.926771906 0.005855          return from _select [250108 usec]
104 java          0 29944 7.928691841 0.029999          return from _select [250100 usec]
104 java          0 29944 8.929828448 0.019108          return from _select [250097 usec]
104 java          0 29944 4.426232284 0.005662          return from _select [250096 usec]
104 java          0 29944 8.429250350 0.009999          return from _select [250089 usec]
104 java          0 29944 7.678503300 0.016433          return from _select [250088 usec]
104 java          0 29944 4.175869414 0.041926          return from _select [250081 usec]
104 java          0 29944 4.676462779 0.032481          return from _select [250080 usec]
104 java          0 29944 8.679499786 0.036143          return from _select [250080 usec]
104 java          0 29944 4.676462779 0.032481          return from _select [250080 usec]
104 java          0 29944 8.679499786 0.036143          return from _select [250080 usec]
104 java          0 29944 8.179039200 0.021662          return from _select [250075 usec]
104 java          0 29944 2.424882026 0.012939          return from _select [250073 usec]
104 java          0 29944 5.927430839 0.003036          return from _select [250071 usec]
104 java          0 29944 3.425409815 0.016963          return from _select [250064 usec]
104 java          0 29944 9.180150683 0.015228          return from _select [250064 usec]
104 java          0 29944 3.425409815 0.016963          return from _select [250064 usec]
104 java          0 29944 9.180150683 0.015228          return from _select [250064 usec]
104 java          0 29944 6.427796087 0.007108          return from _select [250062 usec]

```

This example shows some large return times from `syncd` and `java`. As the `syncd` only featured once, compared to the `java` process 29944, we look at the `java` process. `syncd` may have a lot of data to write to disk because of a problem with the `java` process, and therefore longer return times.

To look at process 29944 in more detail, we run the `trcrpt` command specifying process 29944 in the command line, as in Example 40-11.

Example 40-11 Traces for process 29944 (java)

```

# trcrpt -0 exec=on,pid=on,cpuid=on -o trcrpt.29944 -p 29944 -n trace.nm -t trace.fmt trace.r
# ls trcrpt.29944
trcrpt.29944

```

We can now look directly at the trace file called `trcrpt.29944` using an editor such as `vi` that is able to handle large files. Editing the trace file with `vi` might produce an error stating that there is not enough space in the file system. If you get this error, choose a file system with enough free space to edit the trace file (in this example, `/bigfiles` is the name of the file system), then run these commands:

```
mkdir /bigfiles/tmp ; echo "set dir=/bigfiles/tmp" > $HOME/.exrc
```

This directs `vi` to use the `/bigfiles/tmp` directory for temporary storage.

Attention: As some trace files may be large, be careful that you do not use all of the file system space, as this will cause problems for AIX and other applications running on the system.

From Example 40-10 on page 771 we know that we have a potential problem with process ID 29944 (java). We can now look further into the java process by producing a trace file specific to process 29944 as in the following example (the file we will create is called trcrpt.29944).

Search for the return time of 250117 usec (refer to Example 40-10 on page 771) in trcrpt.29944. This will display the events for the process as shown in Example 40-12.

Example 40-12 A traced routine call for process 29944

```
# cat trcrpt.29944
...(lines omitted)...
252 java      0 29944      1.674567306    0.003879      SOCK soo_select fp=10006FF0
so=7013B000 corl=12 reqevents=00000001 rtneventsp=F00EFA50
116 java      0 29944      1.674568077    0.000771      xmalloc(0020,30000000)
116 java      0 29944      1.674573257    0.005180      xmalloc(0020,30000000)
2F9 java      0 29944      1.674585184    0.011927      WLM STOP THREAD:
pid=29944 class=65 nb_us=112 time=11760
10E java     -1 29944      1.924587939    250.002755    relock: lock
addr=1945040 oldtid=517 newtid=40263
106 java      0 29944      1.924588685    0.000746      dispatch: cmd=java
pid=29944 tid=40263 priority=181 old_tid=517 old_priority=255 CPUID=0 [250117 usec]
200 java      0 29944      1.924589576    0.000891      resume java iar=43620
cpuid=00
104 java      0 29944      1.924604756    0.015180      return from _select [250042
usec]
...(lines omitted)...
```

A similar entry is repeated many times throughout the trace file (trcrpt.29944), suggesting that the same problem occurs many times throughout the trace.

For ease of reading, Example 40-12 has been split vertically, approximately halfway across the page, and shown separately in the next two examples.

Example 40-13 shows the left-hand side with the times.

Example 40-13 A traced routine call for process 29944 (left side)

ID	PROCESS NAME	CPU	PID	I	ELAPSED_SEC	DELTA_MSEC
252	java	0	29944		1.674567306	0.003879
116	java	0	29944		1.674568077	0.000771
116	java	0	29944		1.674573257	0.005180
2F9	java	0	29944		1.674585184	0.011927
10E	java	-1	29944		1.924587939	250.002755

106	java	0	29944	1.924588685	0.000746
200	java	0	29944	1.924589576	0.000891
104	java	0	29944	1.924604756	0.015180

The right-hand side with the system calls is shown in Example 40-14. The trace hooks have been left in to enable you to associate the two examples.

Example 40-14 A traced routine call for process 29944 (right side)

```

ID  SYSCALL  KERNEL  INTERRUPT
252 SOCK soo_select fp=10006FF0 so=7013B000 corl=12 reqevents=00000001 rtneventsp=F00EFA50
116 xmalloc(0020,30000000)
116 xmalloc(0020,30000000)
2F9 WLM STOP THREAD: pid=29944 class=65 nb_us=112 time=11760
10E relock: lock addr=1945040 oldtid=517 newtid=40263
106 dispatch:cmd=java pid=29944 tid=40263 priority=181 old_tid=517 old_priority=255 CPUID=0 [250117 usec]
200 resume java iar=43620 cpuid=00
104 return from _select [250042 usec]

```

As can be seen from the above example, when the java process was trying to reserve memory, the Workload Manager (WLM) stopped the thread from running, which caused a relock to occur. The relock took 250.002755 usec (microseconds). This should be investigated further. You could, in this instance, tune the WLM to allow more time for the java process to complete.

Sequential reads and writes

The **trace** command can be used to identify reads and writes to files.

When the trace report has been generated, you can determine the type of reads and writes that are occurring on file systems when the trace was run.

The following script is useful for displaying the type of file accesses. The script extracts readi and writei Physical File System (PFS) calls from the formatted trace and sorts the file in order of the ip field (Example 40-15).

Example 40-15 Script to sort PFS trace events

```

:
egrep "PFS writei|PFS readi" trcrpt.out > readwrite
> trcrpt.pfs
for ip in `cat readwrite | grep 'ip=' | awk -F'ip=' '{ print $2 }' | \
awk '{ print $1 }' | sort -u`
do
    grep "ip=$ip" readwrite >> trcrpt.pfs
done

```

The output from these scripts is shown in Example 40-16 on page 775.

Example 40-16 PFS file access in trace file

```
# cat trcrpt.pfs
...(lines omitted)...
PFS readi VA.S=0000 3CE000.293C5 bcount=2000 ip=1B160270
PFS readi VA.S=0000 3D0000.293C5 bcount=2000 ip=1B160270
PFS readi VA.S=0000 3D4000.293C5 bcount=2000 ip=1B160270
PFS readi VA.S=0000 3D6000.293C5 bcount=2000 ip=1B160270
PFS readi VA.S=0000 3D8000.293C5 bcount=2000 ip=1B160270
PFS readi VA.S=0000 3E0000.293C5 bcount=2000 ip=1B160270
...(lines omitted)...
```

This example shows that the file at IP address 1B160270 was read from with a block size of 8 KB reads (bcount=2000). By looking at the Virtual Address (VA) field, you will observe that the VA field mostly incremented by 2000 (the 2000 is expressed in hexadecimal). If you see this sequence then you know that the file is receiving a lot of sequential reads. In this case, it could be because that file does not have an index. For an application to read large files without indexes, in some cases, a full table scan is needed to retrieve records. In this case it would be advisable to index the file.

To determine what file is being accessed, it is necessary to map the ip to a file name. This is done with the **ps** command.

For efficiency, it is best to perform file accesses in multiples of 4 KB.

40.2 trcnm

The syntax of the **trcnm** command is:

```
trcnm [ -a [ FileName ] ] | [ FileName ] | -K Symbol ...
```

Flags

- a** Writes all loader symbols to standard output. The default is to write loader symbols only for system calls.
- K Symbol...** Obtains the value of all command line symbols through the knlist system call.

Parameters

- FileName** The kernel file that the **trcnm** command creates the name list for. If this parameter is not specified, the default FileName is /unix.

Symbol The name list will be created only for the specified symbols. To specify multiple symbols, separate the symbols by a space.

The **trcnm** command writes to standard output. When using the output from the **trcnm** command with the **trcrpt -n** command, save this latest output into a file.

40.2.1 Information about measurement and sampling

The **trcnm** command generates a list of symbol names and their addresses for the specified kernel file, or /unix if no kernel file is specified. The symbol names and addresses are read out of the kernel file. The output of the **trcnm** command is similar the output the **stripnm -x** command provides. The output format differs between these commands. Refer to Chapter 40, “The trace, trcnm, and trcrpt commands” on page 759 for more information about the **stripnm** command.

Note: The **trace** command flag **-n** gathers the necessary symbol information needed by the **trcrpt** command and stores this information in the trace log file. The symbol information gathered by **trace -n** includes the symbols from the loaded kernel extensions. The **trcnm** command provides only the symbol information for the kernel. The use of the **-n** flag of **trace** as a replacement for the **trcnm** command is recommended.

40.2.2 Examples for trcnm

The following command is used to create a name list for the kernel file /unix:

```
trcnm >/tmp/trcnm.out
```

To create the name list only for the kernel symbols **net_malloc** and **m_copym**, use the **trcnm -K net_malloc m_copym** command as shown in Example 40-17.

Example 40-17 Using trcnm to create the name list for specified symbols

```
# trcnm -K net_malloc m_copym
net_malloc      001C9FCC
m_copym        001CA11C
```

For each specified symbol the name and the address is printed.

40.3 trcrpt

The following syntax applies to the **trcrpt** command:

```
trcrpt [ -c ] [ -C [ CPUList | all ] ] [ -d List ]  
      [ -D Event-group-list ] [ -e Date ] [ -G ] [ -h ] [ -j ] [ -k List ]  
      [ -K Group-list ] [ -n Name ] [ -o File ] [ -p List ] [ -r ]  
      [ -s Date ] [ -t File ] [ -T List ] [ -v ] [ -O Options ] [-x ] [ File ]
```

Flags

- c** Checks the template file for syntax errors.
- C [CPUList | all]** Generates a report for a multi-CPU trace with **trace -C**. The CPUs can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks. To report on all CPUs, specify **trace -C all**. The **-C** flag is not necessary unless you want to see only a subset of the CPUs traced or have the CPU number show up in the report. If **-C** is not specified, and the trace is a multi-CPU trace, **trcrpt** generates the trace report for all CPUs, but the CPU number is not shown for each hook unless you specify **-O cpu=on**.
- d List** Limits report to hook IDs specified with the List variable. The List parameter items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.
- D Event-group-list** Limits the report to hook IDs in the Event groups list, plus any hook IDs specified with the **-d** flag. List parameter items can be separated by commas or enclosed in double quotation marks and separated by commas or blanks.
- e Date** Ends the report time with entries on or before the specified date. The Date variable has the form **mmddhhmssyy** (month, day, hour, minute, second, and year). Date and time are recorded in the trace data only when trace data collection is started and stopped. If you stop and restart trace data collection multiple times during a trace session, date and time are recorded each time you start or stop a trace data collection. Use this flag in combination with the **-s** flag to limit the trace to data collected during a certain time interval.
- If you specify **-e** with **-C**, the **-e** flag is ignored.
- G** List all event groups. The list of groups, the hook IDs in each group, and each group's description is listed to standard output.

-h	Omits the header information from the trace report and writes only formatted trace entries to standard output.
-j	Displays the list of hook IDs. The trcrpt -j command can be used with the trace -j command that includes IDs of trace events, or the trace -k command that excludes IDs of trace events.
-k List	Excludes from the report hook IDs specified with the List variable. The List parameter items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.
-K Event-group-list	Excludes from the report hook IDs in the event-groups list, plus any hook IDs specified with the -k flag. List parameter items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.
-n Name	Specifies the kernel name list file to be used to interpret addresses for output. Usually this flag is used when moving a trace log file to another system.
-o File	Writes the report to a file instead of to standard output.
-O Options	Specifies options that change the content and presentation of the trcrpt command. Arguments to the options must be separated by commas. Valid options are: <ul style="list-style-type: none"> • 2line=[on off] Uses two lines per trace event in the report instead of one. The default value is off. • cpuid=[on off] Displays the physical processor number in the trace report. The default value is off. • endtime=Seconds Displays trace report data for events recorded before the seconds specified. Seconds can be given in either an integral or rational representation. If this option is used with the starttime option, a specific range can be displayed. • exec=[on off] Displays exec path names in the trace report. The default value is off. • hist=[on off] Logs the number of instances that each hook ID is encountered. This data can be used for generating histograms. The default value is off. This option cannot be run with any other option.

- `ids=[on|off]`
Displays trace hook identification numbers in the first column of the trace report. The default value is on.
- `pagesize=Number`
Controls the number of lines per page in the trace report and is an integer in the range of 0 through 500. The column headings are included on each page. No page breaks are present when the default value (zero) is set.
- `pid=[on|off]`
Displays the process IDs in the trace report. The default value is off.
- `reportedcpus=[on|off]`
Displays the number of CPUs remaining. This option is only meaningful for a multi-CPU trace; that is, if the trace was performed with the `-C` flag. For example, if a report is read from a system having four CPUs, and the reported CPUs value goes from four to three, then you know that there are no more hooks to be reported for that CPU.
- `starttime=Seconds`
Displays trace report data for events recorded after the seconds specified. The specified seconds are from the beginning of the trace file. Seconds can be given in either an integral or rational representation. If this option is used with the `endtime` option, a specific range of seconds can be displayed.
- `svc=[on|off]`
Displays the value of the system call in the trace report. The default value is off.
- `tid=[on|off]`
Displays the thread ID in the trace report. The default value is off.
- `timestamp=[0|1|2|3]`
Controls the time stamp associated with an event in the trace report. The possible values are:
 - 0** Time elapsed since the trace was started. Values for elapsed seconds and milliseconds are returned to the nearest nanosecond and microsecond, respectively. This is the default value.
 - 1** Short elapsed time.
 - 2** Microseconds.
 - 3** No time stamp.

- p List** Reports the process IDs for each event specified by the List variable. The List variable may be a list of process IDs or a list of process names. List items that start with a numeric character are assumed to be process IDs. The list items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.
- r** Outputs unformatted (raw) trace entries and writes the contents of the trace log to standard output one entry at a time. Use the -h flag with the -r flag to exclude the heading. To get a raw report for CPUs in a multi-CPU trace, use both the -r and -C flags.
- s Date** Starts the report time with entries on or before the specified date. The Date variable has the form mmddhhmmssyy (month, day, hour, minute, second, and year). Date and time are recorded in the trace data only when trace data collection is started and stopped. If you stop and restart trace data collection multiple times during a trace session, date and time are recorded each time you start or stop a trace data collection. Use this flag in combination with the -e flag to limit the trace to data collected during a certain time interval.
- If you specify -s with -C, the -s flag is ignored.
- t File** Uses the file specified in the File variable as the template file. The default is the /etc/trcfmt file.
- T List** Limits the report to the kernel thread IDs specified by the List parameter. The list items are kernel thread IDs separated by commas. Starting the list with a kernel thread ID limits the report to all kernel thread IDs in the list. Starting the list with a ! (exclamation point) followed by a kernel thread ID limits the report to all kernel thread IDs not in the list.
- v** Prints file names as the files are opened. Changes to verbose setting.
- x** Displays the exec path name and value of the system call.

Parameters

File Name of the raw trace file.

40.3.1 Information about measurement and sampling

The **trcrpt** command reads the trace log specified by the File parameter, formats the trace entries, and writes a report to standard output. The default file from which the system generates a trace report is the `/var/adm/ras/trcfile` file, but you can specify an alternate File parameter.

40.3.2 Examples for trcrpt

You can use the System Management Interface Tool (SMIT) to run the **trcrpt** command by entering the SMIT fast path **smitty trcrpt**.

Example 40-18 shows how to run **trcrpt** using `/var/adm/ras/trcfile` as the raw trace file.

Example 40-18 Running trcrpt via SMIT

Generate a Trace Report

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[Entry Fields]		
Show exec PATHNAMES for each event?	[yes]	+
Show PROCESS IDs for each event?	[yes]	+
Show THREAD IDs for each event?	[yes]	+
Show CURRENT SYSTEM CALL for each event?	[yes]	+
Time CALCULATIONS for report	[elapsed+delta in milli>	+
Event Groups to INCLUDE in report	<input type="checkbox"/>	+
IDs of events to INCLUDE in report	<input type="checkbox"/>	+X
Event Groups to EXCLUDE from report	<input type="checkbox"/>	+
ID's of events to EXCLUDE from report	<input type="checkbox"/>	+X
STARTING time	<input type="checkbox"/>	
ENDING time	<input type="checkbox"/>	
LOG FILE to create report from	[/var/adm/ras/trcfile]	
FILE NAME for trace report (default is stdout)	<input type="checkbox"/>	

Combining trace buffers

Normally, **trace** groups all CPU buffers into one trace file. If you run **trace** with the **-C** all option, then the events that occurred on the individual CPUs will be separated into CPU-specific files as in the following example. To run **trcrpt** to format the trace into a readable file, you must combine the raw trace files into one raw trace file., then you can remove the specific raw trace files, as these are no longer required and usually are quite large in size. Example 40-19 on page 782 shows this procedure.

Example 40-19 Tracing using one set of buffers per CPU

```
# trace -aC all ; sleep 10 ; trcstop
# ls -l /var/adm/ras/trcfile*
-rw-rw-rw- 1 root system 44468 Apr 16 12:36 /var/adm/ras/trcfile
-rw-rw-rw- 1 root system 598956 Apr 16 12:37 /var/adm/ras/trcfile-0
-rw-rw-rw- 1 root system 369984 Apr 16 12:37 /var/adm/ras/trcfile-1
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-10
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-11
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-12
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-13
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-14
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-15
-rw-rw-rw- 1 root system 394728 Apr 16 12:37 /var/adm/ras/trcfile-2
-rw-rw-rw- 1 root system 288744 Apr 16 12:37 /var/adm/ras/trcfile-3
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-4
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-5
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-6
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-7
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-8
-rw-rw-rw- 1 root system 184 Apr 16 12:36 /var/adm/ras/trcfile-9
# trcrpt -C all -r /var/adm/ras/trcfile > trace.r
# ls -l trace.r
-rw-r--r-- 1 root system 1694504 Apr 16 13:55 trace.r
# trcrpt -O exec=on,pid=on,cpuid=on -n trace.nm -t trace.fmt trace.r >
trcrpt.out
# head -10 trcrpt.out

Wed Apr 16 12:36:57 2003
System: AIX 5.2 Node: lpar05
Machine: 0021768A4C00
Internet Address: 09030445 9.3.4.69
The system contains 16 cpus, of which 16 were traced.
Buffering: Kernel Heap
This is from a 32-bit kernel.
Tracing all hooks.

# rm /var/adm/ras/trcfile*
# trcnm > trace.nm
# cp /etc/trcfmt trace.fmt
# trcrpt -O exec=on,pid=on,cpuid=on -n trace.nm -t trace.fmt trace.r >
trcrpt.out
# head trcrpt.out
...(lines omitted)...
```

For other examples, refer to 40.1.4, “Examples for trace” on page 770.



Part 8

Additional performance topics

APIs for performance monitoring

In this chapter we describe how to use the different Application Programming Interfaces (API) that are available. It contains information about how to use the Perfstat API to develop customized performance monitoring applications. We also describe the basic use of the System Performance Measurement Interface (SPMI) API and the Performance Monitor (PM) API. Additionally, we describe the Resource Monitoring and Control (RMC) subsystem and its use. Finally, we show some examples of using other performance-monitoring subroutines that are available on AIX.

This chapter contains the following sections:

- ▶ 41.1, “Perfstat API” on page 786
- ▶ 41.2, “System Performance Measurement Interface” on page 805
- ▶ 41.3, “Performance Monitor API” on page 818
- ▶ 41.4, “Resource Monitoring and Control” on page 824
- ▶ 41.5, “Miscellaneous performance monitoring subroutines” on page 842

41.1 Perfstat API

The Perfstat API is a collection of C programming language subroutines that execute in user space and extract data from the perfstat kernel extension (kex) to obtain statistics. This API is available in AIX 5L.

The Perfstat API is both a 32-bit and a 64-bit API, and is thread safe, very simple to use, and does not require root security level authentication. It is the preferred way to develop monitoring applications, and the kex is also used by most system monitoring commands. The API is under development, and will have additional API subroutines and data structures in future release. Note that the internal perfstat kex access mechanisms are not available. Only the Perfstat Library API will be maintained for public use.

The Perfstat API subroutines resides in the `libperfstat.a` library in the `/usr/lib` (or `/lib` because `/lib` is a symbolic link to `/usr/lib`) and is part of the `bos.perf.libperfstat` fileset, which is installable from the AIX base installation media and requires that the `bos.perf.perfstat` fileset is installed.

The `/usr/include/libperfstat.h` file contains the subroutine declarations and type definitions of the data structures to use when calling the subroutines. This include file is also part of the `bos.perf.libperfstat` fileset. Sample source code is also available and resides in the `/usr/samples/libperfstat` directory.

The documentation for the subroutines can be found in the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1*.

41.1.1 Compiling and linking

After writing a C program that uses the Perfstat API and includes the `libperfstat.h` header file, run `cc` on it specifying that you want to link to the `libperfstat.a` library, as in Example 41-1.

Example 41-1 Compile and link with libperfstat.a

```
# cc -lperfstat -o perfstat_program perfstat_program.c
```

This creates the `perfstat_program` file from the `perfstat_program.c` source program, linking it with the `libperfstat.a` library. Then **perfstat_program** can be run as a normal command.

41.1.2 Subroutines

The following subroutines make up the Perfstat API:

perfstat_cpu	The <code>perfstat_cpu</code> subroutine retrieves one or more individual CPU usage statistics. The same function can be used to retrieve the number of available sets of CPU statistics.
perfstat_cpu_total	The <code>perfstat_cpu_total</code> subroutine returns global CPU usage statistics.
perfstat_memory_total	The <code>perfstat_memory_total</code> subroutine returns global memory usage statistics.
perfstat_disk	The <code>perfstat_disk</code> subroutine retrieves one or more individual disk usage statistics. The same function can also be used to retrieve the number of available sets of disk statistics.
perfstat_disk_total	The <code>perfstat_disk_total</code> subroutine returns global disk usage statistics.
perfstat_netinterface	The <code>perfstat_netinterface</code> subroutine retrieves one or more individual network interface usage statistics. The same function can also be used to retrieve the number of available sets of network interface statistics.
perfstat_netinterface_total	The <code>perfstat_netinterface_total</code> subroutine returns global network interface usage statistics.

Note: The Perfstat API subroutines return raw data. To create output similar to what is reported by commands such as `iostat` and `vmstat`, take a snapshot, wait for a specified interval of time, then take another snapshot. After this, deduct the first obtained value from the second to get the proper delta for the occurrence during the specified interval time. The `libperfstat.h` file should be reviewed to identify the units of each metric.

The `perfstat` API only gives raw data. The Perfstat API enables you to acquire the data quite easily as can be seen in the following sample programs. Only rudimentary error checking is done in the example program. This is done for clarity of reading purposes only. Another sample program that calls all the APIs are provided in “`perfstat_dump_all.c`” on page 936.

perfstat_cpu

The `perfstat_cpu` subroutine retrieves one or more individual CPU usage statistics. The same function can be used to retrieve the number of available sets of CPU statistics.

```
perfstat_id_t * name;
perfstat_cpu_t * userbuff;
int sizeof_struct;
int desired_number;
```

```
int perfstat_cpu (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	Contains a name identifying the first CPU for which statistics are desired. "" is used to indicate the first available CPU. For example: <code>cpu0</code> , <code>cpu1</code> , and so on.
userbuff	Points to the memory area that is to be filled with one or more <code>perfstat_cpu_t</code> structures.
sizeof_struct	Specifies the size of the <code>perfstat_cpu_t</code> structure: <code>sizeof(perfstat_cpu_t)</code> .
desired_number	Specifies the number of <code>perfstat_cpu_t</code> structures to copy to <code>userbuff</code> .

Example

Example 41-2 shows a code that uses the `perfstat_cpu_t` structure to obtain information about CPU statistics.

Example 41-2 Sample `perfstat_cpu_t` program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>

4  main()
5  {
6      perfstat_id_t   name;
7      perfstat_cpu_t *ub;
8      int             ncpu,i;

9      ncpu = perfstat_cpu (NULL,NULL,sizeof(perfstat_cpu_t),0);
10     ub = malloc(sizeof(perfstat_cpu_t)*ncpu);

11     strcpy(name.name,"");

12     if (perfstat_cpu(&name,ub,sizeof(perfstat_cpu_t),ncpu) >= 0)
13         for (i = 0; i < ncpu; i++) {
14             printf("name      : %s\n",  ub[i].name);
```

```

15         printf("\tuser      : %llu\n", ub[i].user);
16         printf("\tsys       : %llu\n", ub[i].sys);
17         printf("\tidle      : %llu\n", ub[i].idle);
18         printf("\twait      : %llu\n", ub[i].wait);
19         printf("\tpswitch  : %llu\n", ub[i].pswitch);
20         printf("\tsyscall  : %llu\n", ub[i].syscall);
21         printf("\tsysread  : %llu\n", ub[i].sysread);
22         printf("\tsyswrite: %llu\n", ub[i].syswrite);
23         printf("\tsysfork  : %llu\n", ub[i].sysfork);
24         printf("\tsysexec  : %llu\n", ub[i].sysexec);
25         printf("\treadch   : %llu\n", ub[i].readch);
26         printf("\twritech  : %llu\n", ub[i].writech);
27     }
28 }

```

On line 3 the `libperfstat.h` declaration file is included. Then on lines 6 and 7 we declare the variables for calling the `perfstat_cpu` subroutine, which we do on line 12. Note how the usage and reference of structures is done in the call. The first call to `perfstat_cpu` is done to acquire the number of CPUs in the system. This is then used to allocate the appropriate number of structures, with `malloc`, to store the information for each CPU.

The output from the program can look like Example 41-3.

Example 41-3 Sample output from the `perfstat_cpu_t` program

```

# perfstat_cpu_t
name      : cpu0
  user    : 63584
  sys     : 29732
  idle    : 13419287
  wait    : 20660
  pswitch : 2122965
  syscall : 6498220
  sysread : 978004
  syswrite: 607014
  sysfork : 3536
  sysexec : 4666
  readch  : 976572598
  writech : 335808673
...(lines omitted)...
name      : cpu3
  user    : 194219
  sys     : 34758
  idle    : 13063504
  wait    : 35837
  pswitch : 2230810
  syscall : 15141865
  sysread : 754259

```

```
syswrite: 474751
sysfork : 6391
sysexec : 4903
readch  : 1583351139
writech : 490560773
```

These are definitions of each structure element:

name	CPU name (cpu0, cpu1, and so on)
user	CPU user time (raw ticks)
sys	CPU sys time (raw ticks)
idle	CPU idle time (raw ticks)
wait	CPU wait time (raw ticks)
pswitch	Incremented whenever the current running process changes
syscall	Number of syscalls
sysread	Number of readings
syswrite	Number of writings
sysfork	Number of forks
sysexec	Number of execs
readch	Number of bytes read by CPU
writech	Number of bytes written by CPU

perfstat_cpu_total

The perfstat_cpu_total subroutine returns global CPU usage statistics.

```
perfstat_id_t * name;
perfstat_cpu_total_t * userbuff;
int sizeof_struct;
int desired_number;
```

```
int perfstat_cpu_total (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	In AIX 5.2, this must always be set to NULL.
userbuff	Points to the memory area that is to be filled with the perfstat_cpu_total_t structure.
sizeof_struct	Specifies the size of the perfstat_cpu_total_t structure: sizeof(perfstat_cpu_total_t).
desired_number	In AIX 5.2, this must always be set to 1.

Example

The code in Example 41-4 on page 791 uses the perfstat_cpu_total_t structure to obtain information about CPU statistics.

Example 41-4 Sample perfstat_cpu_total_t program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>

4 main()
5 {
6     perfstat_cpu_total_t    ub;

7     if (perfstat_cpu_total ((perfstat_id_t*)NULL, &ub,
sizeof(perfstat_cpu_total_t),1) >= 0) {
8         printf("ncpus      : %d\n", ub.ncpus);
9         printf("ncpus_cfg   : %d\n", ub.ncpus_cfg);
10        printf("description : %s\n", ub.description);
11        printf("processorHZ : %llu\n", ub.processorHZ);
12        printf("user       : %llu\n", ub.user);
13        printf("sys       : %llu\n", ub.sys);
14        printf("idle     : %llu\n", ub.idle);
15        printf("wait    : %llu\n", ub.wait);
16        printf("pswitch  : %llu\n", ub.pswitch);
17        printf("syscall  : %llu\n", ub.syscall);
18        printf("sysread  : %llu\n", ub.sysread);
19        printf("syswrite : %llu\n", ub.syswrite);
20        printf("sysfork  : %llu\n", ub.sysfork);
21        printf("sysexec  : %llu\n", ub.sysexec);
22        printf("readch   : %llu\n", ub.readch);
23        printf("writech  : %llu\n", ub.writech);
24        printf("devintrs : %llu\n", ub.devintrs);
25        printf("softintrs : %llu\n", ub.softintrs);
26        printf("lbolt    : %ld\n", ub.lbolt);
27        printf("loadavg T0 : %llu\n", ub.loadavg[0]);
28        printf("loadavg T-5 : %llu\n", ub.loadavg[1]);
29        printf("loadavg T-15: %llu\n", ub.loadavg[2]);
30        printf("runque   : %llu\n", ub.runque);
31        printf("swpque   : %llu\n", ub.swpque);
32    }
33 }
```

On line 3 the `libperfstat.h` declaration file is included. Then on line 6 we declare the only variable we need for calling the `perfstat_cpu_total` subroutine, which we do on line 7. Note how the usage and reference of structures is done in the call, especially the reference to `NULL` for the pointer to the `perfstat_id_t` reference. The output from the program will look like Example 41-5 on page 792.

Example 41-5 Sample output from the perfstat_cpu_total_t program

```
# perfstat_cpu_total_t
ncpus      : 4
ncpus_cfg  : 4
description : PowerPC_POWER4
processorHZ : 1100152416
user       : 23987733
sys        : 1332681
idle       : 146744848
wait       : 10208601
pswitch    : 304218689
syscall    : 1069171832
sysread    : 86134624
syswrite   : 91759560
sysfork    : 122134
sysexec    : 152505
readch     : 30225867708
writtech   : 21921375932
devintrs   : 0
softintrs  : 0
lbolt      : 45567458
loadavg T0 : 132391
loadavg T-5 : 132552
loadavg T-15 : 132367
runque     : 295136
swpque     : 385398
```

The following are definitions of each structure element:

ncpus	Number of active CPUs
ncpus_cfg	Number of configured CPUs
description	CPU description
processorHZ	CPU speed in Hz
user	CPU user time (raw ticks)
sys	CPU sys time (raw ticks)
idle	CPU idle time (raw ticks)
wait	CPU wait time (raw ticks)
pswitch	Number of changes of the current running process
syscall	Number of syscalls executed
sysread	Number of readings
syswrite	Number of writings
sysfork	Number of forks
sysexec	Number of execs
readch	Total number of bytes read
writtech	Total number of bytes written
devintrs	Total number of interrupts
softintrs	Total number of software interrupts

lbolt	Number of ticks since last reboot
loadavg	Load average now, last 5 minutes, last 15 minutes
runque	Average length of the run queue
swpque	Average length of the swap queue

perfstat_memory_total

The perfstat_memory_total subroutine returns global memory usage statistics.

```
perfstat_id_t * name;
perfstat_memory_total_t * userbuff;
int sizeof_struct;
int desired_number;
```

```
int perfstat_memory_total (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	In AIX 5.2, this must always be set to NULL.
userbuff	Points to the memory area that is to be filled with the perfstat_memory_total_t structures.
sizeof_struct	Specifies the size of the perfstat_memory_total_t structure; sizeof(perfstat_memory_total_t).
desired_number	In AIX 5.2, this must always be set to 1.

Example

The code in Example 41-6 uses the perfstat_memory_total_t structure to obtain information about memory statistics.

Example 41-6 Sample perfstat_memory_total_t program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>
4 main()
5 {
6     perfstat_memory_total_t ub;
7     if (perfstat_memory_total ((perfstat_id_t*)NULL, &ub,
8 sizeof(perfstat_memory_total_t),1) >= 0) {
9         printf("virt_total: %llu\n", ub.virt_total);
10        printf("real_total: %llu\n", ub.real_total);
11        printf("real_free : %llu\n", ub.real_free);
12        printf("real_inuse: %llu\n", ub.real_inuse);
13        printf("pgbad      : %llu\n", ub.pgbad);
14        printf("pgexct     : %llu\n", ub.pgexct);
15        printf("pgins      : %llu\n", ub.pgins);
16        printf("pgouts     : %llu\n", ub.pgouts);
```

```

16     printf("pgspins   : %llu\n", ub.pgspins);
17     printf("pgspouts  : %llu\n", ub.pgspouts);
18     printf("scans     : %llu\n", ub.scans);
19     printf("cycles    : %llu\n", ub.cycles);
20     printf("pgsteals  : %llu\n", ub.pgsteals);
21     printf("numperm   : %llu\n", ub.numperm);
22     printf("pgsp_total: %llu\n", ub.pgsp_total);
23     printf("pgsp_free  : %llu\n", ub.pgsp_free);
24     printf("pgsp_rsvd : %llu\n", ub.pgsp_rsvd);
25 }
26 }

```

On line 3 the `libperfstat.h` declaration file is included. Then on line 6 we declare variables for calling the `perfstat_memory_total` subroutine, which we do on line 7. Note how the usage and reference of structures is done in the call. The output from the program can look like Example 41-7.

Example 41-7 Sample output from the `perfstat_memory_total_t` program

```

# perfstat_memory_total_t
virt_total: 2621440
real_total: 2097152
real_free  : 911629
real_inuse: 1185523
pgbad     : 9
pgexct    : 298073502
pgins     : 5095811
pgouts    : 21968950
pgspins   : 4524147
pgspouts  : 19319465
scans     : 52989124
cycles    : 24
pgsteals  : 19718704
numperm   : 649872
pgsp_total: 524288
pgsp_free : 320133
pgsp_rsvd : 2048

```

These are definitions of each structure element:

<code>virt_total</code>	Total virtual memory (4K pages)
<code>real_total</code>	Total real memory (4K pages)
<code>real_free</code>	Free real memory (4K pages)
<code>real_pinned</code>	Real memory that is pinned (4K pages)
<code>real_inuse</code>	Real memory that is in use (4K pages)
<code>pgbad</code>	Count of bad pages
<code>pgexct</code>	Count of page faults
<code>pgins</code>	Count of pages paged in

pgouts	Count of pages paged out
pgspins	Count of page ins from paging space
pgspouts	Count of page outs from paging space
scans	Count of page scans by clock
cycles	Count of clock hand cycles
pgsteals	Count of page steals
numperm	Number of non-working frames
pgsp_total	Total paging space (4K pages)
pgsp_free	Free paging space (4K pages)
pgsp_rsvd	Reserved paging space (4K pages)

perfstat_disk

The `perfstat_disk` subroutine retrieves one or more individual disk usage statistics. The same function can also be used to retrieve the number of available sets of disk statistics.

```
perfstat_id_t * name;
perfstat_disk_t * userbuff;
int sizeof_struct;
int desired_number;
```

```
int perfstat_disk (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	Contains a name identifying the first disk for which statistics are desired. "" is used to indicate the first available disk. For example: <code>hdisk0</code> , <code>hdisk1</code> , and so on.
userbuff	Points to the memory area that is to be filled with one or more <code>perfstat_disk_t</code> structures.
sizeof_struct	Specifies the size of the <code>perfstat_disk_t</code> structure; <code>sizeof(perfstat_cpu_t)</code> .
desired_number	Specifies the number of <code>perfstat_disk_t</code> structures to copy to <code>userbuff</code> .

Example

The code in Example 41-8 uses the `perfstat_disk_t` structure to obtain information about disk statistics.

Example 41-8 Sample `perfstat_disk_t` program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>

4 main()
5 {
```

```

6     perfstat_id_t   name;
7     perfstat_disk_t *ub;
8     int             ndisk,i;

9     ndisk = perfstat_disk (NULL,NULL,sizeof(perfstat_disk_t),0);
10    ub = malloc(sizeof(perfstat_disk_t)*ndisk);

11    strcpy(name.name,"");

12    if (perfstat_disk (&name,ub,sizeof(perfstat_disk_t),ndisk) >= 0)
13        for (i = 0; i < ndisk; i++) {
14            printf("name       : %s\n", ub[i].name);
15            printf("\tdescription: %s\n", ub[i].description);
16            printf("\tvpname   : %s\n", ub[i].vgname);
17            printf("\tsize     : %llu\n", ub[i].size);
18            printf("\tfree     : %llu\n", ub[i].free);
19            printf("\tbsize    : %llu\n", ub[i].bsize);
20            printf("\txrate    : %llu\n", ub[i].xrate);
21            printf("\txfers    : %llu\n", ub[i].xfers);
22            printf("\twblks    : %llu\n", ub[i].wblks);
23            printf("\trblks    : %llu\n", ub[i].rblks);
24            printf("\tqdepth   : %llu\n", ub[i].qdepth);
25            printf("\ttime     : %llu\n", ub[i].time);
26        }
27 }

```

On line 3 the `libperfstat.h` declaration file is included. Then on lines 6 and 7 we declare variables for calling the `perfstat_disk` subroutine, which we do on line 12. Note how the usage and reference of structures is done in the call. The first call to `perfstat_disk` is done to acquire the number of available sets of disk statistics in the system. This is then used to allocate the appropriate number of structures to keep the information for each statistics set with `malloc`. The output from the program will look something like Example 41-9.

Example 41-9 Sample output from the `perfstat_disk_t` program

```

# perfstat_disk_t
name       : hdisk1
description: 16 Bit SCSI Disk Drive
vgname    : vg0
size      : 8672
free      : 7936
bsize     : 512
xrate     : 0
xfers     : 14104
wblks     : 148913
rblks     : 1298481
qdepth    : 0
time      : 7498

```

```
...(lines omitted)...
name      : cd0
description: SCSI Multimedia CD-ROM Drive
vgname    : None
size      : 0
free      : 0
bsize     : 512
xrate     : 0
xfers     : 0
wblks     : 0
rblks     : 0
qdepth    : 0
time      : 0
```

These are definitions for each structure element:

name	Name of the disk
description	Disk description
vgname	Volume group name
size	Size of the disk (MB)
free	Free portion of the disk (MB)
bsize	Disk block size (bytes)
xrate	KB/sec xfer rate capability
xfers	Total transfers to/from disk
wblks	Blocks written to disk
rblks	Blocks read from disk
qdepth	Queue depth
time	Amount of time disk is active

perfstat_disk_total

The `perfstat_disk_total` subroutine returns global disk usage statistics.

```
perfstat_id_t * name;
perfstat_disk_total_t * userbuff;
int sizeof_struct;
int desired_number;

int perfstat_disk_total (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	In AIX 5.2, this must always be set to NULL.
userbuff	Points to the memory area that is to be filled with one or more <code>perfstat_disk_total_t</code> structures.
sizeof_struct	Specifies the size of the <code>perfstat_disk_total_t</code> structure; <code>sizeof(perfstat_cpu_t)</code> .
desired_number	In AIX 5.2, this must always be set to 1.

Example

The code in Example 41-10 uses the `perfstat_disk_total_t` structure to obtain information about disk statistics.

Example 41-10 Sample `perfstat_disk_total_t` program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>

4 main()
5 {
6     perfstat_disk_total_t  ub;

7     if (perfstat_disk_total ((perfstat_id_t*)NULL, &ub,
sizeof(perfstat_disk_total_t),1) >= 0) {
8         printf("number: %d\n", ub.number);
9         printf("size  : %llu\n", ub.size);
10        printf("free  : %llu\n", ub.free);
11        printf("xrate : %llu\n", ub.xrate);
12        printf("xfers : %llu\n", ub.xfers);
13        printf("wblks : %llu\n", ub.wblks);
14        printf("rblks : %llu\n", ub.rblks);
15        printf("time  : %llu\n", ub.time);
16    }
17 }
```

On line 3 the `libperfstat.h` declaration file is included. Then on line 6 we declare variables for calling the `perfstat_disk_total` subroutine, which we do on line 7. Note how the usage and reference of structures is done in the call. The output from the program will look like Example 41-11.

Example 41-11 Sample output from the `perfstat_disk_total_t` program

```
# perfstat_disk_total_t
number: 5
size  : 34688
free  : 23520
xrate : 0
xfers : 254296
wblks : 3447164
rblks : 5065261
time  : 168958
```

These are definitions of each structure element as displayed above.

number	Number of disks
size	Size of the disks (MB)
free	Free portion of the disks (MB)

xrate	Average kbytes/sec xfer rate capability
xfers	Total transfers to/from disks
wblks	Blocks written to all disks
rblks	Blocks read from all disks
time	Amount of time disk is active

perfstat_netinterface

The `perfstat_netinterface` subroutine retrieves one or more individual network interface usage statistics. The same function can also be used to retrieve the number of available sets of network interface statistics.

```
perfstat_id_t * name;
perfstat_netinterface_t * userbuff;
int sizeof_struct;
int desired_number;

int perfstat_netinterface (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	Contains a name identifying the first network interface for which statistics are desired. "" is used to specify the first available interface. For example: en0, tr1, and so on.
userbuff	Points to the memory area that is to be filled with one or more <code>perfstat_netinterface_t</code> structures.
sizeof_struct	Specifies the size of the <code>perfstat_netinterface_t</code> structure; <code>sizeof(perfstat_cpu_t)</code> .
desired_number	Specifies the number of <code>perfstat_netinterface_t</code> structures to copy to <code>userbuff</code> .

Example

The code in Example 41-12 uses the `perfstat_netinterface_t` structure to obtain information about network statistics.

Example 41-12 Sample perfstat_netinterface_t program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>
4 main()
5 {
6     perfstat_id_t      name;
7     perfstat_netinterface_t *ub;
8     int                nnetinterface,i;
9     nnetinterface = perfstat_netinterface
(NULL,NULL,sizeof(perfstat_netinterface_t),0);
10    ub = malloc(sizeof(perfstat_netinterface_t)*nnetinterface);
```

```

11     strcpy(name.name, "");
12     if (perfstat_netinterface
(&name, ub, sizeof(perfstat_netinterface_t), nnetinterface) >= 0)
13         for (i = 0; i < nnetinterface; i++) {
14             printf("name      : %s\n",    ub[i].name);
15             printf("\tdescription: %s\n",  ub[i].description);
16             printf("\ttype       : %u\n",    ub[i].type);
17             printf("\tmtu       : %llu\n",   ub[i].mtu);
18             printf("\tipackets  : %llu\n",   ub[i].ipackets);
19             printf("\tbytes    : %llu\n",   ub[i].ibytes);
20             printf("\tierrors  : %llu\n",   ub[i].ierrors);
21             printf("\topackets : %llu\n",   ub[i].opackets);
22             printf("\tbodytes  : %llu\n",   ub[i].obytes);
23             printf("\toerrors  : %llu\n",   ub[i].oerrors);
24             printf("\tcollisions : %llu\n",   ub[i].collisions);
25         }
26 }

```

On line 3 the `libperfstat.h` declaration file is included. Then on lines 6 and 7 we declare variables for calling the `perfstat_netinterface` subroutine, which we do on line 9. Note how the usage and reference of structures is done in the call. The first call to `perfstat_netinterface` is done to acquire the number of network interfaces in the system. This is then used to allocate the appropriate number of structures to keep the information for each network interface with `malloc`.

The output from the program will look something like Example 41-13.

Example 41-13 Sample output from the `perfstat_netinterface_t` program

```

# perfstat_netinterface_t
name      : tr0
description: Token Ring Network Interface
type      : 9
mtu       : 1492
ipackets  : 764483
ibytes    : 153429823
ierrors   : 0
opackets  : 499053
obytes    : 93898923
oerrors   : 0
collisions : 0
name      : en0
description: Standard Ethernet Network Interface
type      : 6
mtu       : 1500
ipackets  : 0
ibytes    : 0
ierrors   : 0
opackets  : 3

```

```

        obytes      : 180
        oerrors     : 3
        collisions  : 0
name      : lo0
description: Loopback Network Interface
type      : 24
mtu       : 16896
ipackets  : 17501
ibytes    : 2031836
ierrors   : 0
opackets  : 17501
obytes    : 2031432
oerrors   : 0
collisions : 0

```

The output shows only raw data. The Perfstat API enables you to acquire the data quite easily, as can be seen in the program in Example 41-12 on page 799. Note that the type value of 9, in the output above for token-ring, translates in hex to IS088025 or token-ring as can be seen in Table 41-1.

The following is a short definition of each structure element as displayed above:

```

name           Name of the interface
description    Interface description (lscfg type output)
type          Interface types: see /usr/include/net/if_types.h or Table 41-1
mtu           Network frame size
ipackets      Packets received on interface
ibytes        Bytes received on interface
ierrors       Input errors on interface
opackets      Packets sent on interface
obytes        Bytes sent on interface
oerrors       Output errors on interface
collisions    Collisions on CSMA interface

```

Table 41-1 Interface types from *if_types.h*

Name	Type	Name	Type
1822	0x2	DS3	0x1e
HDH1822	0x3	SIP	0x1f
X25DDN	0x4	FRELAY	0x20
X25	0x5	RS232	0x21
ETHER	0x6	PARA	0x22
OTHER	0x1	ULTRA	0x1d

Name	Type	Name	Type
ISO88023	0x7	ARCNET	0x23
ISO88024	0x8	ARCNETPLUS	0x24
ISO88025	0x9	ATM	0x25
ISO88026	0xa	MIOX25	0x26
STARLAN	0xb	SONET	0x27
P10	0xc	X25PLE	0x28
P80	0xd	ISO88022LLC	0x29
HY	0xe	LOCALTALK	0x2a
FDDI	0xf	SMDSDXI	0x2b
LAPB	0x10	FRELAYDCE	0x2c
SDLC	0x11	V35	0x2d
T1	0x12	HSSI	0x2e
CEPT	0x13	HIPPI	0x2f
ISDNBASIC	0x14	MODEM	0x30
ISDNPRIMARY	0x15	AAL5	0x31
PTPSERIAL	0x16	SONETPATH	0x32
PPP	0x17	SONETVT	0x33
LOOP	0x18	SMDSICIP	0x34
EON	0x19	PROPVIRTUAL	0x35
XETHER	0x1a	PROPMUX	0x36
NSIP	0x1b	VIPA	0x37
SLIP	0x1c		

perfstat_netinterface_total

The perfstat_netinterface_total subroutine returns global network interface usage statistics.

```
perfstat_id_t * name;
perfstat_netinterface_total_t * userbuff;
int sizeof_struct;
```



```
int desired_number;
```

```
int perfstat_netinterface_total (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	In AIX 5.2, this must always be set to NULL.
userbuff	Points to the memory area that is to be filled with the perfstat_netinterface_total_t structure.
sizeof_struct	Specifies the size of the perfstat_netinterface_total_t structure; sizeof(perfstat_netinterface_total_t).
desired_number	In AIX 5.2, this must always be set to 1.

Example

The code in Example 41-14 uses the perfstat_netinterface_total_t structure to obtain information about CPU statistics.

Example 41-14 Sample perfstat_netinterface_total_t program

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <libperfstat.h>

4 main()
5 {
6     perfstat_netinterface_total_t  ub;

7     if (perfstat_netinterface_total ((perfstat_id_t*)NULL, &ub,
sizeof(perfstat_netinterface_total_t),1) >= 0) {
8         printf("number      : %d\n", ub.number);
9         printf("ipackets   : %llu\n", ub.ipackets);
10        printf("ibytes    : %llu\n", ub.ibytes);
11        printf("ierrors   : %llu\n", ub.ierrors);
12        printf("opackets  : %llu\n", ub.opackets);
13        printf("obytes   : %llu\n", ub.obytes);
14        printf("oerrors  : %llu\n", ub.oerrors);
15        printf("collisions: %llu\n", ub.collisions);
16    }
17 }
```

On line 3 the libperfstat.h declaration file is included. Then on line 6 we declare variables for calling the perfstat_netinterface_total subroutine, which we do on line 7. Note how the usage and reference of structures is done in the call. The output from the program will look like Example 41-15 on page 804.

Example 41-15 Sample output from the perfstat_netinterface_total_t program

```
# perfstat_netinterface_total_t
number      : 3
ipackets    : 781984
ibytes      : 155461659
ierrors     : 0
opackets    : 516557
obytes      : 95930535
oerrors     : 3
collisions  : 0
```

The following is a short definition of each structure element as displayed above:

number	Interfaces count
ipackets	Packets received on interface
ibytes	Bytes received on interface
ierrors	Input errors on interface
opackets	Packets sent on interface
obytes	Bytes sent on interface
oerrors	Output errors on interface
collisions	Collisions on csma interface

Makefile for Perfstat

Example 41-16 shows a makefile for compiling the perfstat sample programs.

Example 41-16 Makefile

```
# n1 Makefile
1 CC=cc
2 CFLAGS=-g
3 PERF_LIBS=-lperfstat

4 PERF_PROGRAMS = perfstat_cpu_t perfstat_cpu_total_t perfstat_disk_t
perfstat_disk_total_t perfstat_memory_total_t perfstat_netinterface_t
perfstat_netinterface_total_t perfstat_dump_all perfstat_dude

5 all:    $(PERF_PROGRAMS)

6 $(PERF_PROGRAMS):    $$@.c
7    $(CC) $(CFLAGS) $(LIBS) $(PERF_LIBS) $? -o $@
```

Lines 1-3 are variable declarations that make changing compile parameters easier. Line 4 declares a variable for the programs (PERF_PROGRAMS). Line 6 declares that all of the programs that are targets (declared on line 4) will have a source that they depend on (appended .c to each target). Line 7 is the compile statement itself; if the program perfstat_dump_all was the target (and the

source file was changed since the last created target), then the line would be parsed to look like the following:

```
cc -g -lperfstat perfstat_dump_all.c -o perfstat_dump_all
```

Line 5 declares a target named `all` that, if we had other target:source lines with compile statements, would include them as sources on this line as well. Because this line is the first non-declarative line in the `Makefile`, just typing `make` in the same directory would evaluate it, thus compiling everything that has changed sources since the last time they were compiled.

To use the makefile, just run the `make` command.

Additional Perfstat API subroutines

The following are new additional Perfstat API subroutines to AIX 5.2. Refer to “Perfstat API programming” from *AIX 5L Version 5.2 Performance Tool Guide and Reference* for examples of these subroutines and from the file `libperstat.h`.

perfstat_diskadapter The subroutine retrieves one or more individual disk adapter usage statistics. The same function can be used to retrieve the number of available sets of adapter statistics.

perfstat_protocol The subroutine retrieves protocol usage statistics such as ICMP, ICMPv6, IP, IPv6, TCP, UDP, RPC, NFS, NFSv2, NFSv3.

perfstat_netbuffer The subroutine retrieves network buffer allocation usage statistics. The `perfstat_netbuffer` subroutine retrieves statistics about network buffer allocations for each possible buffer size.

perfstat_pagingspace This function retrieves one or more individual pagingspace usage statistics.

perfstat_reset The `perfstat_reset` subroutine flushes the information cache for the library and should be called whenever the machine configuration has changed.

perfstat_diskpath This subroutine shows the statistic of a particular disk path related to MPIO (multipath I/O) devices.

41.2 System Performance Measurement Interface

The System Performance Measurement Interface (SPMI) is an API that provides standardized access to local system resource statistics. In AIX 5L, SPMI mainly uses the `perfstat` kernel extension (`kex`) to obtain statistics. SPMI and Remote

Statistics Interface (RSi) are utilized by the Performance Toolbox and Performance Aide Products.

By developing SPMI application programs, a user can retrieve information about system performance with minimum system overhead. The SPMI API is supported on both AIX 4.3 and AIX 5L, it has more metrics than the Perfstat API and data is more refined as it provides rates and percentages for some statistics. It also enables user-created data suppliers to export data for processing by the Performance Toolbox.

The SPMI API is a collection of C programming language subroutines that execute in user space and extract data from the running kernel regarding performance statistics.

The SPMI API subroutines reside in the libSpmi.a library in the /usr/lib (or /lib because /lib is a symbolic link to /usr/lib) and is part of the perfagent.tools fileset, which is installable from the AIX base installation media and requires that the bos.perf.perfstat fileset be installed.

The /usr/include/sys/Spmidef.h file contains the subroutine declarations and type definitions of the data structures to use when calling the subroutines. This include file is part of the perfagent.server fileset.

The documentation for the subroutines can be found in the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 2*.

41.2.1 Compiling and linking

After writing a C program that uses the SPMI API and including the sys/Spmidef.h header file, you just run `cc` on it specifying that you want to link to the libSpmi.a library as follows:

```
cc -lSpmi -o spmi_program spmi_program.c
```

This will create the spmi_program file from the spmi_program.c source program, linking it with the libSpmi.a library. Then spmi_program can be run as a normal command.

41.2.2 SPMI data organization

SPMI data is organized in a multilevel hierarchy of contexts. A context may have subordinate contexts, known as sub contexts, as well as metrics. The higher-level context is called a parent context.

The following example illustrates the SPMI data hierarchy for a metric; see also “Traversing and displaying the SPMI hierarchy” on page 816:

```
CPU/cpu0/kern
```

The parents in the example above are CPU and cpu0, and the metric that can contain statistical value is kern (time executing in kernel mode).

When multiple copies of a resource are available, the SPMI uses a base context description as a template. The SPMI creates one instance of that context for each copy of the resource or system object. This process is known as *instantiation*. A context is considered instantiable if at least one of its immediate sub contexts can exist in more than one copy.

The SPMI can generate new instances of the subcontracts of instantiable contexts prior to the execution of API subroutines that traverse the data hierarchy. An application program can also request instantiation explicitly. In either case, instantiation is accomplished by requesting the instantiation for the parent context of the instances. Some instantiable contexts always generate a fixed number of sub context instances in a given system as long as the system configuration remains unchanged. Other contexts generate a fixed number of subcontracts on one system, but not on another. A final type of context is entirely dynamic in that it will add and delete instances as required during operation.

The SPMI uses a shared memory segment created from user space. When an SPMI application program starts, the SPMI checks whether another program has already set up the SPMI data structures in shared memory. If the SPMI does not find the shared memory area, it creates one and generates and initializes all data structures. If the SPMI finds the shared memory area, it bypasses the initialization process. A counter, called users, shows the number of processes currently using the SPMI.

When an application program terminates, the SPMI releases all memory allocated for the application and decrements the users counter. If the counter drops to less than 1, the entire common shared memory area is freed. Subsequent execution of an SPMI application reallocates the common shared memory area. An application program has access to the data hierarchy through the API.

Important: If you need to terminate an SPMI program, use `ki11 <PID>` without specifying a signal. This sends the SIGTERM signal to the process and it will exit properly. If for some reason this is not done, and a SIGKILL signal is sent to terminate the process and its threads, you must clean up the shared memory areas used by the application. The following steps must be done manually:

1. Make sure no other SPMI program is running.
2. Run the `ipcs` command and look for segments with segment IDs beginning with 0x78.
3. Use the `ipcrm` command with the `-m` flag to remove all segments that have a segment ID beginning with 0x78.
4. Run the `slibclean` command.

41.2.3 Subroutines

For a complete list of the SPMI API subroutines refer to the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 2*.

To create a simple monitoring program using the SPMI API, the following subroutine sequence could be used to make a snapshot of the current values for specified statistics:

Spmilnit	Initializes the SPMI for a local data consumer program.
SpmiCreateStatSet	Creates an empty set of statistics.
SpmiPathGetCx	Returns a handle to use when referencing a context.
SpmiPathAddSetStat	Adds a statistics value to a set of statistics.
SpmiGetValue	Returns a decoded value based on the type of data value extracted from the data field of an <code>SpmiStatVals</code> structure.

Before the program exits, the following subroutines should be called to clean up the used SPMI environment (allocated memory is not released until the program issues an `SpmiExit` subroutine call):

SpmiFreeStatSet	Erases a set of statistics.
SpmiExit	Terminates a dynamic data supplier (DDS) or local data consumer program's association with the SPMI, and releases allocated memory.

After setting up an SPMI environment in a monitoring application, the statistical values could be retrieved iteratively by the use of these subroutines:

SpmiFirstVals	Returns a pointer to the first <code>SpmiStatVals</code> structure belonging to a set of statistics.
----------------------	--

- SpmiGetStat** Returns a pointer to the SpmiStat structure corresponding to a specified statistic handle.
- SpmiNextVals** Returns a pointer to the next SpmiStatVals structure in a set of statistics.

Spmilnit

The Spmilnit subroutine initializes the SPMI. During SPMI initialization, a memory segment is allocated and the application program obtains basic address ability to that segment. An application program must issue the SpmiInit subroutine call before issuing any other subroutine calls to the SPMI.

```
int TimeOut;  
  
int SpmiInit (TimeOut)
```

Parameters

TimeOut

Specifies the number of seconds the SPMI waits for a Dynamic Data Supplier (DDS) program to update its shared memory segment. If a DDS program does not update its shared memory segment in the time specified, the SPMI assumes that the DDS program has terminated or disconnected from shared memory and removes all contexts and statistics added by the DDS program. The Time Out value must be either zero or greater than or equal to 15 seconds and less than or equal to 600 seconds. A value of zero overrides any other value from any other program that invokes the SPMI and disables the checking for terminated DDS programs.

SpmiCreateStatSet

The SpmiCreateStatSet subroutine creates an empty set of statistics and returns a pointer to an SpmiStatSet structure:

```
struct SpmiStatSet *SpmiCreateStatSet()
```

SpmiPathGetCx

The SpmiPathGetCx subroutine searches the context hierarchy for a given path name of a context and returns a handle to use when subsequently referencing the context:

```
char *CxPath;  
SpmiCxHdl Parent;  
  
SpmiCxHdl SpmiPathGetCx(CxPath, Parent)
```

Parameters

CxPath

Specifies the path name of the context to find. If you specify the fully qualified path name in the CxPath parameter, you must set the Parent parameter to NULL. If the path name is not qualified or is only partly qualified (that is, if it does not include the names of all contexts higher in the data hierarchy), the SpmiPathGetCx subroutine begins searching the hierarchy at the context identified by the Parent parameter. If the CxPath parameter is either NULL or an empty string, the subroutine returns a handle identifying the top context.

Parent

Specifies the anchor context that fully qualifies the CxPath parameter. If you specify a fully qualified path name in the CxPath parameter, you must set the Parent parameter to NULL.

SpmiPathAddSetStat

The SpmiPathAddSetStat subroutine adds a statistics value to a set of statistics. The SpmiStatSet structure that provides the anchor point to the set must exist before the SpmiPathAddSetStat subroutine call can succeed.

```
struct SpmiStatSet *StatSet;  
char *StatName;  
SpmiCxHdl Parent;
```

```
struct SpmiStatVals *SpmiPathAddSetStat(StatSet, StatName, Parent)
```

Parameters

StatSet

Specifies a pointer to a valid structure of type SpmiStatSet as created by the SpmiCreateStatSet subroutine call.

StatName

Specifies the name of the statistic within the context identified by the Parent parameter. If the Parent parameter is NULL, you must specify the fully qualified path name of the statistic in the StatName parameter.

Parent

Specifies either a valid SpmiCxHdl handle as obtained by another subroutine call or a NULL value.

SpmiFirstVals

The SpmiFirstVals subroutine returns a pointer to the first SpmiStatVals structure belonging to the set of statistics identified by the StatSet parameter.

```
struct SpmiStatSet *StatSet;  
struct SpmiStatVals *SpmiFirstVals(StatSet)
```


Parameters

StatSet

Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the `SpmiCreateStatSet` subroutine call.

`SpmiStatVals` structures are accessed in reverse order, so the last statistic added to the set of statistics is the first one returned. This subroutine call should only be issued after an `SpmiGetStatSet` subroutine has been issued against the `statset`.

SpmiGetValue

The `SpmiGetValue` subroutine returns a decoded value based on the type of data value extracted from the data field of an `SpmiStatVals` structure.

The `SpmiGetValue` subroutine performs the following steps:

1. Verifies that an `SpmiStatVals` structure exists in the set of statistics identified by the `StatSet` parameter.
2. Determines the format of the data field as being either `SiFloat` or `SiLong`, and extracts the data value for further processing.
3. Determines the data value as being of either type `SiQuantity` or type `SiCounter`.
4. If the data value is of type `SiQuantity`, returns the `val` field of the `SpmiStatVals` structure.
5. If the data value is of type `SiCounter`, returns the value of the `val_change` field of the `SpmiStatVals` structure divided by the elapsed number of seconds since the previous time a data value was requested for this set of statistics.

This subroutine call should only be issued after an `SpmiGetStatSet` subroutine has been issued against the `statset`.

```
struct SpmiStatSet *StatSet;  
struct SpmiStatVals *StatVal;  
  
float SpmiGetValue(StatSet, StatVal)
```

Parameters

StatSet

Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the `SpmiCreateStatSet` subroutine call.

StatVal

Specifies a pointer to a valid structure of type `SpmiStatVals` as created by the `SpmiPathAddSetStat` subroutine call, or returned by the `SpmiFirstVals` or `SpmiNextVals` subroutine calls.

SpmiNextVals

The `SpmiNextVals` subroutine returns a pointer to the next `SpmiStatVals` structure in a set of statistics, taking the structure identified by the `StatVal` parameter as the current structure. The `SpmiStatVals` structures are accessed in reverse order so the statistic added before the current one is returned. This subroutine call should only be issued after an `SpmiGetStatSet` subroutine has been issued against the statset.

```
struct SpmiStatSet *StatSet;
struct SpmiStatVals *StatVal;

struct SpmiStatVals *SpmiNextVals(StatSet, StatVal)
```

Parameters

- StatSet** Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the `SpmiCreateStatSet` subroutine call.
- StatVal** Specifies a pointer to a valid structure of type `SpmiStatVals` as created by the `SpmiPathAddSetStat` subroutine call, or returned by a previous `SpmiFirstVals` subroutine or `SpmiNextVals` subroutine call.

SpmiFreeStatSet

The `SpmiFreeStatSet` subroutine erases the set of statistics identified by the `StatSet` parameter. All `SpmiStatVals` structures chained off the `SpmiStatSet` structure are deleted before the set itself is deleted.

```
struct SpmiStatSet *StatSet;

int SpmiFreeStatSet(StatSet)
```

Parameters

- StatSet** Specifies a pointer to a valid structure of type `SpmiStatSet` as created by the `SpmiCreateStatSet` subroutine call.

SpmiExit

A successful `SpmiInit` subroutine or `SpmiDdsInit` subroutine call allocates shared memory. Therefore, a Dynamic Data Supplier (DDS) program that has issued a successful `SpmiInit` or `SpmiDdsInit` subroutine call should issue an `SpmiExit` subroutine call before the program exits the SPMI. Allocated memory is not released until the program issues an `SpmiExit` subroutine call.

```
void SpmiExit()
```

41.2.4 Examples for SPMI

In this section we show three example programs that use the SPMI API:

- ▶ “Hard-coded metrics” on page 813 uses a hard-coded array to store the hierarchical names of the metrics to collect statistics about.
- ▶ “Reading metrics from file” on page 814 reads the metrics from a file.
- ▶ “Traversing and displaying the SPMI hierarchy” on page 816 traverses the SPMI hierarchy and displays all metrics.

Hard-coded metrics

This example uses the `spmi_dude` program given in “`spmi_dude.c`” on page 949. It shows how the SPMI environment can be set up to collect and display statistics. Example 41-17 is a sample output created by the `spmi_dude` program.

Example 41-17 Sample output from the `spmi_dude` program

```
#spmi_dude 1 10
swpq  runq  pgspe  pgspi  pgout  pgin  %used  %free  fr  sr  us  sy  id  wa
  0    0   39   61    0    0    0    0    0  0  17  1  77  5
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
  0    2   39   61    0    0    0    0    0  0  50  0  49  0
  0    2   39   61    0    0    0   11    0  0  50  0  49  1
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
  0    2   39   61    0    0    0    0    0  0  50  0  49  0
  0    2   39   61    0    0    0    0    0  0  50  0  50  0
```

Table 41-2 explains the values shown in the columns in the previous output for the `spmi_dude` program.

Table 41-2 Column explanation

Column	SPMI metric	SPMI description
wa	CPU/glwait	Systemwide time waiting for I/O (percent)
id	CPU/gldle	Systemwide time CPU is idle (percent)
sy	CPU/glkern	Systemwide time executing in kernel mode (percent)
us	CPU/gluser	Systemwide time executing in user mode (percent)
fr	Mem/Virt/scan	Physical memory 4K frames examined by VMM
fr	Mem/Virt/steal	Physical memory 4K frames stolen by VMM

Column	SPMI metric	SPMI description
%free	PagSp/%totalfree	Total free disk paging space (percent)
%used	PagSp/%totalused	Total used disk paging space (percent)
pgin	Mem/Virt/pagein	4K pages read by VMM
pgout	Mem/Virt/pageout	4K pages written by VMM
pgspi	Mem/Virt/pgspgin	4K pages read from paging space by VMM
pgspo	Mem/Virt/pgspgout	4K pages written to paging space by VMM
runq	Proc/runque	Average count of processes that are waiting for the CPU
swpq	Proc/swpqe	Average count of processes waiting to be paged in

Reading metrics from file

The program in “spmi_file.c” on page 959 shows how to set up the SPMI environment to collect and display statistics after reading the SPMI metrics from a file. Example 41-18 is sample output created by the `spmi_file` program shown in the previous example.

Example 41-18 Sample output from the spmi_file program

```
# spmi_file|pr -t -2
IP/NetIF/en0/oerror      : 0      Mem/Virt/pgspgin      : 0
IP/NetIF/en0/octet_kb   : 0      Mem/Virt/pageout      : 0
IP/NetIF/en0/opacket    : 0      Mem/Virt/pagein       : 0
IP/NetIF/en0/ierror     : 0      PagSp/pgspgout       : 0
IP/NetIF/en0/ioctet_kb  : 0      PagSp/pgspgin        : 0
IP/NetIF/en0/ipacket    : 0      PagSp/%totalused     : 39
SysIO/writetech_kb      : 0      PagSp/%totalfree    : 61
SysIO/readch_kb         : 0      PagSp/totalfree      : 320219
Syscall/fork            : 0      PagSp/totalsize      : 524288
Syscall/total           : 0      Mem/Real/numclient   : 261066
Proc/ksched             : 0      Mem/Real/numlocal    : 935329
Proc/swpocc             : 88272  Mem/Real/comp         : 536802
Proc/swpqe              : 0      Mem/Real/noncomp     : 659593
Proc/runocc             : 182151 Mem/Real/numfrb       : 900748
Proc/runque             : 0      Mem/Real/%clnt       : 13
Proc/pswitch            : 0      Mem/Real/%local      : 57
Mem/Kmem/mbuf/blocks   : 0      Mem/Real/%noncomp    : 32
Mem/Kmem/mbuf/memmax    : 2309   Mem/Real/%comp       : 26
Mem/Kmem/mbuf/memuse    : 2305   Mem/Real/%pinned     : 7
Mem/Kmem/mbuf/failures  : 0      Mem/Real/%free       : 43
```

Mem/Kmem/mbuf/calls	: 0	Mem/Real/size	: 2097143
Mem/Kmem/mbuf/inuse	: 2052	CPU/gldle	: 77
Mem/Virt/steal	: 0	CPU/glwait	: 5
Mem/Virt/scan	: 0	CPU/glkern	: 1
Mem/Virt/pgspgout	: 0	CPU/gluser	: 17

The output was formatted with the **pr** command so that the columns created by the **spmi_file** program would fit on one screen. The left column shows the SPMI hierarchy name, and the value to the right of the separating colon (:) is the statistical value. The output Mem/Real/size shows the amount of real memory on the system. The value of the metric, in this case 2097143, is the number of 4 KB memory pages on the system (8GB).

Example 41-19 shows the input file used with the **spmi_file** program to create the output in Example 41-18 on page 814.

Example 41-19 Sample input file SPMI_METRICS

```

CPU/gluser
CPU/glkern
CPU/glwait
CPU/gldle
Mem/Real/size
Mem/Real/%free
Mem/Real/%pinned
Mem/Real/%comp
Mem/Real/%noncomp
Mem/Real/%local
Mem/Real/%clnt
PagSp/totalsize
PagSp/totalfree
PagSp/%totalfree
PagSp/%totalused
PagSp/pgspgin
PagSp/pgspgout
Mem/Real/size
Mem/Real/numfrb
Mem/Real/noncomp
Mem/Real/comp
Mem/Real/numlocal
Mem/Real/numclient
Mem/Virt/pagein
Mem/Virt/pageout
Mem/Virt/pgspgin
Mem/Virt/pgspgout
Mem/Virt/scan
Mem/Virt/steal
Mem/Kmem/mbuf/inuse
Mem/Kmem/mbuf/calls

```

```
Mem/Kmem/mbuf/failures
Mem/Kmem/mbuf/memuse
Mem/Kmem/mbuf/memmax
Mem/Kmem/mbuf/blocks
Proc/pswitch
Proc/runque
Proc/runocc
Proc/swpque
Proc/swpocc
Proc/ksched
Syscall/total
Syscall/fork
SysIO/readch_kb
SysIO/writetech_kb
IP/NetIF/en0/ipacket
IP/NetIF/en0/octet_kb
IP/NetIF/en0/ierror
IP/NetIF/en0/opacket
IP/NetIF/en0/octet_kb
IP/NetIF/en0/oerror
```

Traversing and displaying the SPMI hierarchy

The program in “spmi_traverse.c” on page 961 shows how to set up the SPMI environment, and then traverse and display all metrics found in the SPMI hierarchy. Example 41-20 shows sample output created by the `spmi_traverse` program.

Example 41-20 Sample output from the `spmi_traverse` program

```
CPU/gluser:Systemwide time executing in user mode (percent):Float/Quantity:0-100
CPU/glkern:Systemwide time executing in kernel mode (percent):Float/Quantity:0-100
CPU/glwait:Systemwide time waiting for IO (percent):Float/Quantity:0-100
CPU/glide:Systemwide time CPU is idle (percent):Float/Quantity:0-100
CPU/gluticks:Systemwide CPU ticks executing in user mode:Long/Counter:0-100
CPU/glticks:Systemwide CPU ticks executing in kernel mode:Long/Counter:0-100
CPU/glwticks:Systemwide CPU ticks waiting for IO:Long/Counter:0-100
CPU/gliticks:Systemwide CPU ticks while CPU is idle:Long/Counter:0-100
CPU/cpu0/user:Time executing in user mode (percent):Float/Quantity:0-100
CPU/cpu0/kern:Time executing in kernel mode (percent):Float/Quantity:0-100
CPU/cpu0/wait:Time waiting for IO (percent):Float/Quantity:0-100
CPU/cpu0/idle:Time CPU is idle (percent):Float/Quantity:0-100
CPU/cpu0/uticks:CPU ticks executing in user mode:Long/Counter:0-100
CPU/cpu0/kticks:CPU ticks executing in kernel mode:Long/Counter:0-100
CPU/cpu0/wticks:CPU ticks waiting for IO:Long/Counter:0-100
CPU/cpu0/iticks:CPU ticks while CPU is idle:Long/Counter:0-100
...(lines omitted)...
NFS/V3Svr/mknod:NFS server mknod creation requests:Long/Counter:0-200
NFS/V3Svr/remove:NFS server file removal requests:Long/Counter:0-200
```

NFS/V3Svr/rmdir:NFS server directory removal requests:Long/Counter:0-200
NFS/V3Svr/rename:NFS server file rename requests:Long/Counter:0-200
NFS/V3Svr/link:NFS server link creation requests:Long/Counter:0-200
NFS/V3Svr/readdir:NFS server read-directory requests:Long/Counter:0-200
NFS/V3Svr/readdir+:NFS server read-directory plus requests:Long/Counter:0-200
NFS/V3Svr/fsstat:NFS server file stat requests:Long/Counter:0-200
NFS/V3Svr/fsinfo:NFS server file info requests:Long/Counter:0-200
NFS/V3Svr/pathconf:NFS server path configure requests:Long/Counter:0-200
NFS/V3Svr/commit:NFS server commit requests:Long/Counter:0-200
Spmi/users:Count of common shared memory users:Long/Quantity:0-10
Spmi/statsets:Count of defined StatSets:Long/Quantity:0-50
Spmi/ddscount:Count of active dynamic data suppliers:Long/Quantity:0-10
Spmi/consumers:Count of active data consumers:Long/Quantity:0-10
Spmi/comused:kbytes of common shared memory in use:Long/Quantity:0-200
Spmi/hotsets:Count of defined HotSets:Long/Quantity:0-50

Makefile for SPMI

Example 41-21 shows what a makefile would look like for all of the programs described above.

Example 41-21 Makefile

```
# n1 Makefile
1 CC=cc
2 CFLAGS=-g
3 SPMI_LIBS=-lSpmi

4 SPMI_PROGRAMS = spmi_dude spmi_file spmi_traverse

5 all:    $(SPMI_PROGRAMS)

6 $(SPMI_PROGRAMS):    $$@.c
7    $(CC) $(CFLAGS) $(LIBS) $(SPMI_LIBS) $? -o $@
```

Lines 1-3 are variable declarations that make changing compile parameters easier. Line 4 declares a variable for the programs (SPMI_PROGRAMS). Line 6 declares that all programs that are targets (declared on line 4) will have a source that they depend on (appended .c to each target). Line 7 is the compile statement itself. If the program spmi_dude was the target (and the source file was changed since the last created target), then the line would be parsed to look like the following:

```
cc -g -lSpmi spmi_dude.c -o spmi_dude
```

Line 5 declares a target named all so that if we had other target:source lines with compile statements, they could be included as sources on this line. Because this line is the first non-declarative line in the Makefile, just typing **make** in the

same directory would evaluate it and thus compile everything that has changed sources since the last time they were compiled.

41.3 Performance Monitor API

The Performance Monitor (PM) Application Programming Interface (API) is a collection of C programming language subroutines that provide access to some of the counting facilities of the Performance Monitor features included in selected IBM microprocessors.

The Performance Monitor API and the events available on each of the supported processors are separated by design. The events available are different on each processor. However, none of the API calls depend on the availability or status of any of the events.

The Performance Monitor API includes a set of:

- ▶ System level APIs to enable counting of the activity of a whole machine, or of a set of processes with a common ancestor.
- ▶ First-party kernel thread level APIs to enable threads running in 1:1 mode to count their own activity.
- ▶ Third-party kernel thread level APIs to enable a debugger to count the activity of target threads running in 1:1 mode.

The Performance Monitor API subroutines reside in the `libpmapi.a` library in the `/usr/pmapi/lib` directory. The `libpmapi.a` library is linked to from `/usr/lib` (or `/lib` because `/lib` is a symbolic link to `/usr/lib`) and is part of the `bos.pmapi.lib` fileset, which is installable from the AIX base installation media.

The `/usr/include/pmapi.h` file contains the subroutine declarations and type definitions of the data structures to use when calling the subroutines. This include file is also part of the `bos.pmapi.lib` fileset.

Sample source code is available with the distribution, and it resides in the `/usr/samples/pmapi` directory.

The tables describing different events for different processors reside in the `/usr/pmapi/lib` directory. To extract the events available on the specific processor, use the API subroutine that extracts this information at run time. Refer to Example 41-24 on page 821.

The documentation for the subroutines can be found in the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1* and the

41.3.1 Performance Monitor data access

Hardware counters are extra logic inserted in the processor to count specific events. They are updated at every CPU cycle, and can count metrics such as the number of cycles, instructions, floating-point and fixed-point operations, loads and stores of data, and delays associated with cache. Hardware counters are non-intrusive, are very accurate, and have a low overhead, but they are specific for each processor. The metrics can be useful if you wish to determine such statistics as instructions per cycle and cache hit rates.

Performance Monitor contexts are extensions to the regular processor and thread contexts. They include one 64-bit counter per hardware counter and a set of control words. The control words define what events get counted and when counting is on or off. Because the monitor cannot count every event simultaneously, alternating the counted events can provide more data.

The thread and thread group Performance Monitor contexts are independent. This enables each thread or group of threads on a system to program themselves to be counted with their own list of events. In other words, except when using the system level API, there is no requirement that all threads count the same events.

Only events categorized as *verified* (PM_VERIFIED) have gone through full verification and can be trusted to count accurately. Events categorized as *caveat* (PM_CAVEAT) have been verified but are accurate only within the limitations documented in the event description (returned by `pm_init`). Events categorized as *unverified* (PM_UNVERIFIED) have undefined accuracy.

Note: Use caution with *unverified* events. The PM API software is essentially providing a service to read hardware registers, which may or may not have any meaningful content.

For more detailed information about the Performance Monitoring API, review the following documentation:

- ▶ *AIX 5L Version 5.2 General Programming Concepts*
- ▶ *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1*
- ▶ *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide, SG24-5155*
- ▶ <http://www.austin.ibm.com/tech/monitor.html>

41.3.2 Compiling and linking

After writing a C program that uses the PM API, and including the `pmapi.h` and `sys/types.h` header file, run `cc` on it specifying that you want to link to the `libpmapi.a` library, as in Example 41-22.

Example 41-22 Compile and link with libpmapi.a

```
# cc -lpmapi -o pmapi_program pmapi_program.c
```

This creates the `pmapi_program` file from the `pmapi_program.c` source program, linking it with the `libpmapi.a` library. Then `pmapi_program` can be run as a normal command.

Note: If you create a thread-based monitoring application (using the `threads` library), the `pthread.h` header file must be the first included file of each source file. Otherwise, the `-D_THREAD_SAFE` compilation flag should be used, or the `cc_r` compiler used. In this case, the flag is automatically set.

41.3.3 Subroutines

The following subroutines constitute the basic Performance Monitor API. Each subroutine has four additional variations for first-party kernel thread or group counting, and third-party kernel thread or group counting. These variations have the suffixes `_group`, `_mygroup`, `_mythread`, and `_thread`:

pm_init	Initializes the PM API; always called first.
pm_cycles	Measures processor speed in cycles per second.
pm_error	Decodes PM API error codes.
pm_set_program	Sets systemwide PM programming.
pm_get_program	Retrieves systemwide PM settings.
pm_delete_program	Deletes previously established systemwide PM settings.
pm_start	Starts systemwide PM counting.
pm_stop	Stops systemwide PM counting.
pm_get_data	Returns systemwide PM data.
pm_reset_data	Resets systemwide PM data.

For a detailed description of the subroutines, read the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1*.

41.3.4 Examples for PM API

A program using the PM API usually consists of three parts:

- ▶ Initialization
- ▶ Monitoring

► Reporting

Example 41-23 shows the basic layout of a program that uses the PM API.

Example 41-23 Basic layout of PM API programs

```
main ()
{
/* code that is not monitored */
    pm_init
    pm_set_program
    pm_start
/* code that is monitored */
    pm_stop
    pm_get_data
/* code that is not monitored */
    pm_delete_program
    printf(...);
}
```

The sample program in Example 41-24 traverses the available event list (read at runtime from the .evs files in /usr/pmapi/lib directory), and displays all events on the system.

Example 41-24 Sample pmapi_list.c program for displaying available events

```
1  #include <sys/types.h>
2  #include <pmapi.h>

3  main(int argc, char *argv[])
4  {
5      static pm_info_t    pminfo;
6      static pm_events_t *pmeventp;
7      static int         i,j,rc;

8      if ((rc = pm_init(PM_VERIFIED|PM_UNVERIFIED|PM_CAVEAT, &pminfo)) > 0) {
9          pm_error("pm_init", rc);
10         exit(-1);
11     }

12     for (i = 0; i < pminfo.maxpmcs; i++) {
13         pmeventp = pminfo.list_events[i];
14         for (j = 0; j < pminfo.maxevents[i]; j++, pmeventp++) {
15             printf("proc name   : %s\n",pminfo.proc_name);
16             printf("event id    : %d\n",pmeventp->event_id);
17             printf("status     : %c\n",pmeventp->status);
18             printf("threshold  : %c\n",pmeventp->threshold);
19             printf("short name  : %s\n",pmeventp->short_name);
20             printf("long name   : %s\n",pmeventp->long_name);
21             printf("description: %s\n",pmeventp->description);
```

```
22     }  
23   }  
24 }
```

Example 41-25 is the sample output from the `pmapi_list` program shown in Example 41-24 on page 821.

Example 41-25 Sample output from the sample `pmapi_list` program

...(lines omitted)...

```
proc name : POWER4  
event id  : 1  
status    : u  
threshold : g  
short name : PM_BRQ_FULL_CYC  
long name  : Cycles branch queue full  
description: The ISU sends a signal indicating that the issue queue that feeds  
the ifu br unit cannot accept any more group (queue is full of groups).  
...(lines omitted)...
```

```
proc name : POWER4  
event id  : 19  
status    : v  
threshold : g  
short name : PM_LSU0_LDF  
long name  : LSU0 executed Floating Point load instruction  
description: A floating point load was executed from LSU unit 0
```

```
proc name : POWER4  
event id  : 20  
status    : v  
threshold : g  
short name : PM_LSU1_LDF  
long name  : LSU1 executed Floating Point load instruction  
description: A floating point load was executed from LSU unit 1  
...(lines omitted).....
```

```
proc name : POWER4  
event id  : 42  
status    : v  
threshold : g  
short name : PM_L2SC_ST_REQ  
long name  : L2 slice C store requests  
description: A store request as seen at the L2 directory has been made from the  
core. Stores are counted after gathering in the L2 store queues. The event is  
provided on each of the three slices A,B, and C.
```

```
proc name : POWER4  
event id  : 43
```

```

status      : v
threshold   : g
short name  : PM_L2_PREF
long name   : L2 cache prefetches
description: A request to prefetch data into L2 was made
.....( lines mitted).....
proc name   : POWER4
event id    : 78
status      : v
threshold   : g
short name  : PM_INST_FROM_L35
long name   : Instructions fetched from L3.5
description: An instruction fetch group was fetched from the L3 of another
module. Fetch Groups can contain up to 8 instructions
.....( lines omitted).....

proc name   : POWER4
event id    : 80
status      : v
threshold   : g
short name  : PM_GRP_DISP_REJECT
long name   : Group dispatch rejected
description: A group that previously attempted dispatch was rejected.

proc name   : POWER4
event id    : 81
status      : c
threshold   : g
short name  : PM_INST_CMPL
long name   : Instructions completed
description: Number of Eligible Instructions that completed.

.... (line omitted) ....

```

The output displays events defined on POWER4™ architecture. The status field has the following values:

v	verified
u	unverified
c	caveat char

The threshold field has the following values:

y	thresholdable
g	group-only
G	thresholdable group-only

For more examples of using Performance Monitor APIs, refer to *AIX 5L Version 5.2 Performance Tools Guide and Reference*. Functional sample codes are available in the `/usr/samples/pmapi` directory.

HPM ToolKit is a Hardware Performance Monitor tool developed by IBM Research for performance measurements of applications running on IBM POWER3™ and POWER4 systems. Its implementation is based upon PM API. The toolkit can be downloaded from the following IBM site:

<http://www.alphaworks.ibm.com/tech/hpmtoolkit>

41.4 Resource Monitoring and Control

The Resource Monitoring and Control (RMC) application is part of Reliable Scalable Cluster Technology (RSCT). RMC is the strategic technology for monitoring in AIX 5L. It provides a consistent and comprehensive set of monitoring and response capabilities that can assist in detecting system resource problems. RMC can monitor many aspects of the system resources and specify a wide range of actions to be taken when a threshold or specified condition is met.

By monitoring conditions of interest and providing automated responses when these conditions occur, RSCT RMC helps maintain system availability.

RMC is included in the `rsct.core` package, which is installed automatically when AIX 5L Version 5.2 is installed. The RSCT RMC application executables reside in `/usr/sbin/rsct/bin`.

There are other RSCT packages included with AIX 5.2, but they are not installed automatically:

rsct.basic	Topology Services (HATS) and Group Services (HAGS)
rsct.compat.basic	Event Management (HAEM)
rsct.compat.clients	Event Management (HAEM)

HATS, HAGS, and HAEM are used by Parallel System Support Programs (PSSP) and High Availability Cluster Multi-Processing/Enhanced Scalability (HACMP/ES). Note that HAEM has been moved from the `rsct.basic` and `rsct.clients` packages to the `rsct.compat` package.

RMC can be used by the WebSM Graphical User Interface (GUI), and it also has command line programs that can be used to manage it. For additional information, see *Resource Monitoring and Control Guide and Reference*, SC23-4345. For the latest information, review the README documents in the `/usr/sbin/rsct/README` directory that accompany the RSCT installation media.

41.4.1 RMC commands

The following scripts, utilities, commands, and files can be used to control monitoring on a system with RMC. See the man pages or *AIX 5L Version 5.2 Commands Reference* for detailed usage information.

These are the primary RMC commands:

chrsrc	Changes the persistent attribute values of a resource or resource class.
lsactdef	Lists action definitions of a resource or resource class.
lsrsrc	Lists resources or a resource class.
lsrsrcdef	Lists a resource or resource class definition.
mkrsrc	Defines a new resource.
refrsrc	Refreshes the resources within the specified resource class.
rmrsrc	Removes a defined resource.

These are additional RMC commands:

ctsnap	Gathers configuration, log, and trace information for the RSCT product.
lsaudrec	Lists records from the audit log.
rmaudrec	Removes records from the audit log.
rmcctrl	Manages the RMC subsystem.

Additional Event Response Resource Manager commands:

chcondition	Changes any of the attributes of a defined condition.
lscondition	Lists information about one or more conditions.
mkcondition	Creates a new condition definition that can be monitored.
rmcondition	Removes a condition.
chresponse	Adds or deletes the actions of a response, or renames a response.
lsresponse	Lists information about one or more responses.
mkresponse	Creates a new response definition with one action.
rmresponse	Removes a response.
lscondresp	Lists information about a condition and its linked responses, if any.

mkcondresp	Creates a link between a condition and one or more responses.
rmcondresp	Deletes a link between a condition and one or more responses.
startcondresp	Starts monitoring a condition that has one or more linked responses.
stopcondresp	Stops monitoring a condition that has one or more linked responses.

41.4.2 Information about measurement and sampling

The RMC subsystem and its resource managers are controlled by the System Resource Controller (SRC). The basic flow in RMC for monitoring is that resource managers provide values for dynamic attributes, which are dynamic properties of resources. Resource managers obtain this information from a variety of places depending on the resource. RMC applications then register for events and specify conditions for dynamic attributes for which they want to receive events (event expression/condition). Whenever this condition is true, an event notification is returned to the application (response) and the event expression is disabled until a rearm¹ expression is true.

Comparing RMC with HAEM

Dynamic attributes are the equivalent of resource variables in Event Management. A resource manager in RMC is the equivalent of a resource monitor in HAEM (with respect to monitoring). The overhead in RMC should be about the same as in Event Management with respect to monitoring and event generation. The RMC subsystem acts as a broker between the client processes that use it and the resource manager processes that control resources.

Refer to *Event Management Programming Guide and Reference*, SA22-7354 for more information about HAEM.

Resource managers

A resource manager is a process that maps resource and resource-class abstractions into calls and commands for one or more specific types of resources. A resource manager is a stand-alone daemon. The resource manager contains definitions of all resource classes that the resource manager supports. The following resource managers are provided with the RMC fileset:

IBM.AuditRM	The Audit Log resource manager (AuditRM) provides a systemwide facility for recording information about the
--------------------	---

¹ The rearm expression is commonly the inverse of the event expression (for example, a dynamic attribute is on or off). It can also be used with the event expression to define an upper and lower boundary for a condition of interest.

system's operation, which is particularly useful for tracking subsystems running in the background.

IBM.ERRM	The Event Response resource manager (ERRM) provides the ability to take actions in response to conditions occurring on the system.
IBM.FSRM	The File System resource manager (FSRM) monitors file systems.
IBM.HostRM	The Host resource manager (HostRM) monitors resources related to an individual machine. The types of values that are provided relate to the load (processes, paging space, and memory usage) and status of the operating system. It also monitors program activity from initiation until termination.

Resource classes

A resource class definition includes a description of all attributes, actions, and other characteristics of a resource class. The currently supported resource classes are:

- ▶ IBM.Association
- ▶ IBM.ATMDevice
- ▶ IBM.AuditLog
- ▶ IBM.AuditLogTemplate
- ▶ IBM.Condition
- ▶ IBM.EthernetDevice
- ▶ IBM.EventResponse
- ▶ IBM.FDDIDevice
- ▶ IBM.FileSystem
- ▶ IBM.Host
- ▶ IBM.PagingDevice
- ▶ IBM.PhysicalVolume
- ▶ IBM.Processor
- ▶ IBM.Program
- ▶ IBM.TokenRingDevice
- ▶ IBM.Sfp
- ▶ IBM.ServiceEvent
- ▶ IBM.ManagementServer
- ▶ IBM.NetworkInterface
- ▶ IBM.HostPublic
- ▶ IBM.DRM
- ▶ IBM.WLM

The resource class IBM.Host defines a number of dynamic attributes containing kernel statistics. There are more kernel stats available than what are currently

defined as dynamic attributes. The IBM.Program resource class enables an application to obtain events related to running programs, such as process death and rebirth. To find out more about the definition of a class, see “Examining resource classes and resources” on page 829.

41.4.3 Examples for RMC

In this section we will show how to use the RMC facilities. (RMC facilities can be managed through the WebSM GUI as well.) We show the command line interface because it is used for most other performance monitoring and tuning tools, and the use of the GUI is explained well in the *AIX 5L Differences Guide Version 5.2 Edition*, SG24-5765.. The ordered way to start using monitoring with RMC is to:

1. Know what threshold/resource to monitor.
2. Determine what action to be performed when the event occurs.
3. Create a script that will perform the desired action.
4. Create an RMC condition that meets the monitoring requirements.
5. Create an RMC response for the action script(s).
6. Create an RMC association between the defined RMC condition and RMC response.
7. Activate monitoring for the condition.

Verifying that the RMC is active

To verify that the RMC resource managers are active, run the `lssrc` command as shown in Example 41-26.

Example 41-26 Using lssrc

```
# lssrc -g rsct
Subsystem      Group          PID           Status
ctrmc          rsct           18330        active
ctcas          rsct           22188        active
# lssrc -g rsct_rm
Subsystem      Group          PID           Status
IBM.ERRM      rsct_rm       23736        active
IBM.CSMagentRM rsct_rm       22966        active
IBM.ServiceRM rsct_rm       21428        active
IBM.AuditRM   rsct_rm       19102        active
IBM.HostRM    rsct_rm       19380        active
IBM.DRM       rsct_rm       24004        active
```

The output shows that RMC (ctrmc) is active as well as the default resource managers (IBM.ERRM, IBM.AuditRM, and IBM.HostRM).

Normally the `ctrmc` subsystem will be started by `init` because the installation procedure will create the following entry in `/etc/inittab`:

```
ctrmc:2:once:/usr/bin/startsrc -s ctrmc > /dev/console 2>&1
```

The RMC command `rmcctr1` controls the operation of the RMC subsystem and the RSCT resource managers. It is not normally run from the command line, but it can be used in some diagnostic environments. For example, it can be used to add, start, stop, or delete an RMC subsystem.

Examining resource classes and resources

Use the `lsrsrc` command to list the persistent and dynamic attributes and their values of either a resource class or a resource. By using `lsrsrc` without any flags, it will show all classes, as in Example 41-27.

Example 41-27 Using lsrsrc

```
# lsrsrc
class_name
"IBM.Association"
"IBM.ATMDevice"
"IBM.AuditLog"
"IBM.AuditLogTemplate"
"IBM.Condition"
"IBM.EthernetDevice"
"IBM.EventResponse"
"IBM.FDDIDevice"
"IBM.Host"
"IBM.FileSystem"
"IBM.PagingDevice"
"IBM.PhysicalVolume"
"IBM.Processor"
"IBM.Program"
"IBM.TokenRingDevice"
"IBM.Sfp"
"IBM.ServiceEvent"
"IBM.ManagementServer"
"IBM.NetworkInterface"
"IBM.HostPublic"
"IBM.DRM"
"IBM.WLM"
```

Now we can examine each of these classes in more detail. When we use the `-ap` (default) flags to the `lsrsrc` command, it will only show the persistent attributes defined for the specified class. In Example 41-28 on page 830, we used `IBM.Host`.

Example 41-28 Using lsrsrc with the -ap flags

```
# lsrsrc -ap IBM.Host
Resource Persistent Attributes for: IBM.Host
resource 1:
    Name                = "lpar05"
    nodeNameList        = {"lpar05"}
    NumProcessors       = 16
    RealMemSize         = 8589897728
    OSName              = "AIX"
    KernelVersion       = "5.2"
    DistributionName    = "IBM"
    DistributionVersion = "5.2"
    Architecture        = "ppc"
```

To look at dynamic attributes, use the `-ad` flags with the `lsrsrc` command, as in Example 41-29. Note that we get the current value of the attribute as well².

Example 41-29 Using lsrsrc with the -ad flags

```
# lsrsrc -ad IBM.Host
Resource Dynamic Attributes for: IBM.Host
resource 1:
    ActiveMgtScopes     = 1
    UpTime              = 535139
    NumUsers            = 8
    LoadAverage        = {490103,473272,470732}
    PctRealMemActive    = 70
    VMActivePageCount   = 1469482
    KMemSizeOther       = 151200
    KMemSizeStreams     = 11776
    KMemSizeMblk        = 65920
    KMemSizeOtherIP     = 4128
    KMemSizeProtcb      = 320
    KMemSizeSock        = 2144
    KMemSizeMbuf        = 2360320
    KMemNumOther        = 24
    KMemNumStreams      = 148
    KMemNumMblk         = 115
    KMemNumOtherIP      = 35
    KMemNumProtcb       = 2
    KMemNumSock         = 6
    KMemNumMbuf         = 2052
    KMemFailOtherRate   = 0
    KMemFailStreamsRate = 0
    KMemFailMblkRate    = 0
    KMemFailOtherIPRate = 0
```

² Because some of the dynamic attributes are rates, which require two values obtained over a time interval, it takes a few seconds to execute the `lsrsrc` command.

```

KMemFailProtcbRate = 0
KMemFailSockRate = 0
KMemFailMbufRate = 0
KMemReqOtherRate = 0
KMemReqStreamsRate = 0
KMemReqMblkRate = 0
KMemReqOtherIPRate = 0
KMemReqProtcbRate = 0
KMemReqSockRate = 0
KMemReqMbufRate = 0
VMPgSpOutRate = 0
VMPgSpInRate = 1
VMPgFaultRate = 3
VMPgOutRate = 0
VMPgInRate = 1
RealMemFramesFree = 430009
PctRealMemPinned = 6
PctRealMemFree = 20
PctTotalTimeKernel = 0
PctTotalTimeUser = 25.2053388090349
PctTotalTimeWait = 0
PctTotalTimeIdle = 74.7946611909651
PctTotalPgSpFree = 56.6489219665527
PctTotalPgSpUsed = 43.3510780334473
TotalPgSpFree = 594007
TotalPgSpSize = 1048576
ProcSwapQueue = 4.37159006295268
ProcRunQueue = 2.67393331721446

```

Some classes have a different layout. To look at how the class is structured, use the `lsrsrdef` command as in Example 41-30 with the `IBM.PhysicalVolume` class.

Example 41-30 Using lsrsrdef

```

# lsrsrdef IBM.PhysicalVolume
Resource Persistent Attribute Definitions for: IBM.PhysicalVolume
attribute 1:
  program_name = "Name"
  display_name = ""
  group_name = ""
  properties = {"public","read_only","selectable","reqd_for_define"}
  description = ""
  attribute_id = 0
  group_id = 0
  data_type = "char_ptr"
  variety_list = {[1,1]}
  variety_count = 1
  default_value = ""

```

```

attribute 2:
    program_name = "PVID"
    display_name = ""
    group_name   = ""
    properties   = {"public","inval_for_define","read_only","selectable"}
    description  = ""
    attribute_id = 4
    group_id     = 0
    data_type    = "binary_ptr"
    variety_list = {[1,1]}
    variety_count = 1
    default_value = ""

attribute 3:
    program_name = "NodeNameList"
    display_name = ""
    group_name   = ""
    properties   = {"option_for_define","public","read_only","selectable"}
    description  = ""
    attribute_id = 2147483647
    group_id     = 0
    data_type    = "char_ptr_array"
    variety_list = {[1,1]}
    variety_count = 1
    default_value = {}

```

To examine only specified attributes (in Example 41-31, attributes 1 and 3) from the output in the previous example, we can use `lsrsrc` to show only what is defined for the `Value` and `PVID` attributes from `IBM.PhysicalVolume`.

Example 41-31 Using `lsrsrc` with the `-x dAb` flags

```

# lsrsrc -xdAb IBM.PhysicalVolume Name PVID
"hdisk1":"0x0021768a 0x4fe05e1f 0x00000000 0x00000000":
"hdisk0":"0x0021768a 0xabd8785a 0x00000000 0x00000000":
"hdisk7":"0x0021768a 0xca813afd 0x00000000 0x00000000":
"hdisk6":"0x00071542 0xe0f1cc17 0x00000000 0x00000000":
"hdisk5":"0x00071542 0xe0f18309 0x00000000 0x00000000":
"hdisk4":"0x0021768a 0x9378cb88 0x00000000 0x00000000":
"hdisk3":"0x00000000 0x035d72e7 0x00000000 0x00000000":
"hdisk2":"0x00050592 0x247553da 0x00000000 0x00000000":

```

By using the `-x` (no header), `-d` (delimiter separated output), and `-ab` (both persistent and dynamic attributes) the `lsrsrc` command displays the disk drives and their physical volume ID in our system. A similar output can be shown by using the `-t` flag as is in Example 41-32 on page 833, or the `-xab` flags in combination with `-t`.

Example 41-32 Using lsrsrc with the -t flag

```
# lsrsrc -t IBM.PhysicalVolume Name PVID
Resource Persistent Attributes for: IBM.PhysicalVolume
Name      PVID
"hdisk1"  "0x0021768a 0x4fe05e1f 0x00000000 0x00000000"
"hdisk0"  "0x0021768a 0xabd8785a 0x00000000 0x00000000"
"hdisk7"  "0x0021768a 0xca813afd 0x00000000 0x00000000"
"hdisk6"  "0x00071542 0xe0f1cc17 0x00000000 0x00000000"
"hdisk5"  "0x00071542 0xe0f18309 0x00000000 0x00000000"
"hdisk4"  "0x0021768a 0x9378cb88 0x00000000 0x00000000"
"hdisk3"  "0x00000000 0x035d72e7 0x00000000 0x00000000"
"hdisk2"  "0x00050592 0x247553da 0x00000000 0x00000000"
```

Writing an event response script

An event response script will have the following environment variables set when it is started by RMC:

- ERRM_COND_HANDLE** The condition resource handle that caused the event, represented as a string of six hexadecimal integers that are separated by spaces.
- ERRM_COND_NAME** The name of the condition resource that caused the event. It is enclosed within double quotation marks.
- ERRM_COND_SEVERITY** The significance of the Condition resource that caused the event. For the severity attribute values of 0, 1, and 2, this environment variable has the following values; informational, warning, and critical. All other Condition resource severity attribute values are represented in this environment variable as a decimal string.
- ERRM_COND_SEVERITYID** The significance of the Condition resource that caused the event. For the severity attribute values of 0, 1, and 2, this environment variable has the following values: informational, warning, and critical. All other Condition resource severity attribute values are represented in this environment variable as a decimal string.
- ERRM_ER_HANDLE** The event response resource handle for this event. It is represented as a string of six hexadecimal integers that are separated by spaces.
- ERRM_ER_NAME** The name of the event response resource that is executing this command. It is enclosed within double quotation marks.

ERRM_RSRC_HANDLE	The resource handle of the resource whose state change caused the generation of this event. It is represented as a string of six hexadecimal integers that are separated by spaces.
ERRM_RSRC_NAME	The name of the resource whose dynamic attribute changed to cause this event. It is enclosed within double quotation marks.
ERRM_RSRC_CLASS_NAME	The name of the resource class of the dynamic attribute that caused the event to occur. It is enclosed within double quotation marks.
ERRM_RSRC_CLASS_PNAME	The name of the resource class of the dynamic attribute (enclosed within double quotation marks) that caused the event to occur; set to the programmatic name of the class that caused the event to occur.
ERRM_TIME	The time the event occurred written as a decimal string that represents the time since midnight January 1, 1970, in seconds, followed by a comma and the number of microseconds.
ERRM_TYPE	The type of event that occurred. The two possible values for this environment variable are event and rearm event.
ERRM_TYPEID	The type of event that occurred. The two possible values for this environment variable are event and rearm event.
ERRM_EXPR	The expression that was evaluated that caused the generation of this event. This could be either the event or rearm expression, depending on the type of event that occurred. This can be determined by the value of ERRM_TYPE.
ERRM_ATTR_NAME	The programmatic name of the dynamic attribute used in the expression that caused this event to occur. A variable name is restricted to include only 7-bit ASCII characters that are alphanumeric (a-z, A-Z, 0-9) and the underscore character (_). The name must begin with an alphabetic character.
ERRM_ATTR_PNAME	The programmatic name of the dynamic attribute used in the expression that caused this event to occur. A variable name is restricted to include only

7-bit ASCII characters that are alphanumeric (a-z, A-Z, 0-9) and the underscore character (_). The name must begin with an alphabetic character.

ERRM_DATA_TYPE	RMC ct_data_type_t of the dynamic attribute that changed to cause this event.
ERRM_VALUE	The value of the dynamic attribute that caused the event to occur for all dynamic attributes except those with a data type of CT_NONE.
ERRM_SD_DATA_TYPES	The data type for each element within the structured data (SD) variable separated by commas. This environment variable is only defined when ERRM_DATA_TYPE is CT_SD_PTR.

The ERRM_TIME is a string with the current time in seconds. This must be converted into the current time in a more readable format. Example 41-33 uses `perl` for the conversion.

Example 41-33 Using perl to convert ERRM_TIME

```
perl -e 'use POSIX qw(strftime);print strftime("%Y-%m-%d
iT",localtime('${ERRM_TIME%,*}') );'
```

The basic script in Example 41-34 is an example of how to send an e-mail to the `root` user when a condition occurs that triggers the activation of the event response script.

Example 41-34 Example event response script

```
#!/bin/ksh
_message () {
cat <<-EOF | tee -a /tmp/debug.out
    TIME OF EVENT : $EVENTTIME
    CONDITION     : $ERRM_COND_NAME
    SERVERITY     : $ERRM_COND_SEVERITY
    EVENT TYPE    : $ERRM_TYPE
    EXPRESSION    : $ERRM_EXPR
    RESOURCE NAME : $ERRM_RSRC_NAME
    RESOURCE CLASS: $ERRM_RSRC_CLASS_NAME
    DATA TYPE    : $ERRM_DATA_TYPE
    DATA VALUE   : $ERRM_VALUE
EOF
}
EVENTTIME=$(perl -e 'use POSIX qw(strftime);print strftime("%Y-%m-%d
iT",localtime('${ERRM_TIME%,*}') );')
_message | mail -s "RSCT: ERRM_COND_NAME $ERRM_COND_SEVERITY" root
```

Note: The output is also appended to a debug file in /tmp named debug.out. It can be helpful to use logfiles when developing event response scripts.

Creating a condition

A condition is needed for monitoring of a metric to be performed. To define a condition, use the **mkcondition** command. In Example 41-35 a condition is defined to use the IBM.FileSystem resource manager.

Example 41-35 Creating a condition with the mkcondition command

```
# mkcondition -r IBM.FileSystem -e "PercentTotUsed > 90" -E "PercentTotUsed < 85" -d "Generate event when /home > 90% full" -D "Restart monitoring /home again after back down < 85% full" -s 'Name=="/home"' "_EVENT 12345"
```

This example creates a condition that monitors the /home filesystem and, when the evaluation of PercentTotUsed > 90 is true, it generate an event named "_EVENT 12345" and the monitoring stops. When the expression PercentTotUsed < 85 becomes true, monitoring will restart. This is to prevent an event from being generated repeatedly and indefinitely.

By default, conditions generate informational events. Because we did not specify anything else, the **chcondition** command can be used to change it to a critical condition.

```
chcondition -S c "_EVENT 12345"
```

To check how the definition of the condition looks to RMC, use the **lscondition** command as in Example 41-36.

Example 41-36 Using the lscondition command

```
# lscondition "_EVENT 12345"
Displaying condition information:

condition 1:
  Name           = "_EVENT 12345"
  MonitorStatus  = "Not monitored"
  ResourceClass  = "IBM.FileSystem"
  EventExpression = "PercentTotUsed > 90"
  EventDescription = "Generate event when /home > 90% full"
  RearmExpression = "PercentTotUsed < 85"
  RearmDescription = "Restart monitoring /home again after back down < 85%
full"
  SelectionString = 'Name=="/home"'
  Severity       = "c"
  NodeNames      = {"localnode"}
```

Creating a response to a condition event

In order to perform an action when a condition is activated, a response is needed. In the following example we create a response that activates the script shown in Example 41-34 on page 835. We define our event response script to RMC:

```
mkresponse -n rsct.trapevent -s /rcm/rsct.trapevent rsct.trapevent
```

This created event response has all *stdout* discarded (we did not specify the `-o` flag), will be active only when an event occurs (`-e` flag), and will be active all days and hours in the week (we did not specify otherwise with the `-d` and `-t` flags).

To check how the definition of our response looks to RMC, we can use the **lsresponse** command as in Example 41-37.

Example 41-37 Using the lsresponse command

```
# lsresponse rsct.trapevent
Displaying response information:

ResponseName   = "rsct.trapevent"
Action         = "rsct.trapevent"
DaysOfWeek     = 1-7
TimeOfDay      = 0000-2400
ActionScript   = "/rcm/rsct.trapevent"
ReturnCode     = 0
CheckReturnCode = "n"
EventType      = "a"
StandardOut    = "n"
```

Associating response with condition

To associate an event condition, such as our condition "`_EVENT 12345`", with an event response, such as our response "`rsct.trapevent`", we use the **mkcondresp** command:

```
mkcondresp "_EVENT 12345" "rsct.trapevent"
```

To check how the definition of our condition/response connection looks to RMC, we can use the **lscondresp** command as in Example 41-38.

Example 41-38 Using the lscondresp command

```
# lscondresp _EVENT
Displaying condition with response information:

condition-response link 1:
Condition = "_EVENT 12345"
Response  = "rsct.trapevent"
State     = "Not active"
```

Note that we only used the first part of the condition name (`_EVENT`). It is also possible to use *wildcards* with a similar syntax to the `grep` command. Example 41-39 illustrates how to use wildcards with the `lscondresp` command.

Example 41-39 Using the lscondresp command with wildcards

```
# lscondresp "_EVENT.*6"
Displaying condition with response information:

condition-response link 1:
    Condition = "_EVENT 12346"
    Response  = "rsct.trapevent2"
    State     = "Active"
```

If we were to leave out the search expression for the `lscondresp` command, we would get a line view of all the condition/response connections that are defined on the system as is shown in Example 41-40. Because we prefixed our condition name with an underscore (`_`), it will show up at the top of all listings.

Example 41-40 Using the lscondresp command

```
# lscondresp
Displaying condition with response information:
Condition      Response      State
"_EVENT 12345" "rsct.trapevent" "Not active"
...(lines omitted)...
```

The output above shows the output from the `lscondresp` command in the two previous examples. The condition/response is not active ("**Not active**").

Activating monitoring of a condition

To activate monitoring of a condition, we use the `startcondresp` command. For our condition "`_EVENT 12345`" it would be done as follows:

```
startcondresp "_EVENT 12345"
```

After running the `startcondresp` command, the "`_EVENT 12345`" condition with the "`rsct.trapevent`" response will be monitored (Active) as is shown in the sample output in Example 41-41.

Example 41-41 Using the lscondresp command

```
# lscondresp
Displaying condition with response information:
Condition      Response      State
"_EVENT 12345" "rsct.trapevent" "Active"
```

When we check the condition again with the `lscondition` command it will look something like the output in Example 41-42 and indicate that the condition is now "Monitored".

Example 41-42 Using the `lscondition` command

```
# lscondition _EVENT
Displaying condition information:

condition 1:
  Name           = "_EVENT 12345"
  MonitorStatus  = "Monitored"
  ResourceClass  = "IBM.FileSystem"
  EventExpression = "PercentTotUsed > 90"
  EventDescription = "Generate event when /home > 90% full"
  RearmExpression = "PercentTotUsed < 85"
  RearmDescription = "Restart monitoring /home again after back down < 85%
full"
  SelectionString = 'Name=="/home"'
  Severity        = "c"
  NodeNames       = {"localnode"}
```

The `startcondresp` command can also be used to create a condition-response association, such as our condition "_EVENT 12345", with a event response, such as our response "rsct.trapevent" as the following example shows:

```
startcondresp "_EVENT 12345" "rsct.trapevent"
```

Note, however, that this both creates a condition-response association and activates it, as the `lscondresp` command as Example 41-43 shows. (Refer to "Associating response with condition" on page 837.)

Example 41-43 Using the `startcondresp` and `lscondresp` commands

```
# startcondresp "_EVENT 12345" "rsct.trapevent"

# lscondresp _EVENT
Displaying condition with response information:

condition-response link 1:
  Condition = "_EVENT 12345"
  Response  = "rsct.trapevent"
  State     = "Active"
```

How the condition/response event generation is done

When the event-generating expressions for the "_EVENT 12345" condition becomes true, our shell script generates an e-mail message that looks similar to the output in Example 41-44 on page 840.

Example 41-44 Sample e-mail output

```
# inc
Incorporating new mail into inbox...

8+ 04/14 To:root@wlmhost    RSCT: 2003-04-14 19:14:41 _EVENT /home >90% USED
<<TIME 0

# show 8
(Message inbox:8)
Received: (from root@localhost) by wlmhost (AIX5.2/8.11.0/8.11.0) id
f4F0Ffx22176 for root; Mon, 14 April 2003 19:15:41 -0500
Date: Mon, 14 April 2003 19:15:41 -0500
From: root
Message-Id: <200105150015.f4F0Ffx22176@wlmhost>
To: root
Subject: RSCT: 2003-04-14 19:14:41 _EVENT 12345
```

TIME OF EVENT : 2003-04-14 19:14:41

CONDITION : _EVENT 12345
SERVERITY : Informational
EVENT TYPE : Event
EXPRESSION : PercentTotUsed > 90

RESOURCE NAME : /home
RESOURCE CLASS: File System
DATA TYPE : CT_INT32
DATA VALUE : 77

We used the Mail Handler (MH) commands **inc** and **show**; this e-mail is the current one (8+). Because our event response script also appended the output to a file in the /tmp directory named debug.out, Example 41-45 shows how the same event would look in the file.

Example 41-45 Using tail -f to track the /tmp/debug.out file

```
# tail -f /tmp/debug.out
TIME OF EVENT : 2001-05-14 19:14:41

CONDITION      : _EVENT /home >90% USED
SERVERITY      : Informational
EVENT TYPE     : Event
EXPRESSION     : PercentTotUsed > 90
RESOURCE NAME  : /home
RESOURCE CLASS: File System
DATA TYPE     : CT_INT32
DATA VALUE    : 77
```

Stop monitoring a condition

To stop monitoring a condition, use the **stopcondresp** command (here applied to our sample condition/response monitoring event for the /home filesystem):

```
stopcondresp "_EVENT 12345"
```

To verify that the monitoring has stopped, use the **lsccondresp** command as in Example 41-46.

Example 41-46 Using the lsccondresp command

```
# lsccondresp "_EVENT 12345"
Displaying condition with response information:
Condition      Response      State
"_EVENT 12345" "rsct.trapevent" "Not active"
```

Removing a response definition

To remove a response definition, you must first remove any condition-response associations for the response definition. This can be accomplished by using the **-f** flag with the **rmresponse** command:

```
rmresponse -f rsct.trapevent
```

To perform the same operation in steps, first disassociate the response from the condition (in our example, between the "_EVENT 12345" condition and "rsct.trapevent" response) as shown below:

```
rmcondresp "_EVENT 12345" "rsct.trapevent"
```

After this is done, the response definition can be removed:

```
rmresponse rsct.trapevent
```

Removing a condition

To remove a condition, it is first necessary to remove any condition-response associations for the condition. This can be accomplished by using the **-f** flag with the **rmcondition** command as shown below:

```
rmcondition -f "_EVENT 12345"
```

To perform the same operation in steps, first disassociate the response from the condition (in our example, between the "_EVENT 12345" condition and "rsct.trapevent" response):

```
rmcondresp "_EVENT 12345" "rsct.trapevent"
```

After this is done the condition can be removed:

```
rmcondition "_EVENT 12345"
```

41.5 Miscellaneous performance monitoring subroutines

In this section we describe the use of some subroutines that are available to programmers from different libraries. The documentation for the subroutines can be found in the *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1 & 2*.

41.5.1 Compiling and linking

Many of the subroutines described in this section require different libraries to be linked with the program. For each subroutine that requires a specific library this is mentioned. The general syntax for compiling and linking is:

```
cc -lLIBRARY -o program program.c
```

This creates the program executable file from the program.c source program, linking it with the libLIBRARY.a library. Then **program** can be run as a normal command.

41.5.2 Subroutines

The following subroutines can be used to obtain statistical metrics:

sys_parm	Provides a service for examining or setting kernel run-time tunable parameters.
vmgetinfo	Retrieves Virtual Memory Manager (VMM) information.
swapqry	Returns paging device status.
rstat	Gets performance data from remote kernels.
getprocs	Gets process table entries.
wlm_get_info	Reads the characteristics of superclasses or subclasses.
wlm_get_bio_stats	Reads the WLM disk I/O statistics per class or per device.

sys_parm

The `sys_parm` subroutine is used to query and/or customize run-time operating system parameters. This is a replacement service for `sysconfig` with respect to querying or changing information in the `var` structure.

Syntax

```
int cmd;  
int parmflag;  
struct vario *parmp;  
  
int sys_parm ( cmd, parmflag, parmp)
```


Parameters

cmd	Specifies the SYSP_GET or SYSP_SET function.
parmflag	Specifies the parameter upon which the function will act.
parmp	Points to the user-specified structure from which or to which the system parameter value is copied. parmp points to a structure of type vario as defined in var.h.

Library

libc.a

Examples

The code in Example 41-47 uses the vario structure to obtain information about the run-time operating system parameters.

Example 41-47 Using sys_param

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/var.h>
sys_param_()
{
    struct vario    vario;

    if (!sys_parm(SYSP_GET,SYSP_V_BUFHW,&vario))
        printf("v_bufhw (buffer pool high-water mark)           : %lld\n",vario.v.v_bufhw.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MBUFHW,&vario))
        printf("v_mbufhw (max. mbufs high water mark)         : %lld\n", vario.v.v_mbufhw.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MAXUP,&vario))
        printf("v_maxup (max. # of user processes)                 : %lld\n", vario.v.v_maxup.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MAXPOUT,&vario))
        printf("v_maxpout (# of file pageouts at which waiting occurs): %lld\n", vario.v.v_maxpout.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MINPOUT,&vario))
        printf("v_minpout (# of file pageout at which ready occurs)      : %lld\n", vario.v.v_minpout.value);
    if (!sys_parm(SYSP_GET,SYSP_V_IOSTRUN,&vario))
        printf("v_iostrun (enable disk i/o history)                       : %d\n", vario.v.v_iostrun.value);
    if (!sys_parm(SYSP_GET,SYSP_V_LEASTPRIV,&vario))
        printf("v_leastpriv (least privilege enablement)                  : %d\n", vario.v.v_leastpriv.value);
    if (!sys_parm(SYSP_GET,SYSP_V_AUTOST,&vario))
        printf("v_autost (automatic boot after halt)                      : %d\n", vario.v.v_autost.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MEMSCRUB,&vario))
        printf("v_memscrub (memory scrubbing enabled)                     : %d\n", vario.v.v_memscrub.value);
    if (!sys_parm(SYSP_GET,SYSP_V_LOCK,&vario))
        printf("v_lock (# entries in record lock table)                   : %lld\n", vario.v.v_lock.value);
    if (!sys_parm(SYSP_GET,SYSP_V_FILE,&vario))
        printf("v_file (# entries in open file table)                     : %lld\n", vario.v.v_file.value);
    if (!sys_parm(SYSP_GET,SYSP_V_PROC,&vario))
        printf("v_proc (max # of system processes)                       : %lld\n", vario.v.v_proc.value);
    if (!sys_parm(SYSP_GET,SYSP_VE_PROC,&vario))
        printf("ve_proc (process table high water mark (64 Krnl))        : %llu\n", vario.v.ve_proc.value);
    if (!sys_parm(SYSP_GET,SYSP_V_CLIST,&vario))
        printf("v_clist (# of cblocks in cblock array)                   : %lld\n", vario.v.v_clist.value);
}
```

```

if (!sys_parm(SYSP_GET,SYSP_V_THREAD,&vario))
    printf("v_thread (max # of system threads)           : %lld\n", vario.v.v_thread.value);
if (!sys_parm(SYSP_GET,SYSP_VE_THREAD,&vario))
    printf("ve_thread (thread table high water mark (64 Krnl)) : %llu\n", vario.v.ve_thread.value);
if (!sys_parm(SYSP_GET,SYSP_VB_PROC,&vario))
    printf("vb_proc (beginning of process table (64 Krnl))   : %llu\n", vario.v.vb_proc.value);
if (!sys_parm(SYSP_GET,SYSP_VB_THREAD,&vario))
    printf("vb_thread (beginning of thread table (64 Krnl))   : %llu\n", vario.v.vb_thread.value);
if (!sys_parm(SYSP_GET,SYSP_V_NCPUS,&vario))
    printf("v_ncpus (number of active CPUs)                       : %d\n", vario.v.v_ncpus.value);
if (!sys_parm(SYSP_GET,SYSP_V_NCPUS_CFG,&vario))
    printf("v_ncpus_cfg (number of processor configured)          : %d\n", vario.v.v_ncpus_cfg.value);
if (!sys_parm(SYSP_GET,SYSP_V_FULLCORE,&vario))
    printf("v_fullcore (full core enabled (true/false))           : %d\n", vario.v.v_fullcore.value);
if (!sys_parm(SYSP_GET,SYSP_V_INITLVL,&vario))
    printf("v_initlvl (init level)                                   : %s\n", vario.v.v_initlvl.value);
if (!sys_parm(SYSP_GET,SYSP_V_COREFORMAT,&vario))
    printf("v_coreformat (Core File Format (64 Krnl))               : %s\n", vario.v.v_coreformat.value);
if (!sys_parm(SYSP_GET,SYSP_V_XMGC,&vario))
    printf("v_xmgc (xmalloc garbage collect delay)                 : %d\n", vario.v.v_xmgc.value);
if (!sys_parm(SYSP_GET,SYSP_V_CPUGUARD,&vario))
    printf("v_cpuguard (CPU Guarding Mode (true/false))           : %d\n", vario.v.v_cpuguard.value);
if (!sys_parm(SYSP_GET,SYSP_V_NCARGS,&vario))
    printf("v_ncargs (length of args,env for exec())               : %d\n", vario.v.v_ncargs.value);
}
main()
{
    sys_param_();
}

```

Example 41-48 shows the output from the program above.

Example 41-48 Sample output from the sys_param subroutine program

```

v_bufhw (buffer pool high-water mark)                   : 20
v_mbufhw (max. mbufs high water mark)                   : 0
v_maxup (max. # of user processes)                     : 1000
v_maxpout (# of file pageouts at which waiting occurs) : 0
v_minpout (# of file pageout at which ready occurs)    : 0
v_iostrun (enable disk i/o history)                    : 1
v_leastpriv (least privilege enablement)               : 0
v_autost (automatic boot after halt)                   : 0
v_memscrub (memory scrubbing enabled)                  : 0
v_lock (# entries in record lock table)                 : 200
v_file (# entries in open file table)                   : 511
v_proc (max # of system processes)                     : 262144
ve_proc (process table high water mark (64 Krnl))      : 3791704576
v_clist (# of cblocks in cblock array)                  : 16384
v_thread (max # of system threads)                     : 524288
ve_thread (thread table high water mark (64 Krnl))     : 3925887872
vb_proc (beginning of process table (64 Krnl))         : 3791650816
vb_thread (beginning of thread table (64 Krnl))        : 3925868544

```

v_ncpus (number of active CPUs)	: 4
v_ncpus_cfg (number of processor configured)	: 4
v_fullcore (full core enabled (true/false))	: 0
v_initlvl (init level)	:
v_coreformat (Core File Format (64 Krnl))	:
v_xmgc (xmalloc garbage collect delay)	: 3000
v_cpuguard (CPU Guarding Mode (true/false))	: 0
v_ncargs (length of args,env for exec())	: 6

vmgetinfo

The vmgetinfo subroutine returns the current value of certain Virtual Memory Manager parameters.

Syntax

```
void *out;
int command;
int arg;
```

```
int vmgetinfo(out, command, arg)
```

Parameters

- arg** Additional parameter that depends on the command parameter.
- command** Specifies which information should be returned. The command parameter has the following valid value: VMINFO
- out** Specifies the address where VMM information should be returned.

Library

libc.a

Example

The code in Example 41-49 uses the vminfo structure to obtain information about certain VMM parameters.

Example 41-49 Using vmgetinfo

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/vminfo.h>
vmgetinfo_()
{
    struct vminfo  vminfo;

    if (!vmgetinfo(&vminfo,VMINFO,sizeof(vminfo))) {
        printf("vminfo.pgexct (count of page faults)           : %11d\n",vminfo.pgexct);
        printf("vminfo.pgrclm (count of page reclaims)           : %11d\n",vminfo.pgrclm);
        printf("vminfo.lockexct (count of lockmisse)                       : %11d\n",vminfo.lockexct);
        printf("vminfo.backtrks (count of backtracks)                       : %11d\n",vminfo.backtrks);
    }
}
```

```

printf("vminfo.pageins (count of pages paged in)           : %lld\n",vminfo.pageins);
printf("vminfo.pageouts (count of pages paged out)        : %lld\n",vminfo.pageouts);
printf("vminfo.pgspgins (count of page ins from paging space) : %lld\n",vminfo.pgspgins);
printf("vminfo.pgspgouts (count of page outs from paging space) : %lld\n",vminfo.pgspgouts);
printf("vminfo.numsios (count of start I/Os)              : %lld\n",vminfo.numsios);
printf("vminfo.numiodone (count of iodones)               : %lld\n",vminfo.numiodone);
printf("vminfo.zerofills (count of zero filled pages)     : %lld\n",vminfo.zerofills);
printf("vminfo.exfills (count of exec filled pages)       : %lld\n",vminfo.exfills);
printf("vminfo.scans (count of page scans by clock)       : %lld\n",vminfo.scans);
printf("vminfo.cycles (count of clock hand cycles)        : %lld\n",vminfo.cycles);
printf("vminfo.pgsteals (count of page steals)            : %lld\n",vminfo.pgsteals);
printf("vminfo.freewts (count of free frame waits)        : %lld\n",vminfo.freewts);
printf("vminfo.extendwts (count of extend XPT waits)      : %lld\n",vminfo.extendwts);
printf("vminfo.pendiowts (count of pending I/O waits)     : %lld\n",vminfo.pendiowts);
printf("vminfo.pings (count of ping-pongs: source => alias) : %lld\n",vminfo.pings);
printf("vminfo.pangs (count of ping-pongs):alias => alias) : %lld\n",vminfo.pangs);
printf("vminfo.pongs (count of ping-pongs):alias => source) : %lld\n",vminfo.pongs);
printf("vminfo.dpongs (count of ping-pongs):alias page delete) : %lld\n",vminfo.dpongs);
printf("vminfo.wpongs (count of ping-pongs):alias page writes) : %lld\n",vminfo.wpongs);
printf("vminfo.cachef (count of ping-pong cache flushes)   : %lld\n",vminfo.cachef);
printf("vminfo.cachei (count of ping-pong cache invalidates) : %lld\n",vminfo.cachei);
printf("vminfo.numfrb (number of pages on free list)       : %lld\n",vminfo.numfrb);
printf("vminfo.numclient (number of client frames)         : %lld\n",vminfo.numclient);
printf("vminfo.numcompress (no of frames in compressed segments) : %lld\n",vminfo.numcompress);
printf("vminfo.numperm (number frames non-working segments) : %lld\n",vminfo.numperm);
printf("vminfo.maxperm (max number of frames non-working)   : %lld\n",vminfo.maxperm);
printf("vminfo.memsizepgs (real memory size in 4K pages)   : %lld\n",vminfo.memsizepgs);
printf("vminfo.minperm (no fileonly page steals)           : %lld\n",vminfo.minperm);
printf("vminfo.minfree (minimun pages free list (fblru))    : %lld\n",vminfo.minfree);
printf("vminfo.maxfree (maxfree pages free list (fblru))    : %lld\n",vminfo.maxfree);
printf("vminfo.maxclient (max number of client frames)      : %lld\n",vminfo.maxclient);
printf("vminfo.rpgcnt[0] (repaging cnt)                     : %lld\n",vminfo.rpgcnt[0]);
printf("vminfo.rpgcnt[1] (repaging cnt)                     : %lld\n",vminfo.rpgcnt[1]);
printf("vminfo.numpout (number of fblru page-outs)          : %lld\n",vminfo.numpout);
printf("vminfo.numremote (number of fblru remote page-outs) : %lld\n",vminfo.numremote);
printf("vminfo.numwseguse (count of pages in use for working seg) : %lld\n",vminfo.numwseguse);
printf("vminfo.numpseguse (count of pages in use for persistent seg) : %lld\n",vminfo.numpseguse);
printf("vminfo.numclseguse (count of pages in use for client seg) : %lld\n",vminfo.numclseguse);
printf("vminfo.numwsegpin (count of pages pinned for working seg) : %lld\n",vminfo.numwsegpin);
printf("vminfo.numpsegpin (count of pages pinned for persistent seg) : %lld\n",vminfo.numpsegpin);
printf("vminfo.numclsegpin (count of pages pinned for client seg) : %lld\n",vminfo.numclsegpin);
printf("vminfo.numvpages (accessed virtual pages)           : %lld\n",vminfo.numvpages);
}
}
main()
{
vmgetinfo_();
}

```

Example 41-50 on page 847 shows sample output from the previous program.

Example 41-50 Sample output from the vmgetinfo subroutine program

```
vminfo.pgexct (count of page faults)           : 14546505012618220
vminfo.pgrclm (count of page reclaims)        : 536876590
vminfo.lockexct (count of lockmisses)         : 536876658
vminfo.backtrks (count of backtracks)         : 120109297309366
vminfo.pageins (count of pages paged in)      : 2014365968504570
vminfo.pageouts (count of pages paged out)    : 1418138608473918
vminfo.pgspgins (count of page ins from paging space) : 3805877901186
vminfo.pgspgouts (count of page outs from paging space) : 10523206752198
vminfo.numeios (count of start I/Os)         : 3372769634949130
vminfo.numiodone (count of iodones)          : 1953278648653902
vminfo.zerofills (count of zero filled pages) : 4932190655748242
vminfo.exfills (count of exec filled pages)  : 657018864015574
vminfo.scans (count of page scans by clock)   : 10112917647137050
vminfo.cycles (count of clock hand cycles)    : 77846288734
vminfo.pgsteals (count of page steals)        : 2602183782570402
vminfo.freewts (count of free frame waits)    : 877973456558566
vminfo.extendwts (count of extend XPT waits) : 536877610
vminfo.pendiwts (count of pending I/O waits)  : 731223013988974
vminfo.pings (count of ping-pongs: source => alias) : 536877746
vminfo.pangs (count of ping-pongs):alias => alias) : 536877814
vminfo.pongs (count of ping-pongs):alias => source) : 536877882
vminfo.dpongs (count of ping-pongs):alias page delete) : 536877950
vminfo.wpongs (count of ping-pongs):alias page writes) : 536878018
vminfo.cachef (count of ping-pong cache flushes) : 536878086
vminfo.cachei (count of ping-pong cache invalidates) : 536878154
vminfo.numfrb (number of pages on free list)  : 65345
vminfo.numclient (number of client frames)    : 23562
vminfo.numcompress (no of frames in compressed segments) : 0
vminfo.numperm (number frames non-working segments) : 32535
vminfo.maxperm (max number of frames non-working) : 32761
vminfo.memsizpgs (real memory size in 4K pages) : 131047
vminfo.minperm (no fileonly page steals)     : 6552
vminfo.minfree (minimun pages free list (fblru)) : 120
vminfo.maxfree (maxfree pages free list (fblru)) : 128
vminfo.maxclient (max number of client frames) : 104016
vminfo.rpgcnt[0] (repaging cnt)              : 0
vminfo.rpgcnt[1] (repaging cnt)              : 0
vminfo.numpout (number of fblru page-outs)    : 0
vminfo.numremote (number of fblru remote page-outs) : 0
vminfo.numwseguse (count of pages in use for working seg) : 33167
vminfo.numpseguse (count of pages in use for persistent seg): 8973
vminfo.numclseguse (count of pages in use for client seg) : 23562
vminfo.numwsegpin (count of pages pinned for working seg) : 14195
vminfo.numpsegpin (count of pages pinned for persistent seg): 0
vminfo.numclsegpin (count of pages pinned for client seg) : 0
vminfo.numvpages (accessed virtual pages)    : 34567
```

swapqry

The swapqry subroutine returns information to a user-designated buffer about active paging and swap devices.

Syntax

```
char *PathName;
struct pginfo *Buffer;
int swapqry (PathName, Buffer)
```

Parameters

PathName Specifies the full path name of the block device.
Buffer Points to the buffer into which the status is stored.

Library

libc.a

Example

The code in Example 41-51 uses the pginfo structure to obtain information about active paging and swap devices.

Example 41-51 Using swapqry

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/vminfo.h>
swapqry_()
{
    struct pginfo    pginfo;
    char             device[256];
    char             path[256];
    char             cmd[256];
    FILE             *file;

    bzero(cmd,sizeof(cmd));
    sprintf(cmd,"odmget -q \"value = paging\" CuAt|awk '/name/{gsub(\"\\\\\\\\\"\",\"\\\\\",$3);print
$3}'\n");
    if (file = popen(cmd,"r"))
        while (fscanf(file,"%s\n", &device)!=EOF) {
            sprintf(path,"/dev/%s", device);
            if (!swapqry(path,&pginfo)) {
                printf("pagingspace           : %s\n",path);
                printf("devno (device number)           : %u\n",pginfo.devno);
                printf("size (size in PAGESIZE blocks)   : %u\n",pginfo.size);
                printf("free (# of free PAGESIZE blocks): %u\n",pginfo.free);
                printf("iocnt (number of pending i/o's)  : %u\n",pginfo.iocnt);
            }
        }
    }
    fclose(file);
}
```

```
}
main()
{
    swapqry();
}
```

Example 41-52 shows the output from the program in Example 41-51 on page 848.

Example 41-52 Sample output from the swapqry subroutine program

```
pagingspace           : /dev/hd6
devno (device number) : 655362
size (size in PAGE_SIZE blocks) : 262144
free (# of free PAGE_SIZE blocks): 259240
iocnt (number of pending i/o's) : 0
```

rstat

The `rstat` subroutine gathers statistics from remote kernels. These statistics are available on items such as paging, swapping, and CPU utilization. It communicates with the `rstatd` service.

Syntax

```
char *host;
struct statstime *statp;

rstat (host, statp)
```

Parameters

host Specifies the name of the machine to be contacted to obtain statistics found in the `statp` parameter.

statp Contains statistics from `host`.

Library

librpcsvc.a

Example

The code in Example 41-53 uses the `statstime` structure to obtain statistics from the remote host specified in the `host` variable.

Example 41-53 Using rstat

```
#include <stdio.h>
#include <stdlib.h>
#include <rpcsvc/rstat.h>
rstat_(char *host)
{
```

```

struct statstime statstime;
if (!rstat(host, &statstime)) {
    printf("host          : %s\n", host);
    printf("cp_time[0]    : %d\n", statstime.cp_time[0]);
    printf("cp_time[1]    : %d\n", statstime.cp_time[1]);
    printf("cp_time[2]    : %d\n", statstime.cp_time[2]);
    printf("cp_time[3]    : %d\n", statstime.cp_time[3]);
    printf("dk_xfer[0]    : %d\n", statstime.dk_xfer[0]);
    printf("dk_xfer[1]    : %d\n", statstime.dk_xfer[1]);
    printf("dk_xfer[2]    : %d\n", statstime.dk_xfer[2]);
    printf("dk_xfer[3]    : %d\n", statstime.dk_xfer[3]);
    printf("v_pgpgin     : %u\n", statstime.v_pgpgin);
    printf("v_pgpgout    : %u\n", statstime.v_pgpgout);
    printf("v_pswpin     : %u\n", statstime.v_pswpin);
    printf("v_pswpout    : %u\n", statstime.v_pswpout);
    printf("v_intr       : %u\n", statstime.v_intr);
    printf("if_ipackets  : %d\n", statstime.if_ipackets);
    printf("if_ierrors   : %d\n", statstime.if_ierrors);
    printf("if_opackets  : %d\n", statstime.if_opackets);
    printf("if_oerrors   : %d\n", statstime.if_oerrors);
    printf("if_collisions: %d\n", statstime.if_collisions);
    printf("v_swtxch    : %d\n", statstime.v_swtxch);
    printf("avenrun[0]   : %d\n", statstime.avenrun[0]);
    printf("avenrun[1]   : %d\n", statstime.avenrun[1]);
    printf("avenrun[2]   : %d\n", statstime.avenrun[2]);
    printf("boottime    : %s", ctime(&statstime.boottime.tv_sec));
    printf("curtime     : %s", ctime(&statstime.curtime.tv_sec));
}
}
main()
{
    rstat_("wlmhost");
}

```

The `librpcsvc.a` library contains the `rstat` subroutine. Link this library to the `cc` command with the `-lrpcsvc` flag as follows:

```
cc -lrpcsvc -o <program> <program>.c
```

Note: This line must be enabled in `/etc/inetd.conf` for the `rstat` subroutine to work:

```
rstatd sunrpc_udp udp wait root /usr/sbin/rpc.rstatd rstatd 100001 1-3
```

Example 41-54 shows the output from running the program above.

Example 41-54 Sample output from the `rstat` subroutine program

```

host          : wlmhost
cp_time[0]    : 28498

```



```
cp_time[1] : 0
cp_time[2] : 0
cp_time[3] : 10747805
dk_xfer[0] : 24944
dk_xfer[1] : 361
dk_xfer[2] : 31
dk_xfer[3] : 31
v_pgggin   : 469012
v_pgggout  : 330709
v_pswpin   : 886
v_pswpout  : 2458
v_intr     : 44313756
if_ipackets : 436778
if_ierrors : 0
if_opackets : 240334
if_oerrors : 4
if_collisions: 0
v_swtxch   : 7168446
avenrun[0] : 3
avenrun[1] : 5
avenrun[2] : 3
boottime   : Mon Jun  4 08:01:53 2001
curtime    : Tue Jun  5 13:01:36 2001
```

getprocs

The `getprocs` subroutine returns information about processes, including process table information defined by the `procsinfo` structure, and information about the per-process file descriptors defined by the `fdsinfo` structure.

Syntax

```
struct procsinfo *ProcessBuffer;
or struct procsinfo64 *ProcessBuffer;
int ProcessSize;
struct fdsinfo *FileBuffer;
int FileSize;
pid_t *IndexPointer;
int Count;
```

```
int getprocs(ProcessBuffer,ProcessSize,FileBuffer,FileSize,IndexPointer,Count)
```

Parameters

ProcessBuffer Specifies the starting address of an array of `procsinfo`, `procsinfo64`, or `procentry64` structures to be filled in with process table entries. If a value of `NULL` is passed for this parameter, the `getprocs` subroutine scans the process table and sets return values as normal, but no process entries are retrieved.

- ProcessSize** Specifies the size of a single procsinfo, procsinfo64, or procsinfo64 structure.
- FileBuffer** Specifies the starting address of an array of fdsinfo or fdsinfo64 structures to be filled in with per-process file descriptor information. If a value of NULL is passed for this parameter, the getprocs subroutine scans the process table and sets return values as normal, but no file descriptor entries are retrieved.
- FileSize** Specifies the size of a single fdsinfo or fdsinfo64 structure.
- IndexPointer** Specifies the address of a process identifier, which indicates the required process table entry. A process identifier of zero selects the first entry in the table. The process identifier is updated to indicate the next entry to be retrieved.
- Count** Specifies the number of process table entries requested.

Library

libc.a

Example

The code in Example 41-55 uses the procsinfo structure to obtain information about processes.

Example 41-55 Using getprocs

```
#include <procsinfo.h>
#include <sys/proc.h>
getprocs_()
{
    struct procsinfo ps[8192];
    pid_t index = 0;
    int nprocs;
    int i;
    char state;

    if ((nprocs = getprocs(&ps, sizeof(struct procsinfo), NULL, 0, &index, 8192)) > 0) {
        printf("total # %-8d %3s %5s %5s %5s %5s %5s %5s %5s %5s %5s\n", nprocs,
            "cmd", "state", "pid", "ppid", "uid",
            "nice", "#thrd", "io/4k", "size",
            "%real", "io/b");
        for (i=0; i<nprocs; i++) {
            if (ps[i].pi_pid == 0) strcpy(ps[i].pi_comm, "swapper");
            if (ps[i].pi_comm[0] == '\0') strcpy(ps[i].pi_comm, "zombie");
            switch (ps[i].pi_state) {
                case SNONE: state='E'; break;
                case SIDL: state='C'; break;
                case SZOMB: state='Z'; break;
                case SSTOP: state='S'; break;
            }
        }
    }
}
```

```

        case SACTIVE:  state='A'; break;
        case SSWAP:   state='P'; break;
    }
    printf("%20s %5c %5d %5d %5d %5d %5d %5d %5d %5d\n",
        ps[i].pi_comm, state, ps[i].pi_pid, ps[i].pi_ppid, ps[i].pi_uid,
        ps[i].pi_nice, ps[i].pi_thcount, ps[i].pi_majflt, ps[i].pi_size,
        ps[i].pi_prm, ps[i].pi_ioch);
    }
}
}
main()
{
    getprocs_();
}

```

Example 41-56 shows the output from running the example program above.

Example 41-56 Sample output from the getprocs subroutine program

total #	cmd	state	pid	ppid	uid	nice	#thrd	io/4k	size	%real	io/b
65	swapper	A	0	0	0	41	1	7	3	6	0
	init	A	1	0	0	20	1	91	203	0	94344704
	wait	A	516	0	0	41	1	0	2	6	0
	wait	A	774	0	0	41	1	0	2	6	0
	wait	A	1032	0	0	41	1	0	2	6	0
	wait	A	1290	0	0	41	1	0	2	6	0
	lrud	A	1548	0	0	41	1	0	3	6	0
	xmgc	A	1806	0	0	41	1	0	4	6	0
	netm	A	2064	0	0	41	1	1	4	6	0
	gil	A	2322	0	0	41	5	0	16	6	0
	wlmsched	A	2580	0	0	41	1	0	4	6	0
	dog	A	3184	1	0	20	4	0	10	6	0
	lvmbb	A	3372	0	0	20	1	0	4	6	0
	bsh	A	4602	1	0	22	1	0	314	0	10949

...(lines omitted)...

wlm_get_info

The `wlm_get_info` subroutine is used to get the characteristics of the classes defined in the active Workload Manager (WLM) configuration, together with their current resource usage statistics.

Syntax

```

struct wlm_args *wlmargs;
struct wlm_info *info
int *count

int wlm_get_info ( wlmargs, info, count)

```

Parameters

wlmargs

The address of a struct `wlm_args` data structure. The `versflags` fields of the `wlm_args` structure must be provided and initialized with `WLM_VERSION`. Optionally, the following flag values can be or'ed to `WLM_VERSION`: `WLM_SUPER_ONLY`, `WLM_SUB_ONLY`, `WLM_VERBOSE_MODE`. `WLM_SUPER_ONLY` and `WLM_SUB_ONLY` are mutually exclusive.

name

Contains either a null string or the name of a valid superclass or subclass (in the form `Super.Sub`). This field can be used in conjunction with the flags to further narrow the scope of `wlm_get_info`.

All the other fields of the `wlm_args` structure can be left uninitialized.

info

The address of an array of structures of type `struct wlm_info`. Upon successful return from `wlm_get_info`, this array contains the WLM statistics for the classes selected.

count

The address of an integer containing the maximum number of elements (of type `wlm_info`) for `wlm_get_info` to copy into the array above. If the call to `wlm_get_info` is successful, this integer contains the number of elements actually copied. If the initial value is equal to zero (0), `wlm_get_info` sets this value to the number of classes selected by the specified combination of `versflags` and `name` above.

Library

`libwlm.a`

Example

The code in Example 41-57 uses the `wlm_info` structure to obtain information about characteristics of the active WLM classes.

Example 41-57 Using `wlm_get_info`

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wlm.h>
#include <sys/wlm.h>
wlm_get_info_(
{
    struct wlm_args wlmargs;
    struct wlm_info *wlm_info;
    int wlmcount = 0;
    int i=0;
```

```

if (!wlm_initialize(WLM_VERSION)) {
    wlmargs.versflags = WLM_VERSION;
    bzero(wlmargs.cl_def.data.descr.name, sizeof(wlmargs.cl_def.data.descr.name));
    if (!wlm_get_info(&wlmargs, NULL, &wlmcount) && wlmcount > 0) {
        wlminfo = malloc(wlmcount * sizeof(struct wlm_info));
        if (!wlm_get_info(&wlmargs, wlminfo, &wlmcount)) {
            printf("%-15s %8s %8s %8s %8s %8s %8s\n",
                "Class", "Tier", "Id", "Pri", "Inuse", "#Pages", "ChgLvl");
            for (i = 0; i < wlmcount; i++) {
                printf("%-15s %8d %8d %8d %8d %8d %8d\n",
                    wlminfo[i].i_descr.name, wlminfo[i].i_descr.tier, wlminfo[i].i_class_id,
                    wlminfo[i].i_cl_pri, wlminfo[i].i_cl_inuse, wlminfo[i].i_cl_npages,
                    wlminfo[i].i_cl_change_level);
            }
        }
    }
}
}
}
}
main()
{
    wlm_get_info();
}

```

Example 41-58 shows how the output of the example program above could look.

Example 41-58 Sample output from the wlm_get_info subroutine program

Class	Tier	Id	Pri	Inuse	#Pages	ChgLvl
Unclassified	0	0	10	1	28911	1
Unmanaged	0	16	10	1	14244	1
Default	0	32	47	3	0	2
Shared	0	48	47	0	4843	2
System	6	64	145	54	30695	2
db1	0	80	0	0	0	1
db1.Default	0	81	23	0	0	2
db1.Shared	0	82	23	0	0	2
db1.sub1	0	83	0	0	0	1
db2	0	96	47	0	0	1
...(lines omitted)...						

The libwlm.a library contains the wlm_get_info subroutine. Link this library to the cc command with the -lwlm flag as follows:

```
cc -lwlm -o <program> <program>.c
```

Note: To initialize the WLM API connection, you must use the `wlm_initialize` subroutine before other WLM subroutines can be used. This only needs to be done once per process.

wlm_get_bio_stats

The `wlm_get_bio_stats` subroutine is used to get the WLM disk I/O statistics. There are two types of statistics available:

- ▶ The statistics about disk I/O utilization per class and per devices, returned by `wlm_get_bio_stats` in `wlm_bio_class_info_t` structures
- ▶ The statistics about the disk I/O utilization per device, all classes combined, returned by `wlm_get_bio_stats` in `wlm_bio_dev_info_t` structures

Syntax

```
dev_t dev;  
void *array;  
int *count;  
char *class;  
int flags;  
int wlm_get_bio_stats ( dev, array, count, class, flags)
```

Parameters

- flags** Must be initialized with `WLM_VERSION`. Optionally, the following flag values can be or'ed to `WLM_VERSION`: `WLM_SUPER_ONLY`, `WLM_SUB_ONLY`, `WLM_BIO_CLASS_INFO`, `WLM_BIO_DEV_INFO`, `WLM_BIO_ALL_DEV`, `WLM_BIO_ALL_MINOR`, `WLM_VERBOSE_MODE`. One of the mutually exclusive flags `WLM_BIO_CLASS_INFO` or `WLM_BIO_DEV_INFO` must be specified. `WLM_SUPER_ONLY` and `WLM_SUB_ONLY` are mutually exclusive.
- dev** Device identification (major, minor) of a disk device. If `dev` is equal to 0, the statistics for all devices are returned (even if `WLM_BIO_ALL_DEV` is not specified in the flags argument).
- array** Pointer to an array of `wlm_bio_class_info_t` structures (when `WLM_BIO_CLASS_INFO` is specified in the flags argument) or an array of `wlm_bio_dev_info_t` structures (when `WLM_BIO_DEV_INFO` is specified in the flags argument). A NULL pointer can be passed together with a count of 0 to determine how many elements are in scope for the set of arguments passed.
- count** The address of an integer containing the maximum number of elements to be copied into the array above. If the call to `wlm_get_bio_stats` is successful, this integer will contain the number of elements actually copied. If the initial value is equal to 0,

wlm_get_bio_stats sets this value to the number of elements selected by the specified combination of flags and class.

class A pointer to a character string containing the name of a superclass or subclass. If class is a pointer to an empty string (""), the information for all classes is returned. The class parameter is taken into account only when the flag WLM_BIO_CLASS_INFO is set.

Library

libwlm.a

Example

The code in Example 41-59 uses the wlm_bio_dev_info_t structure to obtain information about WLM disk I/O statistics.

Example 41-59 Using wlm_get_bio_stats

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/wlm.h>
#include <sys/wlm.h>
wlm_get_bio_(
{
    dev_t          wlmdev = 0;
    struct wlm_bio_dev_info_t *wlmarray;
    int            wlmcount = 0;
    char          *wlmclass = NULL;
    int           wlmflags = WLM_VERSION|WLM_BIO_ALL_DEV;
    int           i=0;

    if (!wlm_initialize(WLM_VERSION)) {
        wlmflags |= WLM_BIO_DEV_INFO;
        if (!wlm_get_bio_stats(wlmdev,NULL,&wlmcount,wlmclass,wlmflags) && wlmcount > 0) {
            wlmarray = (struct wlm_bio_dev_info_t*)malloc(wlmcount*sizeof(struct
wlm_bio_dev_info_t));
            if (!wlm_get_bio_stats(wlmdev,(void*)wlmarray,&wlmcount,wlmclass,wlmflags)) {
                for (i = 0; i< wlmcount; i++) {
                    printf("device                : %ld\n", wlmarray[i].wbd_dev);
                    printf("active_cntrl (# of active cntrl)    : %d\n", wlmarray[i].wbd_active_cntrl);
                    printf("in_queue (# of requests in waiting queue) : %d\n", wlmarray[i].wbd_in_queue);
                    printf("max_queued (maximum # of requests in queue): %d\n", wlmarray[i].wbd_max_queued);
                    printf("last[0] (Statistics of last second)       : %d\n", wlmarray[i].wbd_last[0]);
                    printf("max[0] (Maximum of last second statistics) : %d\n", wlmarray[i].wbd_max[0]);
                    printf("av[0] (Average of last second statistics)  : %d\n", wlmarray[i].wbd_av[0]);
                    printf("total[0] (Total of last second statistics) : %d\n", wlmarray[i].wbd_total[0]);
                    printf("\n");
                }
            }
        }
    }
}
```

```

    }
}
main()
{
    wlm_get_bio_();
}

```

Example 41-60 shows what the output of the program above would look like.

Example 41-60 Sample output from the wlm_get_bio_stats subroutine program

```

device                               : 917504
active_cntrl (# of active cntrl)      : 0
in_queue (# of requests in waiting queue) : 0
max_queued (maximum # of requests in queue): 0
last[0] (Statistics of last second)   : 0
max[0] (Maximum of last second statistics) : 0
av[0] (Average of last second statistics) : 0
total[0] (Total of last second statistics) : 0

device                               : 917504
active_cntrl (# of active cntrl)      : 2
in_queue (# of requests in waiting queue) : 0
max_queued (maximum # of requests in queue): 0
last[0] (Statistics of last second)   : 0
max[0] (Maximum of last second statistics) : 72
av[0] (Average of last second statistics) : 0
total[0] (Total of last second statistics) : 0
...(lines omitted)...

```

The libwlm.a library contains the wlm_get_info subroutine. Link this library to the cc command with the -lwlm flag as follows:

```
cc -lwlm -o <program> <program>.c
```

Note: To initialize the WLM API connection, you must use the wlm_initialize subroutine before other WLM subroutines can be used. This only needs to be done once per process.

41.5.3 Combined example

The dudestat.c program in “dudestat.c” on page 965 illustrates how the different subroutines could be used together. Sample output of the dudestat program is shown in Example 41-61 on page 859.

Example 41-61 Sample output from the dudestat program

```
# dudestat root kiwi saffy fuzzy swede
PARTY ON!
```

The root dude is online and excellent!

There are 4 dudes missing!

Dude, here is some excellent info for you today

```
v_maxup (max. # of user processes)           : 1000
v_maxpout (# of file pageouts at which waiting occurs): 0
v_minpout (# of file pageout at which ready occurs) : 0
v_file (# entries in open file table)         : 511
v_proc (max # of system processes)           : 262144
freewts (count of free frame waits)          : 877973724082172
extendwts (count of extend XPT waits)        : 0
pendiowts (count of pending I/O waits)       : 740774484377600
numfrb (number of pages on free list)        : 51945
numclient (number of client frames)          : 19994
numcompress (no of frames in compressed segments) : 0
numperm (number frames non-working segments) : 32628
maxperm (max number of frames non-working)   : 32761
maxclient (max number of client frames)      : 104016
memsizepgs (real memory size in 4K pages)    : 131047
paging space device                           : /dev/hd6
size (size in PAGESIZE blocks)               : 262144
free (# of free PAGESIZE blocks)             : 259171
iocnt (number of pending i/o's)              : 0
```

Archived

Workload Manager tools

Workload Manager (WLM) is a subsystem that enables the system administrator to manage resources by using management classes and assigning processes to these classes. Each class can be designated a specific amount of CPU, physical memory, and disk I/O bandwidth by allocating shares and percentages of resources depending on its needs. This results in users of a class being able to function without the system resources being stolen by another class.

The purpose of this chapter is to discuss WLM performance tools rather than to explain the intricacies of setting up WLM. For information about configuring WLM refer to *AIX 5L Workload Manager (WLM)*, SG24-5977.

42.1 WLM tools overview

The tools discussed in this chapter are only effective when WLM is operative, and they are discussed here in that context. These tools are useful for monitoring and analyzing WLM activity. This situation may be useful for determining the effect of Workload Manager on the system. For example, statistics can be collected with Workload Manager in Passive mode or in Active mode.

Active mode In this mode, WLM monitors and regulates the CPU, memory, and disk I/O utilization of the processes in the various classes.

Passive mode In this mode, WLM only monitors the resource utilization without interfering with the standard operating system resource allocation mechanisms.

This chapter discusses the use of tools that give current real-time information, as in the case of the **wlmstat** command. Other commands capture data over a period of time, such as the **wlmmon** command that captures data over a 24-hour period. **wlmpperf** is similar to **wlmmon** but will monitor WLM statistics for up to one year. **wlmmon** and **wlmpperf** are graphical; others such as **wlmstat** are text-based. **wlmstat**, **wlmmon**, and **wlmpperf** are discussed in great detail in this chapter.

Other tools capture WLM statistics, but they are not discussed in this chapter because of their general nature, such as the **topas** command (discussed in Chapter 11, “The topas command” on page 179). The **topas** command is useful in that it shows the top hot classes as well as their associated processes in WLM in real time. In the same way, the **ps** command (Chapter 8, “The ps command” on page 127) can list WLM classes. This can be useful to determine if a process belongs to a particular WLM class; see Example 42-7 on page 869 for details. The **svmon** command can display WLM class and tier statistics. Refer to Chapter 24, “The svmon command” on page 387 for more information.

The specific purpose of each WLM tool is listed below:

wlmstat -v	Checks the WLM configuration.
wlmstat	Real-time monitoring tool
topas	Real-time monitoring tool on AIX 5L Version 5.2
wlmmon	Long-term analysis tool
wlmpperf	Long-term analysis tool

42.2 wlmstat

The **wlmstat** command displays resource utilization per class. The information displayed is typically CPU, memory, and disk I/O usage for each class. If WLM is

not running on a system and the **wlmstat** command is executed, then the following error message will be displayed:

```
1495-576 WLM is not running
```

This command is useful on systems that have Workload Manager in either active or passive mode.

The **wlmstat** command resides in `/usr/sbin` and is part of the `bos.rte.control` fileset. This fileset is installed as part of the default AIX operating system installation.

The syntax of the **wlmstat** command is:

```
wlmstat [ -l Class | -t Tier ] [ -S | -s ] [ -c | -m | -b ]  
[ -B Device ] [ -q ] [ -T ] [ -a ] [ -w ] [ -v ] [ Interval ] [ Count ]
```

Flags

- a** Gives absolute figures (relative to the total amount of the resource available to the whole system) for subclasses, with a 0.01% resolution. By default, the figures shown for subclasses are a percentage of the amount of the resource consumed by the superclass, with a one percent resolution. For instance, if a superclass has a CPU target of seven percent and the CPU percentage shown by **wlmstat** without **-a** for a subclass is five percent, **wlmstat** with **-a** shows the CPU percentage for the subclass as 0.35 percent.
- b** Displays only disk I/O statistics.
- B Device** Displays disk I/O device statistics. Passing an empty string (**-B ""**) displays the statistics for all disks accessed by the class.
- c** Shows only CPU statistics.
- l Class** Displays statistics for Class name. If not specified, all classes display along with a summary for appropriate fields.
- m** Shows only physical memory statistics.
- s** Displays only subclass statistics.
- S** Displays only superclasses statistics.
- t Tier** Displays statistics only for the specified Tier.

- T Displays the total numbers for resource utilization since WLM was started or the class was created, whichever happened last. The units are:
 - ▶ CPU: The total CPU time consumed by a class in microseconds, since that class was started
 - ▶ MEM: Unused
 - ▶ DKIO: Total number of 512-byte blocks sent/received by a class for all disk devices accessed
 - ▶ PROCESSES: Number of running processes belonging to that class
 - ▶ THREADS: Number of running threads belonging to that class
 - ▶ LOGINS: Number of user Logins belonging to that class associated with user rules belonging to that class
- v Specifies verbose mode. This flag, intended for troubleshooting, also displays some class attributes, resource shares and limits, and other WLM parameters including internal parameter values intended for AIX support personnel.
- w Displays the memory high-water mark, which is the maximum number of pages that a class had in memory at any given time since WLM was started or the class was created (whichever happened last).

Parameters

Class	The name of a collection of processes and their associated threads.
Tier	The position in the hierarchy of resources that a class is in.
Interval	The length of time in seconds between measurements.
Count	The number of iterations. If this value is omitted, then the output will be continuous.
Device	The name of a disk device to be monitored.

42.2.1 Information about measurement and sampling

The results obtained by the `wlmstat` command from the kernel structures are tabulated with the following fields:

CLASS The name of the class field.

CPU	The percentage of total CPU time consumed by the class.
MEM	The percentage of physical memory consumed by the class.
DKIO	The percentage of the disk I/O bandwidth consumed by the class. This number is the average of the disk bandwidth on all disk devices accessed by the class and usually is not very significant. For instance, if a class consumes 80 percent of the bandwidth of one disk and 5 percent of the bandwidth of two other disks, the DKIO column will show 30 percent. For details on per-device utilization, use the -B Device option.

The `wlm_get_info` subroutine is used to get the characteristics of the classes defined in the active WLM configuration together with their current resource usage statistics. The kernel updates the statistics once a second. The values that are displayed by the `wlmstat` command are decayed averages over the sample period.

For more information about the `wlm_get_info` subroutine, refer to *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 2*.

The sampling interval for the `wlmstat` command can be supplied when the command is issued. If an interval value is not specified on the command line, then a single output is displayed, which is a decayed average over the sample period. If the `wlmstat` command is used with an interval figure, then the sampling interval will be that supplied value in seconds. The number of iterations is determined by the value of the Count parameter on the command line. If this value is omitted, and an interval value is used on the command line, then the number of iterations is assumed to be infinite.

42.2.2 Examples for wlmstat

Example 42-1 shows the output when no parameters are supplied with the `wlmstat` command.

Example 42-1 Output of the wlmstat command without any flags or parameters

```
# wlmstat
      CLASS CPU MEM DKIO
Unclassified  0  75  0
      Unmanaged  0  12  0
      Default  0  0  0
      Shared  0  6  0
      System  0  18  0
      db1  0  0  0
db1.Default  -  -  -
db1.Shared  -  -  -
db1.sub1  -  -  -
```

db2	0	0	0
devlt	0	0	0
devlt.Default	-	-	-
devlt.Shared	-	-	-
devlt.hackers	-	-	-
devlt.hogs	-	-	-
devlt.editors	-	-	-
devlt.build	-	-	-
VPs	0	0	0
acctg	0	0	0
TOTAL	0	100	0

Using the **wlmstat** command with the **-v** flag produces the output shown in Example 42-2.

Example 42-2 wlmstat -v provides a verbose output

```
# wlmstat -vc
CLASS tr i #pr CPU sha min smx hmx des rap urap pri
Unclassified 0 0 1 0 -1 0 100 100 100 0 47 10
Unmanaged 0 0 1 0 -1 0 100 100 100 0 47 10
Default 0 0 3 0 -1 0 100 100 100 0 47 47
Default.Default 0 0 3 - 1 0 100 100 100 100 23 23
Default.Shared 0 0 0 - -1 0 100 100 0 0 46 46
Shared 0 0 0 0 -1 0 100 100 100 0 47 47
Shared.Default 0 0 0 - 1 0 100 100 100 100 23 23
Shared.Shared 0 0 0 - -1 0 100 100 0 0 46 46
System 6 0 62 0 -1 0 100 100 100 0 145 145
System.Default 0 0 62 - 1 0 100 100 100 100 121 121
System.Shared 0 0 0 - -1 0 100 100 0 0 144 144
db1 0 1 0 0 2 0 100 100 100 100 0 0
db1.Default 0 0 0 - -1 0 100 100 100 0 23 23
db1.Shared 0 0 0 - -1 0 100 100 100 0 23 23
db1.sub1 0 0 0 - -1 25 100 100 100 100 0 0
db2 0 1 0 0 -1 0 100 100 100 0 47 47
db2.Default 0 0 0 - 1 0 100 100 100 100 23 23
db2.Shared 0 0 0 - -1 0 100 100 0 0 46 46
devlt 0 0 0 0 30 0 100 100 100 100 0 0
devlt.Default 0 0 0 - -1 0 100 100 100 0 23 23
devlt.Shared 0 0 0 - -1 0 100 100 100 0 23 23
devlt.hackers 0 0 0 - -1 0 20 100 100 0 23 23
devlt.hogs 0 0 0 - -1 0 100 100 100 0 23 23
```

The output of the **wlmstat -v** command will wrap around on the screen if **-v** is used on its own. In the previous example, the **-c** flag has been added so the statistics are only for CPU. Table 42-1 lists information could be of interest to users.

Table 42-1 Output of *wlmstat -v*

Header	Description
CLASS	Class name.
tr	Tier number (0 (zero) to 9).
i	Value of the inheritance attribute: 0 (zero) = no, 1 (one) = yes.
#pr	Number of processes in the class. If a class has no process assigned to it, the values shown in the other columns may not be significant.
CPU	CPU utilization of the class expressed as a percentage.
MEM	Physical memory utilization of the class expressed as a percentage.
DKIO	Disk IO bandwidth utilization for the class expressed as a percentage.
sha	Number of shares ('-' is represented as -1).
in	Resource minimum limit expressed as a percentage.
smx	Resource soft maximum limit expressed as a percentage.
hmx	Resource hard maximum limit expressed as a percentage.
des	(desired): percentage goal (target) calculated by WLM using the shares numbers.
npg	Number of memory pages owned by the class.

The other columns are for internal use only and bear no meaning for administrators and end users. This format is better used with a resource selector (**-c**, **-m**, or **-b**). Otherwise the lines might be too long to fit into a line of a display terminal.

In Example 42-3, the **wlmstat** command is used with various flags together with the **ps** command to resolve a performance and WLM configuration problem.

Example 42-3 *wlmstat* showing unexpected distribution of resources

```

root: / =>wlmstat
      Name CPU MEM
Unclassified  1 18
      System  5  2
      Default 91  5
      DB2_user  0  0
      DB2_system 0  0

```

```
http 0 0
notes 0 0
```

In this example, the Default class unexpectedly has 91 percent of the total CPU usage. The other classes, specifically DB2_user, DB2_system, http, and notes, have no CPU usage even though processes that are expected to be assigned to these classes are running.

Example 42-4 Examining the WLM rules and classes files

```
root: /etc/wlm/standard =>cat classes
System:
Default:
DB2_user:
    description = "DB2 Clients"
DB2_system:
    description = "DB2 server"
http:
    description = "http testing"
notes:
    description = "Domino"

root: /etc/wlm/standard =>cat rules
*class    resvd    user    group    application type    tag
System    -         root    -         -         -
Default  -         -       -         -         -
DB2_system -         db2as   -         -         -
DB2_user  -         db2inst1 -       -         -
http      -         nobody  -         -         -
notes     -         notes   -         -         -
```

The problem is in the rules file. The rule for Default is in the second line. Because the WLM class assignment algorithm reads through the rules file from top to bottom, the Default class is configured prior to the notes, http, DB2_user and DB2_system classes. The class assignment algorithm goes through the rules in order and assigns the process to the class corresponding to the first rule that matches the process attributes. This results in all non-root processes being assigned to the Default class. In the ideal WLM rules file, the Default class is inserted at the bottom of this file.

Example 42-5 The WLM rules file after the change in class order

```
root: /etc/wlm/standard =>cat rules
*class    resvd    user    group    application type    tag
System    -         root    -         -         -
DB2_system -         db2as   -         -         -
DB2_user  -         db2inst1 -       -         -
http      -         nobody  -         -         -
```

notes	-	notes	-	-
Default	-	-	-	-

In this example, the `Default` class has been moved and is now the last entry in the rules file. Note that the order in the other files such as `classes`, `limits`, and `shares` is irrelevant. The rules file in this instance, where the order of the rules is incorrect, can be edited using the `vi` editor. Under normal circumstances, the `smitty wlm` command or Web-based System Manager should be used to modify the rules file. For the changes to the rules file to take effect, WLM can be updated using the `wlmcntrl -u` command.

Example 42-6 wlmstat output after the changes to the WLM rules file

```
root: / =><wlmstat>
      Name CPU MEM
Unclassified  1 17
      System  5  2
Default    0  0
      DB2_user 79  6
      DB2_system 0  0
           http  0  0
           notes 13  1
```

In Example 42-6, the `DB2_user` and `notes` classes are now registering resource usage. Note that the `http` and `DB2_system` classes still have no resource usage.

Example 42-7 The ps command output shows that http processes are running

```
root: / =>ps -e -o pid,ppid,user,class,args | grep httpd
  PID  PPID  USER          CLASS  COMMAND
   9140 159154 nobody        System /usr/HTTPServer/bin/httpd
  18414 159154 nobody        System /usr/HTTPServer/bin/httpd
  21716 159154 nobody        System /usr/HTTPServer/bin/httpd
  24316 159154 nobody        System /usr/HTTPServer/bin/httpd
  24808 159154 nobody        System /usr/HTTPServer/bin/httpd
  31626 159154 nobody        System /usr/HTTPServer/bin/httpd
  39070 159154 nobody        System /usr/HTTPServer/bin/httpd
  41582 159154 nobody        System /usr/HTTPServer/bin/httpd
...(lines omitted)...
```

There are processes running on the system that are expected to be running in the `http` class. These processes are actually running in the `System` class, as can be seen from the `ps` command output in Example 42-7. The parent process, which has PID 159154 and UID root, is classified in the `System` class. A process will remain in its parent's class if the inheritance option is enabled for that class, regardless of the classification rules. In this case, inheritance should be disabled (which is the default). Alternately, classification can be done by application name

as in Example 42-8. In this way, the parent process starts as member of the `http` class, and all child processes remain in that class.

Example 42-8 The WLM rules file changed for child processes

```

root: /etc/wlm/standard =>cat rules
*class      resvd  user   group  application type  tag
http      -      -      -      /usr/HTTPServer/bin/httpd
System      -      root   -      -
DB2_system  -      db2as  -      -
DB2_user    -      db2inst1 -    -
notes      -      notes  -      -
Default  -      -      -      -

```

The changes to the rules file with the modified entry for the `http` class can be seen in Example 42-8. When the process under the `application type` heading is executed, the child processes will run in that same class.

Example 42-9 wlmstat output

```

# wlmstat
      CLASS CPU MEM DKIO
Unmanaged  0 43  0
Default    0  0  0
Shared     0  6  0
System     3 33  0
  db1      0  0  0
db1.Default - - -
db1.Shared - - -
db1.sub1   - - -
  db2      0  0  0
  devlt    0  0  0
devlt.Default - - -
devlt.Shared - - -
devlt.hackers - - -
devlt.hogs - - -
devlt.editors - - -
devlt.build - - -
  acctg    0  0  0
  Red      29 25  2
Red.Default  0  0  0
Red.Shared  0  0  0
Red.Authors 65 48  0
Red.Editors 34 48 100
TOTAL     32 100  2

```

Example 42-9 shows the output of the `wlmstat` command. The hard maximum value of memory set up in the WLM configuration for the Red class is set to 25

percent. The `wlmstat` command reports the value of 25 percent of total memory consistently used over a period of time. This means that all of the available memory for this class is consistently used up. This is an obvious bottleneck on performance for this class. It will be necessary to analyze the nature of the work type that the Red class does because the running jobs could be of a batch nature and the priority may be low. Alternately, the Red class could be a group of users trying to process invoices in which performance may be poor and the Red class should be allocated more memory. Another problem that will occur when a class runs out of memory is that it will start to page. This paging activity will affect the entire system's performance.

Example 42-10 Global paging problem caused by Red class memory shortage

```
# vmstat 2
kthr  memory                page                faults                cpu
-----
 r  b  avm  fre  re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa
23  7 58807  957  0  21  13 384 752289  6 1052  743 560 48  7 13 31
 6  3 58810  860  0  4  24 336 626827  5 645  809 687  0 12 23 64
 8  4 58811  856  0  5  13 312 375770  3 639  756 663  0  9 30 61
 6  6 58814  833  0 52  6 218 251173  2 657  710 618  0  7 19 74
 7  3 58815  825  0 100  8 359 544865  4 723  672 823  0 11 18 71
 8  3 58837  862  0 67 23 334 697884  5 686  814 712  0 12 20 67
19  8 58872  971  0 70  7 480 877304  7 1148  826 986 48  8  8 36
 8  3 58960  814  0 11 15 312 376204  3 902  829 709  0  9 23 69
10  2 58930  816  0  3 154 384 501383  4 699  895 940  0  9 25 67
10  2 58963  852  0  1 36 395 659926  5 664  793 608  0 13 34 53
19  5 58920  932  0 10 13 349 466821  4 886  640 621 57  6 11 26
 7  4 58951  874  0 114  59 336 126230  1 739  773 847  0  6 17 76
 8  3 58952  881  0 134  2 408 1742  0 755  700 999  0  5 19 76
 9  3 59023  845  0  0 10 342 645514  5 643  780 539  0 13 36 51
 8  1 59026  908  0  2 102 336 510580  4 689  659 566  0 10 30 59
 8  1 59027  906  0  4  75 360 501603  4 683  661 502  0 10 41 48
20  3 59028  918  0  7 18 576 707263  5 1099 1094 953 46  7 18 29
 7  6 59340  921  0 56 17 456 171002  1 660 34325 873  1  8 20 71
 8  3 59341  835  0  0 52 192 249882  2 612  663 624  0  7 23 70
 7  6 59033  839  0 47 39 288 250585  2 2698 2902 4798  0  2 12 86
```

From Example 42-10, it can be seen that the system has started to page due to a lack of memory in the Red WLM class. It can also be seen that the number of jobs in the run and wait queues are consistently high.

42.3 wlmmon / wlmperf

The **wlmmon** and **wlmperf** commands graphically display Workload Manager resource activity by class. **wlmmon** only monitors WLM class information over a period of up to 24 hours. The **wlmperf** command can monitor the WLM class information for days, weeks, or even months at a time for up to one year.

The **wlmmon** command resides in `/usr/bin` and is part of the `perfagent.tools` fileset, which is installable from the AIX base installation media.

The **wlmperf** command resides in `/usr/bin` and is part of the `perfmgr.analysis.jazizo` fileset, which is installable from the Performance Toolbox (PTX) media.

Note: In order for the **wlmmon** and **wlmperf** commands to function correctly, the following Java filesets must be installed:

- ▶ Java130.adt
- ▶ Java130.ext
- ▶ Java130.rte

The syntax of the **xmwlm** daemon is:

```
xmwlm {-d recording_dir} {-n recording_name} {-t trace_level}
```

where:

- d recording_dir** This flag specifies the output directory for the recording file(s). By default the directory name is `/etc/perf/wlm`.
- n recording_name** The recording file name is specified by this flag. By default the name of the file is `xmwlm.date`, where `date` is the system date at file creation time and is in the format `yymmdd`.
- t Trace level** This is the trace level and can be a whole number in the range of one to nine. The higher the trace number, the greater the amount of trace information gathered. The trace file is useful for debugging problems.

The syntax of the **xmtrend** daemon is:

```
xmtrend {-f infile} {-d recording_dir} {-n recording_name} {-t trace_level}
```

- f infile** The name of the configuration file that is used by **xmtrend** to determine which parameters to monitor. The default file name is `/etc/perf/xmtrend.cf`.

- d recording_dir** This flag specifies the output directory for the recording file(s). The default directory is /etc/perf.
- n recording_name** This flag specifies the name of the recording file. By default, **xmtrend** creates a recording file named **xmtrend.date**. If **-n myfile** is specified, the recording files will be named **myfile.date**, where **date** is the system date at file creation time in the format **yymmdd**.
- t trace_level** Trace level can be any whole number from one to nine. The log file is located in /etc/perf. The higher the value of the trace level, the greater the amount of trace information supplied. The log file name is **xmtrend.log1** and **xmtrend.log2** when **xmtrend.log1** is full.

The **wlmmmon** and **wlmpperf** commands do not have any arguments or flags. These commands are graphical in nature, so a graphical display is required to view report information from the log files. Both **wlmmmon** and **wlmpperf** can display data gathered by the **xmwlm** daemon in AIX 5L Version 5.2; however, for monitoring long-term statistics on AIX 4.3.3, the **wlmpperf** command and the **xmtrend** daemon are required. They are both part of the PTX media.

42.3.1 Information about the **xmwlm** and **xmtrend** daemons

This section discusses information about the daemons used by the WLM graphical commands. The daemons are used to gather the required information from the system. Both **xmwlm** and **xmtrend** (the daemons used by **wlmmmon** and **wlmpperf** respectively) can run when WLM is not active. The **xmwlm** daemon is installable from the AIX base operating system, and **xmtrend** is part of the PTX media.

Starting the daemons

The two daemons are initiated as follows.

xmwlm

The **xmwlm** daemon is used to gather information for the **wlmmmon** command. If there is a requirement to gather information on a daily basis, the **xmwlm** daemon can be started from the /etc/rc.tcpip file. Add this entry as the last line in the file:

```
xmwlm -n /etc/perf/myfile -d /etc/perf
```

The daemon can be started up by typing its name on the command line with the required flags, as shown in Example 42-11.

*Example 42-11 Starting the **xmwlm** daemon*

```
# nohup xmwlm -d /etc/perf/wlm -n devsys &
[1] 26292
```

```
# Sending nohup output to nohup.out.
```

The reason for using the **nohup** command is that the daemon is not terminated when the current window from which the command was issued is closed. Also, the **xmwl** command has been put into background by use of the **'&'**.

xmtrend

The **xmtrend** daemon, when used in a WLM environment, gathers information for the **xmperf** command. The **xmtrend** daemon can be started from the command line and, in order to start, needs a configuration file, which can be specified by using the **-f** flag. If the **-f** flag is not specified, then the **xmtrend** daemon will search in several predefined directories for the configuration file; if does not exist, the **xmtrend** daemon will not know which metrics to collect information for and will die. To start the daemon after each reboot, add this command to the **/etc/rc.tcpip** file:

```
xmtrend -f /etc/perf/xmtrend.cf -n /etc/perf/myfile -d /etc/perf
```

If the daemon will not start, it may be due to incorrectly terminating the daemons.

Attention: If you need to stop the **xmwl** or **xmtrend** daemons, do *not* use the **kill -9 PID** command. This may result in the shared memory of the System Performance Measurement Interface (SPMI) Application Program Interface (API) used by the **xmwl** and **xmtrend** daemons not being re-initialized. If changes are made to the WLM configuration, they will not be displayed by **wlmon** and **wlmp** after the **kill -9 PID** command has been used on them. Use the **kill PID** command instead. If the **kill -9 PID** command has been used to kill the daemons, this procedure can be used to correct the problem:

```
# ipcs -m
```

```
IPC status from /dev/mem as of Fri May 18 10:49:56 CDT 2001
```

```
T          ID      KEY          MODE          OWNER      GROUP
```

```
Shared Memory:
```

```
m      262153  0x7804129c  --rw-rw-rw-   root      system
```

```
#ipcrm -m 262153
```

```
#slibclean
```

Now restart the **xmwl** or **xmtrend** daemons

Look for any shared memory segments that start with **0x78xxxxxx** under the column headed **KEY**. Get the ID number (such as **262153**) to remove it.

At least 10 MB of disk space should be available in the directory where the recording files are to be kept.

The recording file created by the `xmtrend` daemon can also be monitored by the PTX tools `xmperf` and `3dmon`. Users may find the three-dimensional reports of the `3dmon` program useful. For more information about setting up metrics in PTX, refer to Chapter 43, “Performance Toolbox Version 3 for AIX” on page 891.

42.3.2 Information about measurement and sampling

Before it is possible to view any WLM statistics, it is necessary to ensure that the appropriate daemon is active on the system. While the `wlmstat` command provides a per-second view of WLM activity, it is not suited for the long-term analysis; it has no means of storing displayed data. To supplement the `wlmstat` command, the `wlmon` and `wlperf` commands provide reports of WLM activity over much longer time periods with minimal impact to system performance. The reports generated by these tools are based on samplings made by the associated recording daemon. These daemons sample the WLM and system statistics at a very high rate in seconds, but only record at a low rate in minutes. These values represent the minimum, maximum, mean, and standard deviation values for each collected statistic over the recording period. To collect the statistics, they use the same mechanism as the `wlmstat` command. Reports are generated from these statistics. For more information about the collection of statistics by the `wlmstat` command, refer to 42.2.1, “Information about measurement and sampling” on page 864.

42.3.3 Exploring the graphical windows

Figure 42-1 on page 876 shows the typical default screen of `wlmon` and `wlperf` when they are started. To start `wlmon` or `wlperf`, merely enter the names on the command line. Only users with root authority can run these commands. Across the bottom of the graphical display the following fields are displayed:

RunQ	The average number of threads in the run queue over the reporting period.
SwapQ	The average number of threads in the wait queue over the reporting period.
CPU Busy%	The percentage of time during the reporting period that the CPU was busy.
I/O Wait%	The amount of time that the CPU was idle while there was an outstanding I/O to disk (local or remote).

PagSp Used%	The percentage of paging space used during the reporting period.
Host	The name of the host on which the report information was captured.
WLM State	This field shows the state of WLM at the time of capturing the data, for example <i>Active</i> .
Period 1	This field is used when comparisons are made on the same report between different time periods. If no comparison is made, then this field is blank. This field represents the earlier of the two time periods. The format is: month/day start time - month/day end time.
Period 2	When comparisons are <i>not</i> being made to determine trends, the value in this field shows the date and time of the recording period in this format: month/day start time - month/day end time. When comparisons are being made, this field has the date and time of the second trend period.

The fields cannot be manually adjusted, but some of their contents can be changed by using the **Selected** tab down menu option seen in Figure 42-1.

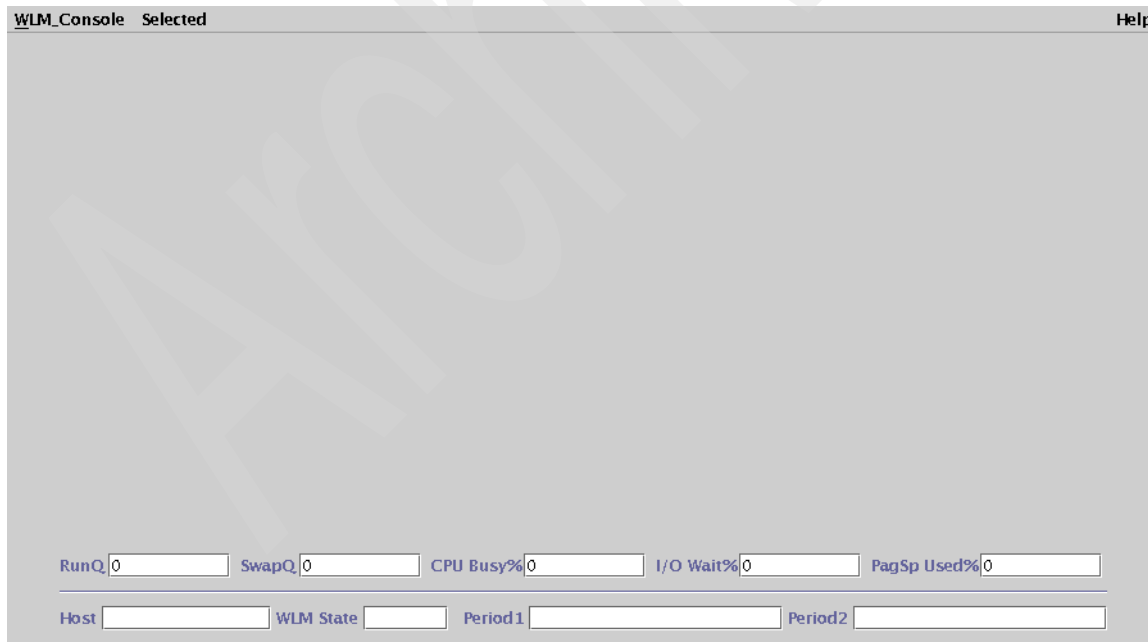


Figure 42-1 Initial screen when *wlmp perf* and *wlmm on* are started

The WLM_Console menu

Clicking on the **WLM_Console** tab down menu displays the following options:

Open log	Opens the required log file containing the report data.
Reports	Used to open, copy, and delete reports (wlmperv only).
Print	Prints the current report.
Exit	Exits the tool.

The tab down menu bar with its options is shown in Figure 42-2. Note that currently unavailable options are greyed out.

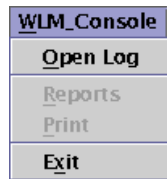


Figure 42-2 The WLM_Console tab down menu

Open log

Open a log file by choosing **Open Log** from the tab down bar. If the appropriate daemon is running on the system, then a log file or log files with a name similar to `xmwlm.010517` will be displayed. The first part of the name is the WLM daemon name, and the part after the period (.) is the date when the log file was created. These files are located in the directory `/etc/perf/wlm`, as shown in Figure 42-3.

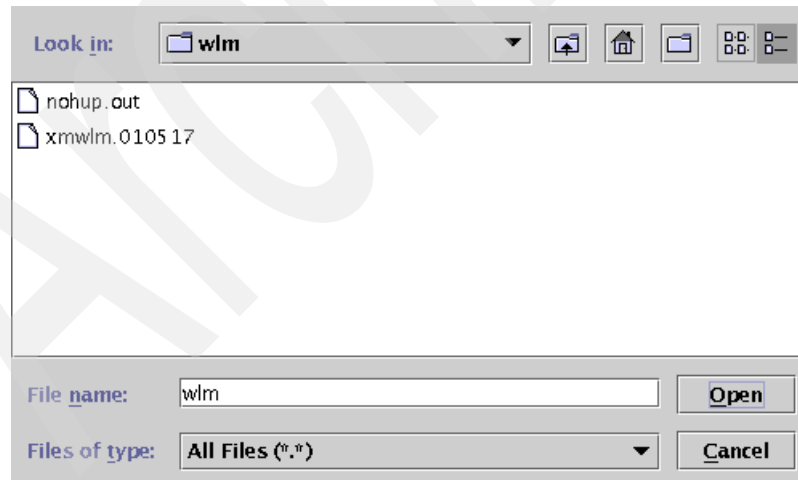


Figure 42-3 The open log option from the tab down bar

Move the mouse pointer to the appropriate file, in this case xmw1m.010517. Click the file name, then click **Open** to open the log file. This opens the reporting window in Figure 42-4.

Tier	Class	Shares	Target	Min	SMax	HMax	Actual	Low	High	StDev	Samples
0	db1.Shared	0	100	0	100	100	0	0	0	0	66
0	Red.Shared	0	100	0	100	100	0	0	0	0	66
0	devit.editors	0	100	0	100	100	0	0	0	0	66
6	System	0	100	0	100	100	13	0	51	6	66
0	Red.Editors	0	100	0	100	100	5	0	21	2	66
0	devit.hogs	0	100	0	100	100	0	0	0	0	66
0	acctg	0	100	0	100	100	0	0	0	0	66

RunQ: 1 SwapQ: 0 CPU Busy%: 49 I/O Wait%: 14 PagSp Used%: 9

Host: wlmhost WLM State: Active Period1: Period2: 5/17 14:08 - 5/17 17:38

Figure 42-4 The WLM table visual report

Three different reports can be displayed. They are **Table View**, **Bar View**, and **Snapshot View**. These views will now be discussed in detail.

Table View

The **Table View** provides actual numeric statistics for each class. Notice that the Table View shown in Figure 42-4 has a tab down menu. By default this tab down menu has the word **CPU** on it, indicating that CPU statistics are being displayed. Clicking on this tab down menu gives the options of **MEM** and **DISK I/O**, as shown in Figure 42-5.



Figure 42-5 The CPU, memory, and disk I/O tab down menu

Moving the mouse pointer to any of the options available from the tab down menu and clicking on the item results in the statistics for that item being displayed.

Across the top of the table view report screen, field headings are displayed. The field headings are:

Shares	The defined shares in the WLM configuration.
Target	The share value target computed by WLM in percentage. If the share is undefined, the value will be set to 100.
Min	The class minimum as defined in WLM limits.
SMax	The class soft maximum value as defined in WLM limits.
HMax	The class hard maximum value as defined in WLM limits.
Actual	The calculated average value over the sample period.
Low	The actual observed minimum value across the time period.
High	The actual observed maximum value across the time period.
Standard Deviation	The computed standard deviation of the Actual, High, and Low values. This value indicates the actual variability of the value across the recording period. A high value of standard deviation indicates more variability while a lower value of standard deviation indicates less variability.
Samples	The number of samples for the period.

On the extreme right of the example in Figure 42-4 on page 878, a slide bar is available. Often all of the classes cannot be displayed on one screen, and the slide bar allows the additional classes to be displayed.

Bar View

The **Bar View** visual report can be seen in Figure 42-6 on page 880. The class information is displayed in bar-graph style, together with the appropriate percentage values of the resource used over the specified time period. The percentage value is calculated based on the total system resources.

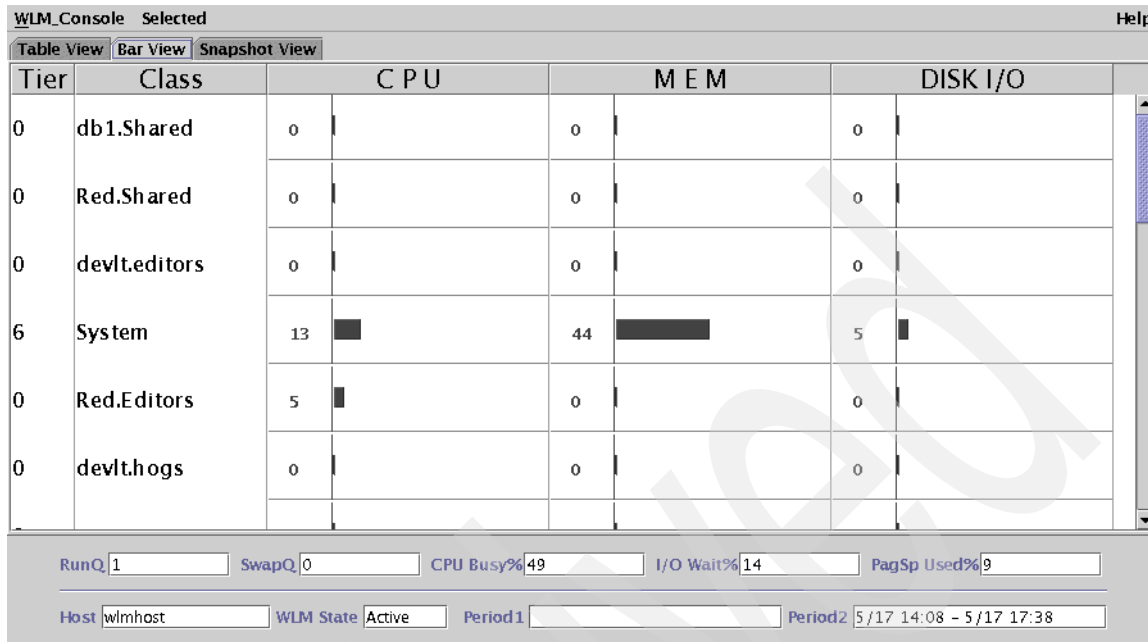


Figure 42-6 The bar-graph-style visual report

Snapshot View

The **Snapshot View** visual report is a quick reference to check that there are no serious eminent problems. No statistical values are displayed; instead, colored dots or bulbs indicate the status of a resource for a specific class. The color coding of the bulbs can be seen in the **Advanced Menu**.

The order of the bulbs is shown in Figure 42-7.

Colored circles or "bulbs" are used to associate displacement. Only one bulb is displayed at a time

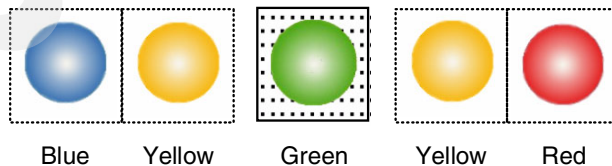


Figure 42-7 The order of the snapshot visual report colored bulbs

In the **Snapshot View** of Figure 42-8 on page 881, the resources are outside of the range and lower than the target. This is with the exception of the System class where the memory usage is between the inside and outside ranges. See advanced options for more details in Figure 42-14 on page 886.

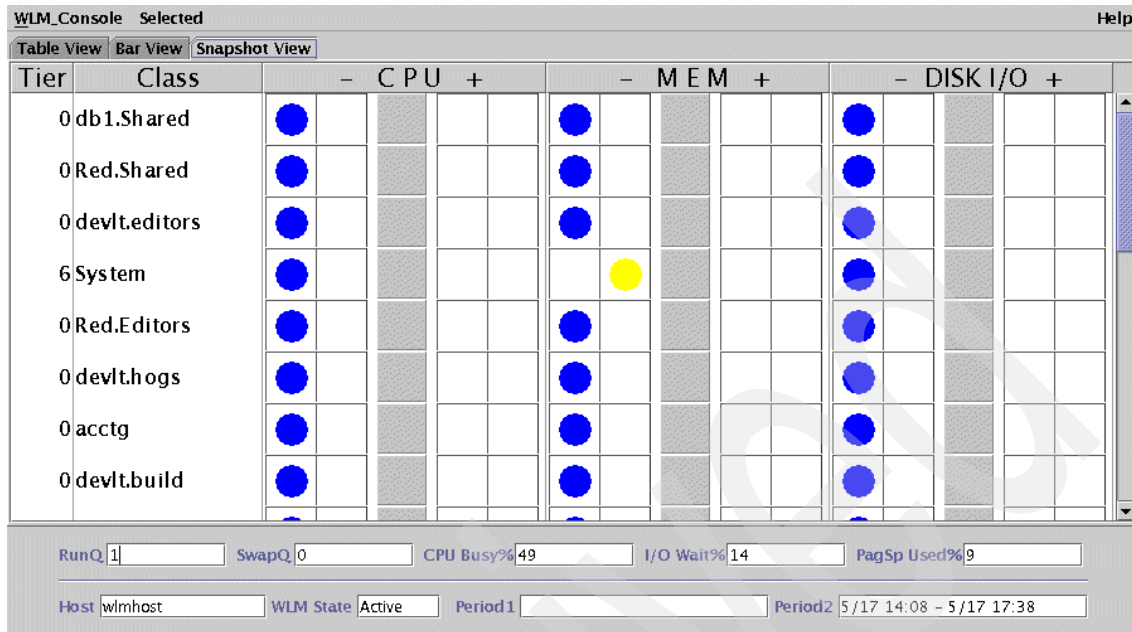


Figure 42-8 The snapshot visual report

The Selected menu

When the **Selected** menu option is chosen, the tab down menu in Figure 42-9 is displayed.

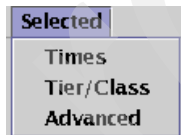


Figure 42-9 The Selected tab down menu

Times

The **Times** option displays the screen shown in Figure 42-10. If the **Trend** block is checked, then comparisons can be made between different times within the report file. In the example, the report period of 11am on May 18 is compared to the report period of 1pm on May 18. The width of the monitor interval can also be changed; default is five minutes.

Time Periods:

Time Range in recording: May 18, 10:37 AM - May 18, 13:48 PM

Trend :

Width of Interval: 5 min

End of First Period (MM:DD:hh:mm)

May	18	11 am	00
-----	----	-------	----

End of Last Period (MM:DD:hh:mm)

May	18	01 pm	00
-----	----	-------	----

Ok Cancel

Figure 42-10 The time window for setting trend periods

When selecting a trended view, the output of the visual report screens change slightly. In Figure 42-11 the figures are shown in parentheses. The Period1 field contains the first report period, while the Period2 field contains the second period value. The format of the date can be seen in the beginning of this section. In the visual reports, the values at the bottom of the screen, for example RunQ, are also in parentheses, comparing the difference at the two reporting periods.

WLM_Console Selected Help

Table View Bar View Snapshot View

CPU ▾

Tier	Class	Shares	Target	Min	SMax	HMax	Actual	Low	High	StDev	Samples
0	System	(3,3)	(100,100)	(0,1)	(14,16)	(17,14)	(12,16)	(3,2)	(27,28)	(0,0)	(4,5)
0	Batch1	(4,2)	(40,40)	(0,0)	(3,5)	(4,3)	(8,5)	(0,0)	(14,14)	(0,0)	(4,5)
0	Batch2	(2,5)	(5 0,5 0)	(0,0)	(3,2)	(5,4)	(10,8)	(6,1)	(17,17)	(0,0)	(4,5)
0	Shared	(3,4)	(28,28)	(2,2)	(4,3)	(4,4)	(4,2)	(1,0)	(9,7)	(0,0)	(4,5)
0	Unmanaged	(18,16)	(53,53)	(0,0)	(18,33)	(26,29)	(18,6)	(1,0)	(32,20)	(0,0)	(4,5)
0	Default	(8,11)	(26,26)	(0,0)	(2,4)	(5,4)	(3,5)	(0,0)	(9,8)	(0,0)	(4,5)

RunQ (0,0) SwapQ (0,0) CPU Busy% (24,21) I/O Wait% (5,3) PagSp Used% (22,21)

Host wimhost WLM State Active Period1 1/4 8:00 - 1/4 10:00 Period2 1/4 10:00 - 1/4 12:00

Figure 42-11 The table visual report with trend values shown

The bar-graph style report shows two bar graphs for each field of the report. In any field, the bottom value, represented by the white bar graph, shows the value of the first reporting period. The top, black bar graph shows the second period. This can be seen in Figure 42-12. Note that the trend values for the general statistics are also visible, at the bottom of the screen and in parentheses.

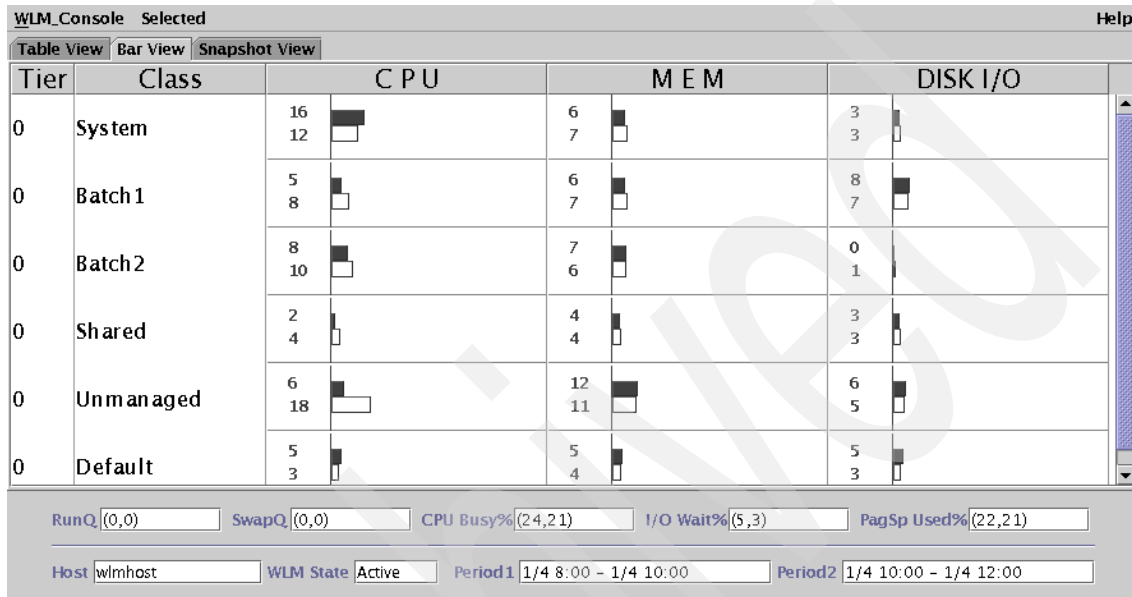


Figure 42-12 The bar-graph style report showing a trend

The Snapshot visual report can be seen in Figure 42-13. The greater than (>) and less than (<) symbols indicate the change. If there is no symbol indicated, then there was no change in this value. For example, the CPU usage was less at the second report period than at the first report period for the Unmanaged class.

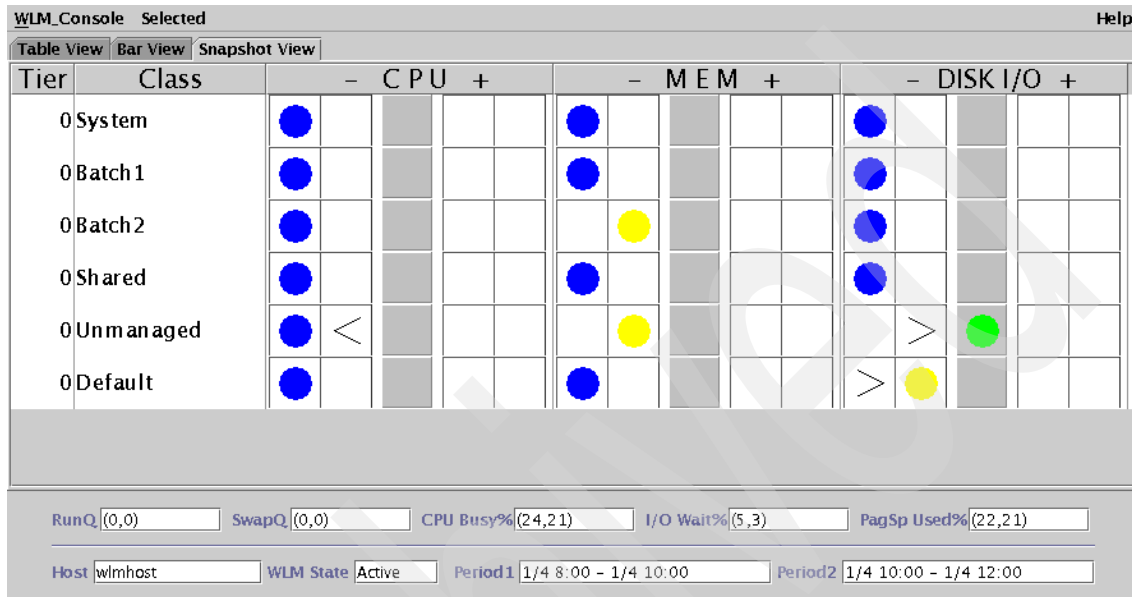


Figure 42-13 The snapshot visual report showing the trend

This example shows a snapshot visual report. This is a trend type of report, which means that there are two period values and trend indicators for the bulbs.

This snapshot has two report periods: the first is from 08:00 to 10:00 on January 4, and the second is from 10:00 to 12:00 on January 4. It shows that there is a general underutilization of resources; however, take into consideration that this is merely a snapshot of two short periods of one day so it may *not* be a true reflection of the state of the system. Ideally, further investigation should be conducted over additional time periods.

The trended view is ideal for a before-and-after view when resources have been reallocated on the system. For example, a trended snapshot view can be taken prior to increasing the amount of memory and CPU allocated to a class. Taking another trended snapshot view afterward will determine the effect of the changes on the classes.

Typically, the snapshot view brings to light the underutilization of resources for some classes, and overutilization in another. The idea of the snapshot is to instantly highlight possible problem areas.

Advanced

The **Advanced** option in the **Selected** tab down menu is used together with the snapshot visual report. This option is only appropriate for the snapshot visual report. The advanced option menu can be seen in Figure 42-14.

Snapshot Selections:

Option 1
Ignore user-defined min and max settings
Green Range falls within set %of the target share
Red Range falls outside set %of the target share

Option 2
Use %of the difference between target shares and specified min/max settings
Green Range falls within set %of the difference between target and min/max
Blue/Red Range falls outside set %of the difference between target and min/max

Green Range:
50

Blue/Red Range:
80

Ok Cancel

Figure 42-14 Advanced option under the Selected tab down menu

The interpretation of the colored bulbs changes with the option chosen. There are two options available: option one ignores the user-defined minimum and maximum settings, and option two uses a percentage of difference between the defined target and the minimum and maximum values.

Figure 42-15 shows the Advanced Menu options displayed in a graphical format. The graph labeled **Class** shows the **Soft** and **Hard** maximum and minimum limits. This graph is of a specific class that has a 50 percent **Target** (share value), a minimum of 20 percent (**Min**), and a maximum limit of 90 percent (**Max**).

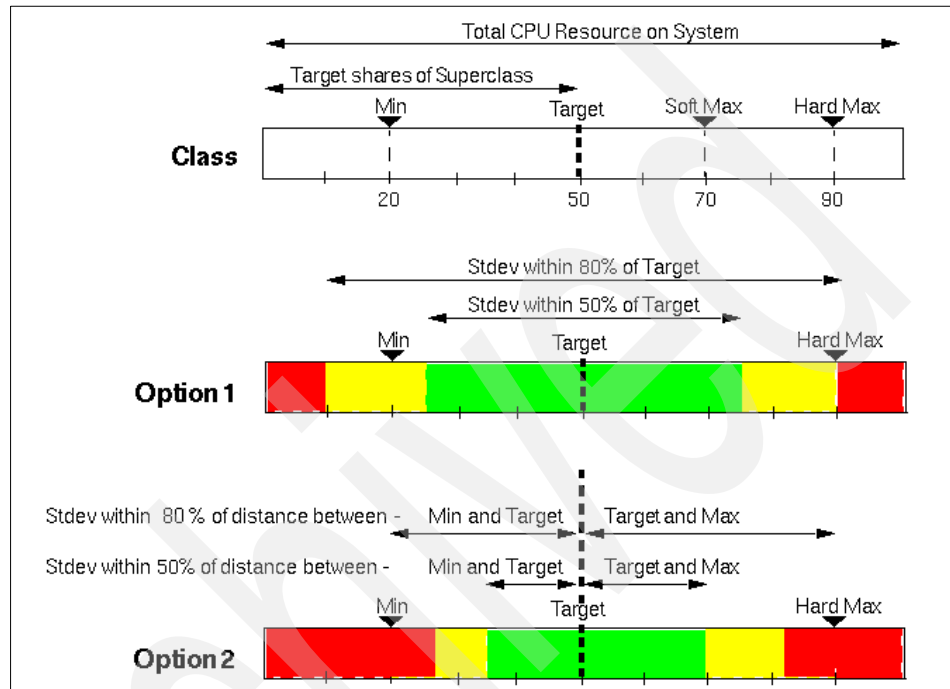


Figure 42-15 The Advanced Menu options shown in graphical form

The **Option1** graph ignores the user-defined maximum and minimum limits. Using the settings in Figure 42-14 on page 886, the green range of the graph is shown by the label **Stdev within 50% of Target**, and it is set to a value of 50 percent. The red range is shown by the label **Stdev within 80% of Target** and is set to a value of 80 percent.

The green range can be determined by the following formula:

$$\text{Low green range} = \text{Target} - (\text{Target} * \text{green}\%) = 50 - (50 * 50\%) = 25$$

$$\text{High green range} = \text{Target} + (\text{Target} * \text{green}\%) = 50 + (50 * 50\%) = 75$$

Therefore, the range covered by the arrowed line labeled **Stdev within 50% of Target** is from 25 to 75 percent. The values for the red range can be calculated:

$$\text{Low red range} = \text{Target} - (\text{Target} * \text{red}\%) = 50 - (50 * 80\%) = 10$$

$$\text{High red range} = \text{Target} + (\text{Target} * \text{red}\%) = 50 + (50 * 80\%) = 90$$

The red range, therefore, is from zero to 10 percent and 90 to 100 percent. This range is denoted in the figure by the arrowed line labeled *Stdev within 80% of Target*. The yellow range is that region between the red and green regions.

In the *Option2* graph, the predefined minimum and maximum settings are taken into account. If the same options as in Figure 42-14 on page 886 are selected, and the hard minimum (Min) and maximum (Max) values are 20 percent and 90 percent respectively, the ranges can be determined as follows:

$$\begin{aligned}\text{Low green range} &= \text{Target} - ((\text{Target} - \text{Min}) * \text{green}\%) \\ &= 50 - ((50 - 20) * 50\%) = 35 \\ \text{High green range} &= \text{Target} + ((\text{Max} - \text{Target}) * \text{green}\%) \\ &= 50 + ((90 - 50) * 50\%) = 70 \\ \text{Low red range} &= \text{Target} - ((\text{Target} - \text{Min}) * \text{red}\%) \\ &= 50 - ((50 - 20) * 80\%) = 26 \\ \text{High red range} &= \text{Target} + ((\text{Max} - \text{Target}) * \text{red}\%) \\ &= 50 + ((90 - 50) * 80\%) = 82\end{aligned}$$

Once again, the yellow range is between the red range and the green range.

Tier/Class

From the **Selected** menu, the **Tier/Class** option enables the viewing of a selected class or tier. The class/tier pane is shown in Figure 42-16. If the class **Red** was chosen from the list of classes, the snapshot output shown in Figure 42-17 on page 889 can be displayed.

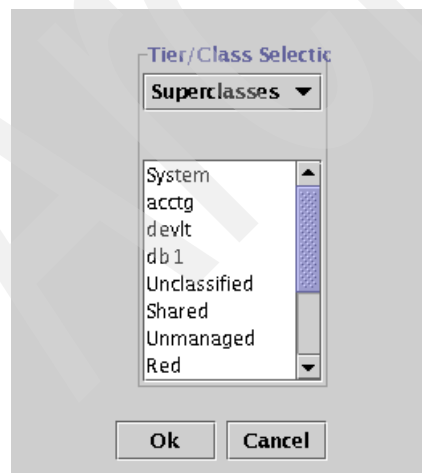


Figure 42-16 The class/tier option from the selected tab down menu

Only the Red superclass with its subclasses is shown in Figure 42-17, which helps to analyze report information for this specific class.

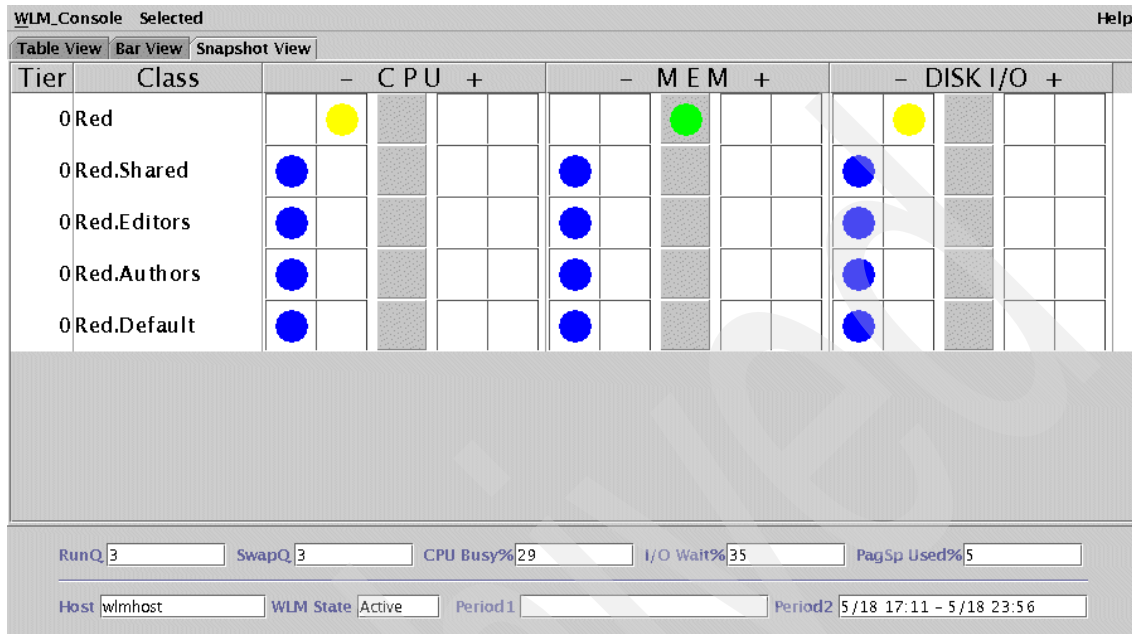


Figure 42-17 The snapshot report showing only the Red WLM class

Archived

Performance Toolbox Version 3 for AIX

This chapter describes the use of some of the utilities available in the AIX Performance Toolbox Version 3 (PTX). There are numerous component parts to the PTX, and this chapter will explain the use of the most important. For a full description of all of the features of the PTX, refer to *Performance Toolbox Version 2 and 3 Guide and Reference*.

This chapter shows examples of the use of the popular tools and explains the results obtained. The components that are covered in this document are:

- ▶ **xmperf**, which is useful for monitoring system statistics
- ▶ **3dmon**, which monitors performance of multiple hosts on the same network
- ▶ **jazizo**, which is ideal for monitoring long-term performance

The aim is to invite the reader to use the PTX to arrive at meaningful and useful conclusions to performance issues. We will not cover details on the System Performance Measurement Interface (SPMI) Application Program Interface (API) in depth. For more information about the SPMI, refer to 41.2, “System Performance Measurement Interface” on page 805.

43.1 Introduction

The Performance Toolbox Version 3 consists of two parts: the manager and the agent. The agent, also known as Performance AIDE, must be loaded on all nodes that are to be monitored by the manager. The PTX can be useful in performing the following functions:

- Load monitoring** Assists in monitoring system resources to detect performance problems.
- Analysis and control** When a problem is encountered, determines the correct tool for analyzing the problem; determines the root cause of the problem so that the necessary corrective action can be taken.
- Capacity planning** Performs long-term monitoring to determine in advance the correct quantity of additional resources required.

The PTX is a graphical tool, so it requires a suitable graphical monitor for display purposes.

The PTX filesets are:

- ▶ perfagent.server
- ▶ perfmgr.analysis.jazizo
- ▶ perfmgr.common
- ▶ perfmgr.network

The required AIX base fileset is:

- ▶ perfagent.tools

These filesets are available on the Performance Toolbox Version 3 media CDs. PTX Version 3, both manager and AIDE, is supported on AIX Version 4.3.3 and AIX 5L Version 5.2. When AIDE is installed on a remote host (node), it is necessary to refresh the `inetd` daemon. The refreshing of the daemon makes it aware of the PTX xmquery protocol. As a user with root authority, run the following command: **refresh -s inetd**.

Where an address of a remote client is outside of the local subnet, PTX requires the `Rsi.hosts` file to be edited on the manager host to include this remote client. In the case where there may be a name resolution delay it is recommended that the IP address is supplied instead of the hostname in the `Rsi.hosts` file. For more information about setting up the `Rsi.hosts` file, refer to the Web page:

http://www16.boulder.ibm.com/doc_link/en_US/a_doc_lib/perftool/prfusrgd/ch02body.htm

During the course of this chapter, as a performance tool is used, the specific fileset of which it is a component, together with the path name, will be supplied. The tools discussed in this chapter are: **xmperf**, **3dmon**, and **jazizo**.

Some other tools available in PTX:

3dplay	A program to play back 3dmon recordings in a 3dmon-like view.
chmon	Supplied as an executable as well as in source form, this program enables monitoring of vital statistics from a character terminal.
exmon	A program that enables monitoring of alarms generated by the filtd daemon running on remote hosts.
azizo	Legacy recording tool replaced by jazizo in PTX Version 3. Enables you to analyze any recording of performance data, lets you zoom in on sections of the recording, and provides graphical as well as tabular views of the entire recording or zoomed-in parts of it.
ptxtab	A program that can format statistics from recording files for printing.
ptxmerge	This program enables merging of up to 10 recording files into one. For example, you could merge xmservd recordings from the client and server sides of an application into one file to better correlate the performance impact of the application on the two sides.
ptxsplit	In cases where recording files are too large to analyze as one file, this program enables you to split the file into multiple smaller files for better overview and faster analysis.
ptxrlog	A program to create recordings in ASCII or binary format.
ptxls	A program to list the control information of a recording file, including a list of the statistics defined in the file.
a2ptx	The a2ptx program can generate recordings from ASCII files in a format as produced by the ptxtab or ptxrlog programs or the Performance Toolbox for AIX SpmiLogger sample program. The generated recording can then be played back by xmperf and analyzed with jazizo .
ptxconv	The format of recordings has changed between versions of the Performance Toolbox for AIX. As a convenience to users of multiple versions of the Performance Toolbox for AIX, this program converts recording files between those formats.
ptx2stat	Converts data collected in a recording file to a format that resembles the recording format for the statistic set. Permits postprocessing of data with the programs that enable playback and manipulation of recordings.

- ptxhottab** A program that can format and print hotset information collected in recording files.
- wlmpperf** Program for analyzing Workload Management (WLM) activity from **xmtrend** recordings. Provides reports on class activity across hours, days, or weeks in a variety of formats. This application is available only in PTX Version 3. See 42.3, “wlmmon / wlmpperf” on page 872 for more information.

43.2 xmpperf

The **xmpperf** program is used to monitor the statistics of a system. Monitoring system performance is one of a system administrator's most important functions. The **xmpperf** program has the ability to monitor statistics on the system where it is running as well as on remote systems via a network. The agent component must be running on all remote hosts as well as the host where the **xmpperf** program is running.

The **xmpperf** program resides in `/usr/bin`, which is part of the `perfmgr.network` fileset that is installable from the Performance Toolbox Version 3 for AIX media. The `perfmgr.common` and `perfagent.server` filesets are prerequisites for `perfmgr.network`. They are also installable from the AIX Performance Toolbox Version 3 media.

The syntax of the **xmpperf** command is:

```
xmpperf [-v auxz] [-w width] [-o options_file] [-p weight] [-h localhostname]
[-r network_timeout]
```

Flags

All command line flags are optional and all except `-r` and `-h` correspond to X Window System resources that can be used in place of the command line arguments.

The options `-v`, `-a`, `-u`, `-x`, and `-z` are true or false options. If one of those options is set through an X Window System resource, it cannot be overridden by the corresponding command line argument. The options are as follows:

- v** Verbose. This option prints the configuration file lines to the **xmpperf** log file `$HOME/xmpperf.log` as they are processed. Any errors detected for a line will be printed immediately below the line. The option is intended as a help to find and correct errors in a configuration file. Use the option if you do not understand why a line in your configuration file does not have the expected effect.

Setting the X Window System resource BeVerbose to true has the same effect as this flag.

- a** Adjusts the path name size that is displayed in an instrument to a minimum length. The length can be less than the default fixed length (or the length specified by the `-w` option if used) but never longer. The use of this option can result in consoles where the time scales are not aligned from one instrument to the next. Note that for pie chart graphs, adjustment is always done, regardless of this command line argument. Setting the X Window System resource LegendAdjust to true has the same effect as this flag.
- u** Use pop-up menus. As described in Console Windows, the overall menu structure can be based upon pull-down menus (which is the default) or pop-up menus as activated with this flag. Typically, pull-down menus are easier for occasional users to understand, while pop-up menus provide a faster but less-intuitive interface. Setting the X Window System resource PopupMenus to true has the same effect as this flag.
- x** Subscribe to exception packets from remote hosts. This option makes `xmperf` inform all the remote hosts it identifies that they should forward exception packets produced by the `filtd` daemon, if the daemon is running. If this flag is omitted, `xmperf` will not subscribe to exception packets. Setting the X Window System resource GetExceptions to true has the same effect as this flag.
- z** For monochrome displays and X stations, you might try the `-z` option, which causes `xmperf` to draw graphical output directly to the display rather than always redrawing from a pixmap. By default, `xmperf` first draws graphical output to a pixmap and then, when all changes are done, moves the pixmap to the display. Generally, with a locally attached color display, performance is better when graphical output is redrawn from pixmaps. Also, a flaw in some levels of X Window System can be bypassed when this option is in effect. Setting the X Window System resource DirectDraw to true has the same effect as this flag.
- w width** Must be followed by a number between 8 and 32 to define the number of characters from the value path name to display in instruments. The default number of characters

is 12. Alternatively, the legend width can be set through the X Window System resource `LegendWidth`.

- o options_file** Must be followed by a file name of a configuration file to be used in the execution of `xmperf`. If this option is omitted, the configuration file name is assumed to be `$HOME/xmperf.cf`. Alternatively, the configuration file name can be set through the X Window System resource `ConfigFile`.
- p weight** If given, this flag must be followed by a number in the range 25-100. When specified, this flag turns on “averaging” or “weighting” of all observations for state graphs before they are plotted. The number is taken as the “weight percentage” to use when averaging the values plotted in state graphs. The formula used to calculate the average is:
- $$\text{val} = \text{new} * \text{weight}/100 + \text{old} * (100 - \text{weight}) / 100$$
- where:
- `val` is the value used to plot.
 - `new` is the latest observation value.
 - `old` is the `val` calculated for the previous observation.
- h local hostname** Must be followed by the host name of a remote host that is to be regarded as `Localhost`. The `Localhost` is used to qualify all value path names that do not have a host name specified. If not specified, Local host defaults to the host where `xmperf` executes.
- r network_timeout** Specifies the timeout, in milliseconds, used when waiting for responses from remote hosts. The value specified must be between 5 and 10000. If not specified, this value defaults to 100 milliseconds. In systems where the subnets are large, it is recommended to set the value of this flag to 10000.

Note: On networks that extend over several routers, gateways, or bridges, the default value is likely to be too low.

One indication of too low a timeout value is when the list of hosts displayed by `xmperf` contains many host names that are followed by two asterisks. The two asterisks indicate that the host did not respond to `xmperf` broadcasts within the expected timeout period.

Parameters

width	Length of the path name to be used in graphical instruments.
options_file	Configuration file name.
weight	Weight specified by the -p flag. If a number outside the valid range is specified, a value of 50 is used. If this flag is omitted, averaging is not used. The averaging weight can be set through the X Window System resource Averaging as well. The weight also controls calculation of weighted average in tabulating windows.
local_hostname	Name of a remote host.
network_timeout	Network timeout value in milliseconds.

43.2.1 Information about measurement and sampling

Over and above the ability to monitor performance in real time, the `xmperf` program has the functionality to record performance and to play back a recording. The `xmperf` program uses the SPMI API to gather information. The information is of two types dependant on the instrument type:

SiCounter	The value is incremented continuously and the instrument shows the change in the value between time intervals.
SiQuantity	The value represents a level. The observed value is displayed by the instrument.

Display requirements

The `xmperf` program requires a graphical display for its output, but the display need not be the system console. If, for example, the system `res07` is an AIX system and it has a color graphical monitor, that is ideal for displaying `xmperf` statistics. The system `wlmhost` has the manager part of the PTX installed, but has no graphical display. To monitor the statistics on the `res07` monitor, first run the `xhost +` command on the `res07` system. The system will show the message:

```
access control disabled, clients can connect from any host
```

The `xhost` command enables the system `wlmhost` to output to the X-Window of `res07`.

It is now necessary to export the display of `res07` on the system `wlmhost`:

```
export DISPLAY=res07:0
```

Ensure that the remote system name, in this case `res07`, appears in the `/etc/hosts` file of the `wlmhost` system. Once the X-session has been established, the `xmperf` program can be run.

Starting xmpperf

Start **xmpperf** without any arguments. The window in Figure 43-1 will be displayed.

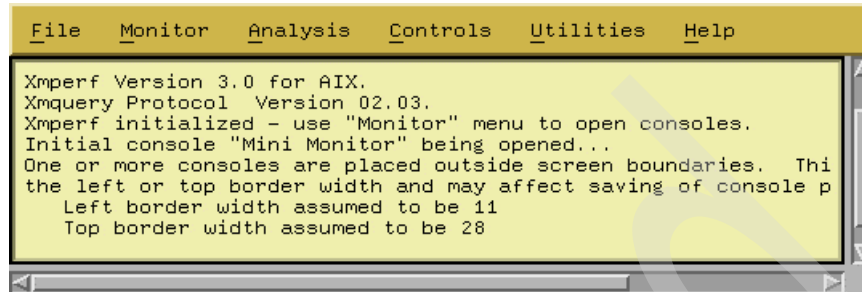
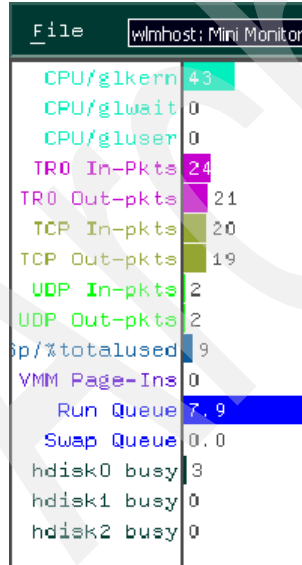


Figure 43-1 The initial xmpperf window

The windows or panes that are displayed by the **xmpperf** program are referred to as consoles. Note also that the “Mini Monitor” console is displayed by default as seen in Figure 43-2.

Monitoring instruments occupy a rectangular area within a console. An instrument has the ability to plot 24 values simultaneously. Only values from the same system can be displayed in an instrument.



The screenshot shows a window titled 'wlmhost: Mini Monitor' with a menu bar containing 'File'. The main text area displays a list of monitoring instruments and their values:

CPU/glkern	43
CPU/glwait	0
CPU/gluser	0
TR0 In-Pkts	24
TR0 Out-pkts	21
TCP In-pkts	20
TCP Out-pkts	19
UDP In-pkts	2
UDP Out-pkts	2
\$p/%totalused	9
VMM Page-Ins	0
Run Queue	7.9
Swap Queue	0.0
hdisk0 busy	3
hdisk1 busy	0
hdisk2 busy	0

Figure 43-2 The Mini Monitor window

The **xmpperf** program displays two types of monitoring instruments: recording instruments and state instruments.

Recording instruments

Recording instruments monitor information over a period of time. They should not be confused with recording or saving data to a disk. The displayed values are pushed to the left as they are replaced by the new values. In Figure 43-3, the aged or older data is moved to the left as it is replaced with new data. The data is moved to the left at a user-defined time interval.

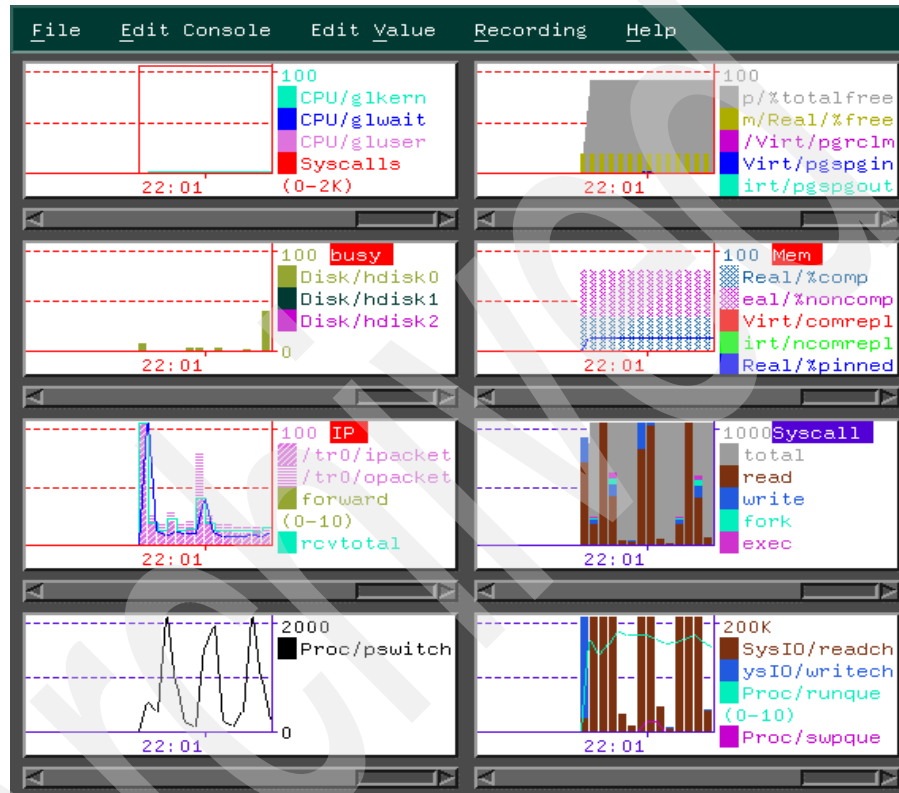


Figure 43-3 Aged data moved to the left

State instruments

State instruments show the latest information for a system resource. The “Mini Monitor” console shown in Figure 43-2 on page 898 is a state instrument.

Primary graphic styles

Various primary graphic styles are available for each of the instrument types. For recording instruments, the styles are:

- ▶ Skyline
- ▶ Bar graph
- ▶ Area

- ▶ Line

For the state instruments, the styles are:

- ▶ Pie chart
- ▶ State bar
- ▶ State light
- ▶ Speedometer

The **xmperf** program enables a system administrator to create consoles. A console can consist of more than one graphic type and may consist of both recording and state instruments. See 43.2.2, “Examples” on page 904 for details on how to set up a user-defined console.

System commands

In addition to the monitoring features, the **xmperf** program has an enhanced interface to system commands. From the initial **xmperf** window shown in Figure 43-1 on page 898, the tab down menus **Utilities**, **Analysis**, and **Control** can be used to access the system commands. The tab down menus and some of their commands are listed next (Figure 43-4, Figure 43-5 on page 901, and Figure 43-6 on page 901).

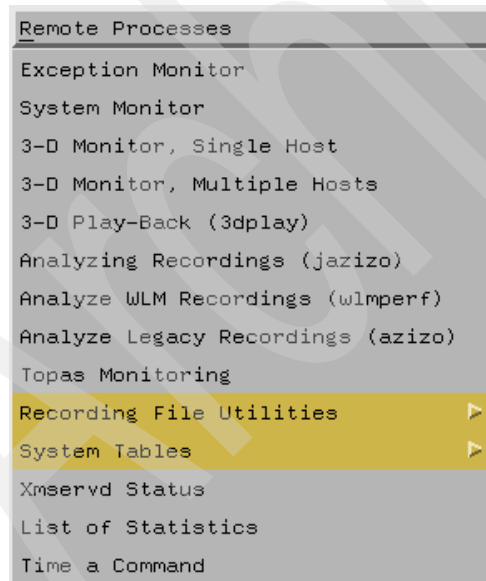


Figure 43-4 The Utilities tab down menu

Utilities Includes the **topas**, **wlmperf**, **time**, and **pstat** commands.

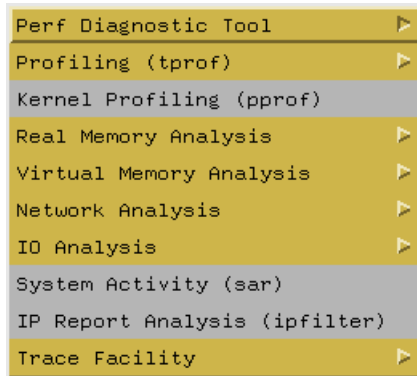


Figure 43-5 The Analysis tab down menus

Analysis Includes the **tprof**, **pprof**, **PDT**, **rmss**, **svmon**, **vmstat**, **netpmn**, **netstat**, **nfsstat**, **filemon**, **iostat**, **fileplace**, **sar**, **ipfilter**, and **trace** commands.

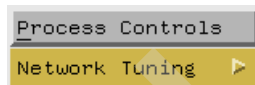


Figure 43-6 The Controls tab down menu

Control Includes the **no** and **chnfs** command.

Recording and playback

As previously mentioned, the **xmperf** program is able to save the recorded information to a file. This information can be played back for analysis at a later time. To record information from a console, select that console as in Figure 43-3 on page 899. Choose the **Record** tab down menu as shown in Figure 43-7 and select **Console Recording**.



Figure 43-7 The Recording tab down menu

In the menu option shown in Figure 43-8 on page 902, choose the option **Begin Recording** to start the recording.



Figure 43-8 The Console Recording options

If the data from this instrument has been recorded before, then the cautionary window in Figure 43-9 will be displayed. This is to prevent accidental overwriting of the existing file. Note that there is an option to **Append** to the existing file. Choosing the **Replace** option will overwrite the file.

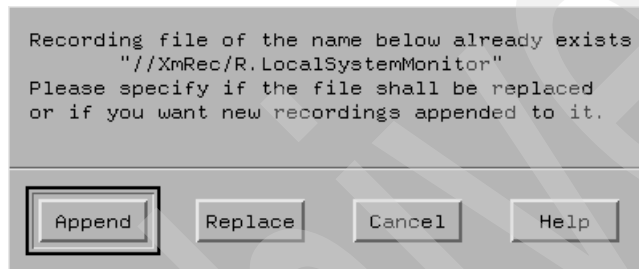


Figure 43-9 Cautionary window when recording an instrument

Choosing either the **Append** or **Replace** options starts the recording process. To stop the recording, select the tab down **Recording** menu shown in Figure 43-3 on page 899 and select **Console Recording**. Select the **End Recording** option as shown in Figure 43-10. This option ends the recording to file.

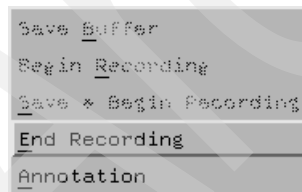


Figure 43-10 Console Recording tab down menu: End Recording option

In order to play back the recording, select the **File** tab down menu of the initial **xmperf** window shown in Figure 43-1 on page 898. The window in Figure 43-11 on page 903 is displayed. From this menu, select **Playback**.

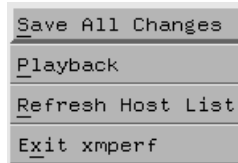


Figure 43-11 Options under the initial xperf window File tab down menu

The window in Figure 43-12 is displayed showing the list of playback files. Select the appropriate file by moving the mouse pointer to the file.

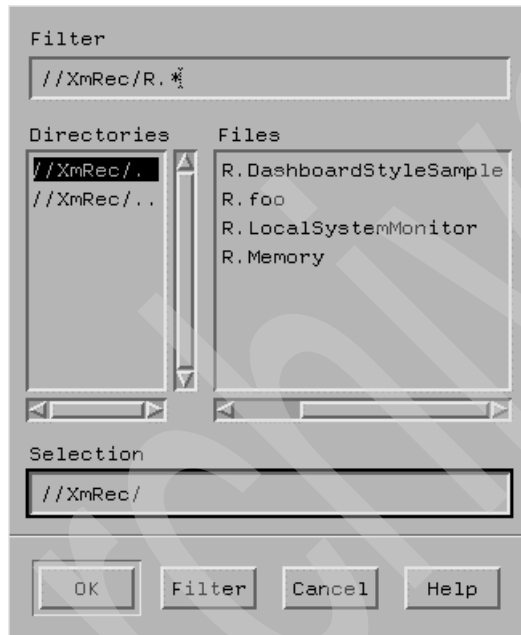


Figure 43-12 The Playback window

Once the file name has been selected, click **OK** at the bottom of the window. The window in Figure 43-13 on page 904 shows the playback monitor.



Figure 43-13 The playback monitor

Click **Play** to review the monitored data.

43.2.2 Examples

Follow these steps to use the **xmperf** program to create a user-defined console.

From the **xmperf** initial window shown in Figure 43-1 on page 898, choose the **Monitor** tab down menu. From this menu, select **Add New Console**, to display the window in Figure 43-14. The number in the top field is constructed by the system from the date and time. Overwrite it with a name for the new console. For syntax and illegal character information, click **Help**.

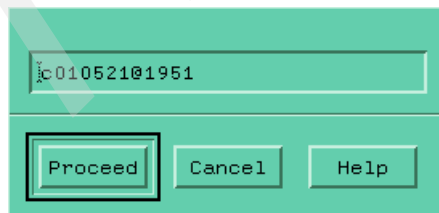


Figure 43-14 Naming the user-defined console

To continue creating the console, click **Proceed** to display the window in Figure 43-15. From the **Edit Console** menu, select **Add Local Instrument**.

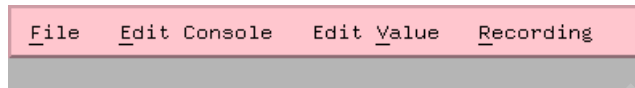


Figure 43-15 Choose the Edit Console menu

The window in Figure 43-16 shows a list of the resources that can be monitored. Make the selection by moving the mouse pointer to the required resource and clicking the left mouse button. Several other windows with options for the chosen resource will be displayed. These options vary from resource to resource so are not displayed here.

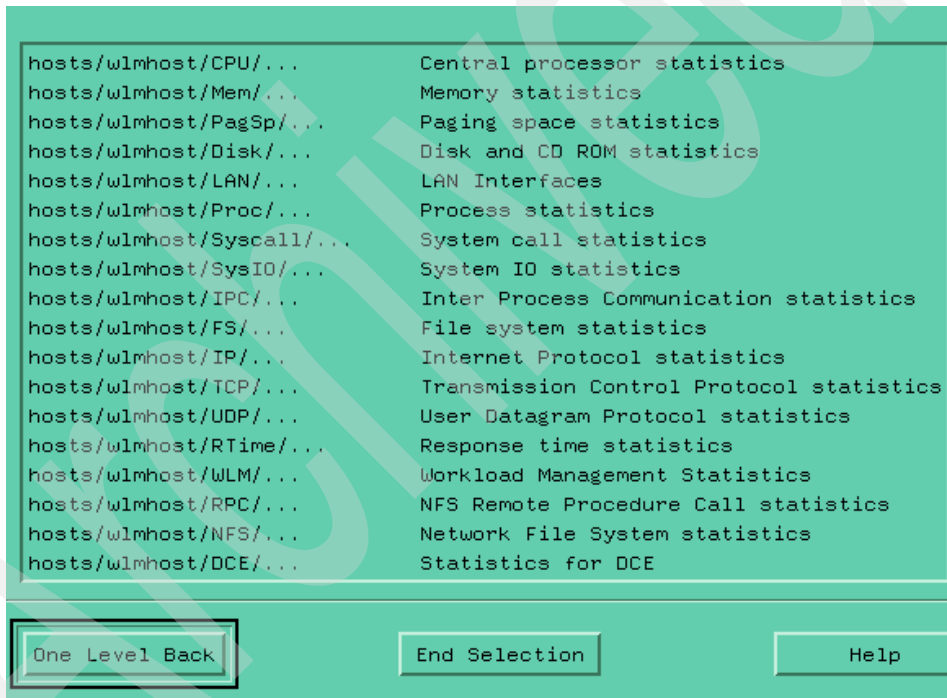


Figure 43-16 Dynamic Data Supplier Statistics window

When all required selections have been made, the Changing Properties of a Value window (Figure 43-17 on page 906) is displayed. The color and type of graph can be changed as well as the maximum, minimum, and threshold values. In this example, CPU idle time was monitored. The graph type can be changed to either line, area, skyline, or bar. In this example, the default style (line) was used.

Properties in this window include:

Lower range for value The lowest point to be displayed on the graph. Typically, this value would be set to zero.

Upper range of value The highest point plotted on the graph. It determines the scale of the graph. This property is not required for the state light graph type.

Threshold for value The value at which the state light will change state. This property is only required by the state light graph type.

hosts/wlmhost/CPU/gldle
System-wide time CPU is idle (percent)

Color: ForestGreen

Line Area Skyline Bar

Your label:

0
Lower range for value

100
Upper range for value

0
Threshold for value

Alarm when above threshold Alarm when below threshold

OK Cancel Help

Figure 43-17 The Change Properties of a Value window

Finally, the monitoring console can be seen in Figure 43-18. This is a recording instrument, and as a new reading is taken, the older or aged values are pushed to the left.

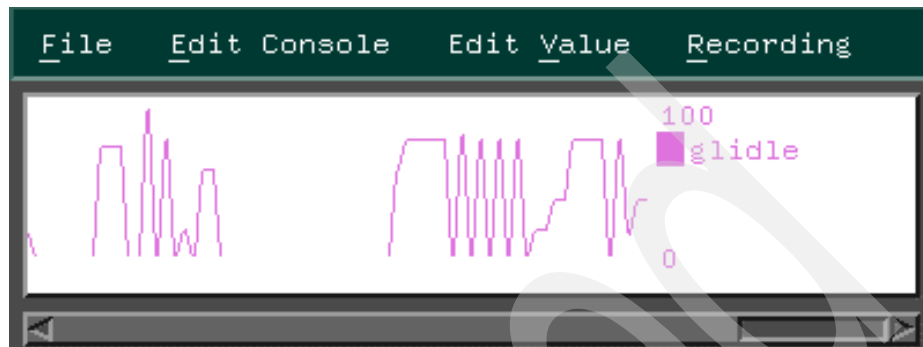


Figure 43-18 The final console monitoring CPU idle time

The console can be enhanced further by changing the name from `glidle` to something more meaningful. Click on the instrument to be changed. (When selected, the graphic will have a phantom border around it.) From the **Edit Value** menu, select **Change Value** to open the Change Properties of a Value window shown in Figure 43-17 on page 906. Change the label name of the recording instrument by typing over the name `glidle` in the label box and clicking **OK**.

A more useful instrument in this case may be the pie chart for CPU usage. For example, a pie chart would be ideal for representing the percentage of time the CPU is in user mode, in kernel mode, idle while an I/O is outstanding, and idle (excluding the time that it is waiting for I/O). To change to pie chart style, select the instrument to be changed. Click the **Edit Console** menu and select **Modify Instrument**, as shown in Figure 43-19.

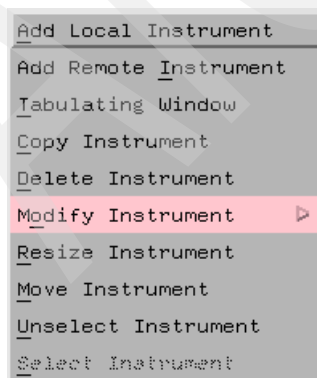


Figure 43-19 The Edit Console tab down menu

The options are shown in Figure 43-20. Select **Style & Stacking** from the menu.

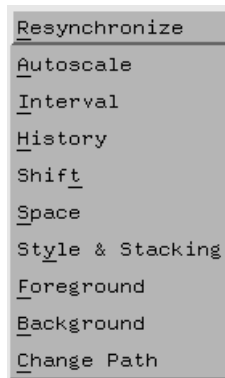


Figure 43-20 The Modify Instrument menu options

This opens the window in Figure 43-21. Select the **Pie_Chart** option and the **Stacking** option, then click **Proceed** to close the window.

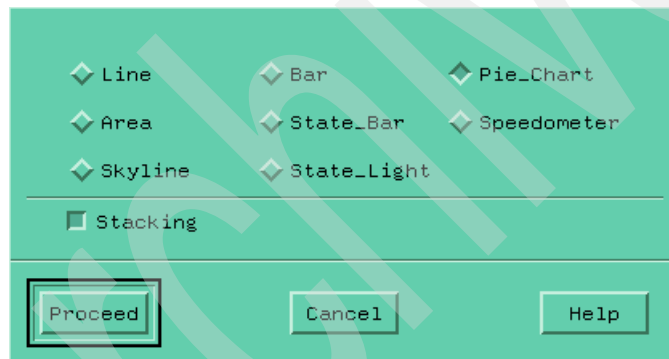


Figure 43-21 The Style & Stacking menu option

To include the additional CPU statistics, select the **Edit Value** tab down menu in the instrument console shown in Figure 43-18 on page 907. This displays the menu options shown in Figure 43-22.

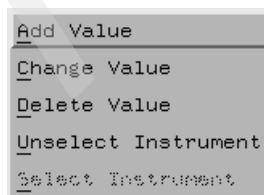


Figure 43-22 Menu options from the Edit Value tab down menu

Select **Add Value** from the menu. This opens the Dynamic Data Supplier Statistics window shown in Figure 43-16 on page 905. The procedure from this point is the same as for creating a new instrument. Figure 43-23 shows the final result, with all of the CPU statistics are being measured by the same instrument.

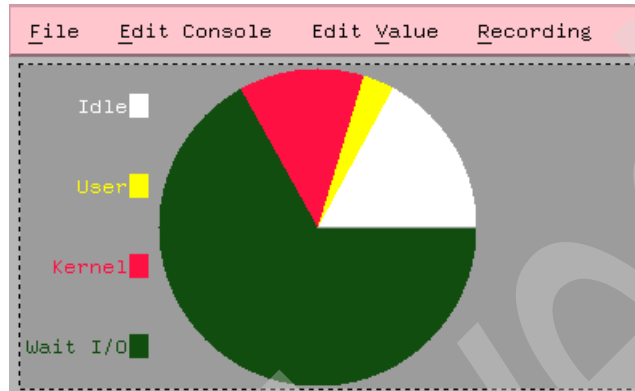


Figure 43-23 An example of a CPU usage instrument

43.3 3D monitor

The **3dmon** program is useful for monitoring the same statistics on numerous hosts across a network. The results of the **3dmon** program are three-dimensional, graphical, chessboard-like outputs. The number of fields can range from one to 24. This is a graphical program and hence requires a graphical monitor.

The **3dmon** program resides in `/usr/bin` and is part of the `perfmgr.network` fileset, which is installable from the Performance Toolbox Version 3 for AIX media. The `perfmgr.common` and `perfagent.server` filesets are prerequisites for `perfmgr.network`. They are also installable from the Performance Toolbox media.

The syntax of the **3dmon** command is:

```
3dmon [-vng] [-f config_file] [-i seconds_interval] [-h hostname]
      [-w weight_percent] [-s spacing] [-p filter_percent] [-c config]
      [-a "wildcard_match_list"] [-t resync_timeout] [-d invitation_delay]
      [-l left_side_tile] [-r right_side_tile] [-m top_tile]
```

Flags

-v Verbose mode. Causes the program to display warning messages about potential errors in the configuration file to standard error. Also causes **3dmon** to print a line for each statset created and for

- each statistic added to the statset, including the results of re-synchronizing.
- n** This flag only has an effect if a filter percentage is specified with the **-p** argument. When specified, draws only a simple outline of the grid rectangles for statistics with values that are filtered out. If not specified, a full rectangle is outlined and the numerical value is displayed in the rectangle.
- g** Usually, **3dmon** attempts to resynthesize for each statset; it does not receive data-feeds for resync-timeout seconds. If more than half of the statsets for any host are found to not supply data-feeds, re-synchronizing is attempted for all of the statsets of that host. By specifying the **-g** option, you can force re-synchronization of all statsets of a host if any one of them becomes inactive.
- f config_file** Allows the specifying of a configuration file name other than the default. If not specified, **3dmon** looks for the `$HOME/3dmon.cf` file.
- i seconds interval** If specified, this argument is taken as the number of seconds between sampling of the statistics. If omitted, the sampling interval is five seconds. You can specify a one- to 60-second sampling interval.
- h hostname** Used to specify which host to monitor. This argument is ignored if the specified wildcard is `hosts`. If omitted, the local host is assumed.
- w weight_percent** Modifies the default weight percentage used to calculate a weighted average of statistics values before plotting them. The default value for the weight is 50 percent, meaning that the value plotted for statistics is composed of 50 percent of the previously plotted value for the same statistic and 50 percent of the latest observation. The specified percentage is taken as the percentage of the previous value to use. For example, if you specify 40 with this argument, the value plotted is:
- $$0.4 * \text{previous} + (1 - 0.4) * \text{latest}$$
- Weight can be specified as any percentage from 0 (zero) to 100.

- s spacing** Spacing (in pixels) between the pillars representing statistics. The default space is four pixels. You can specify from 0 (zero) to 20 pixels.
- p filter_percent** If specified, only statistics with current values of at least -p percent of the expected maximum value for the statistic are drawn. The idea is to allow you to specify monitoring "by exception" so statistics that are approaching a limit stand out while others are not drawn. Filtering can be specified as any percentage from 0 (zero) to 100. The default is 0 (zero) percent.
- c config** When specified, overrides the default configuration set and causes **3dmon** to configure its graph using the named configuration set. The argument specified after the -c flag must match one of the wildcard stanzas in the configuration file. If this argument is omitted, the configuration set used is the first one defined in the configuration file.
- a wildcard_match_list** Wildcard match list. When specified, is assumed to be a list of host names. If the primary wildcard in the selected configuration set is hosts, then the list to display host names is suppressed, as **3dmon** automatically selects the supplied hosts from the list of active remote hosts. Depending on the configuration set definition, **3dmon** then either goes directly on with displaying the monitoring screen or, when additional wildcards are present, displays the secondary selection list.
- The list of host names must be enclosed in double quotation marks if it contains more than one host name. Individual host names must be separated by white space or commas. The primary purpose of this option is to allow the invocation of **3dmon** from other programs. For example, you could customize NetView® to invoke **3dmon** with a list of host names, corresponding to hosts selected in a NetView window.
- t resync_timeout** Re-synchronizing timeout. When specified, overrides the default time between checks for whether synchronizing is required. The default is 30 seconds; any specified timeout value must be at least this long.

- d invitation delay** Enables you to control the time **3dmon** waits for remote hosts to respond to an invitation. Value must be given in seconds; it defaults to 10 seconds. Use this flag if the default value results in an incomplete list of hosts when you want to monitor remote hosts.
- l left_side_tile** Specifies the number of the tile to use when painting the left side of the pillars. Specify a value in the range 0 (zero) to 8 (eight). The values correspond to the tile names:
- 0: foreground (100% foreground)
 - 1: 75_foreground (75% foreground)
 - 2: 50_foreground (50% foreground)
 - 3: 25_foreground (25% foreground)
 - 4: background (100% background)
 - 5: vertical
 - 6: horizontal
 - 7: slant_right
 - 8: slant_left
- The default tile number for the left side is 1 (75_foreground).
- r right_side_tile** Specifies the number of the tile to use when painting the right side of the pillars. Specify a value in the range 0 (zero) to 8 (eight). The values correspond to the tile names specified above for option -l. The default tile number for the right side is 8 (slant_left).
- m top_tile** Specifies the number of the tile to use when painting the top of the pillars. Specify a value in the range 0 (zero) to 8 (eight). The values correspond to the tile name specified above for option -l. The default tile number for the top is 0 (foreground).

43.3.1 Information about measurement and sampling

To start the **3dmon** program as the root user, enter the command **3dmon -i1**. The sampling interval will be one second as selected by the -i flag in the command.

The initial screen in Figure 43-24 is displayed. A host name or host names must be selected. To select more than one host name, click on the first host name, then hold the Ctrl key and click to highlight the second host name as well. When this selection has been made, click on **Click here when selection complete**.

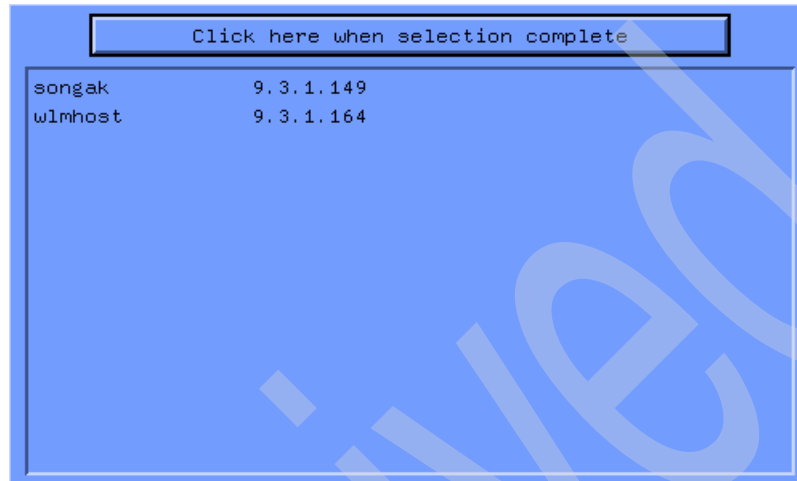


Figure 43-24 Initial 3dmon screen

The window in Figure 43-25 shows a typical chessboard-style three-dimensional window output of the **3dmon** command. The name of the host can be seen on the right-hand side. In front of each graphic bar is the name of the statistic that is being monitored and displayed. On top of each bar is the actual value of the statistic being measured. In this case, only one host is being monitored. If multiple hosts were being monitored, the three-dimensional display would be staggered behind the first display.

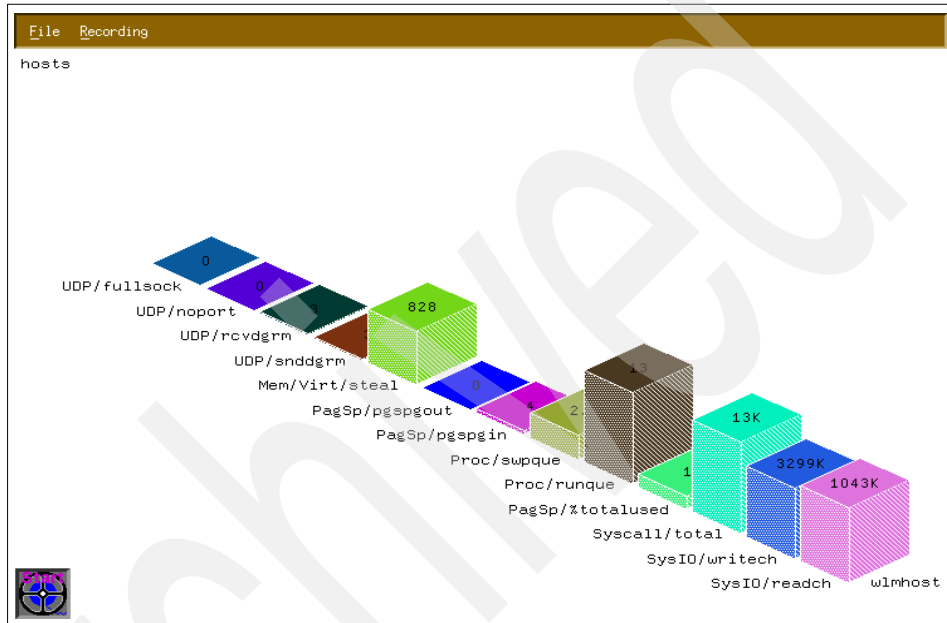


Figure 43-25 3-D window from 3dmon showing the statistics of a host

The monitored values in Figure 43-25 can be found in the 3dmon.cf configuration file. Usually this file is in the user's home directory. In this case, the file is in /usr/lpp/perfmgr. This file can be modified to produce customized graphical monitoring instruments. Run the **xmpeek -l** command for a listing of the metrics that can be measured. Example 43-1 shows the 3dmon.cf configuration file.

Example 43-1 The 3dmon configuration file 3dmon.cf

```
# @(#)01 1.7 src/perf/perfmgr/usr/samples/perfmgr/3dmon.cf, perfmgr,
43perf30
0, 0101A_43perf300 12/3/96 06:52:31

#
# COMPONENT_NAME: (PERFMGR) - Performance Manager
#
# FUNCTIONS: Configuration file
#
```



```

# ORIGINS: 30
#
# (C) COPYRIGHT International Business Machines Corp. 1992, 1993
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
wildcard:  hosts          # remote hosts
UDP/fullsock
UDP/noport
UDP/rcvdgrm
UDP/snddgrm
Mem/Virt/steal
PagSp/pgspgout
PagSp/pgspgin
Proc/swpqe
Proc/runque
PagSp/%totalused
Syscall/total
SysIO/writtech
SysIO/readch

```

The **3dmon** program uses the SPMI API to obtain the kernel statistics.

43.3.2 Examples

Example 43-2 shows the **3dmon.cf** file modified to produce a customized output.

Example 43-2 Customizing the 3dmon.cf file

```

# @(#)01 1.7 src/perf/perfmgr/usr/samples/perfmgr/3dmon.cf, perfmgr,
43perf30
0, 0101A_43perf300 12/3/96 06:52:31

#
# COMPONENT_NAME: (PERFMGR) - Performance Manager
#
# FUNCTIONS: Configuration file
#
# ORIGINS: 30
#
# (C) COPYRIGHT International Business Machines Corp. 1992, 1993
# All Rights Reserved
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
wildcard:  myconfig      hosts
CPU/cpu0/user

```

```

CPU/cpu0/kern
CPU/cpu0/wait
CPU/cpu0/idle
CPU/cpu1/user
CPU/cpu1/kern
CPU/cpu1/wait
CPU/cpu1/idle
CPU/cpu2/user
CPU/cpu2/kern
CPU/cpu2/wait
CPU/cpu2/idle
CPU/cpu3/user
CPU/cpu3/kern
CPU/cpu3/wait
CPU/cpu3/idle

```

The wildcard stanza has been modified to display the CPU usage of all four of the CPUs of the system. The following command was issued to produce the graphical display shown in Figure 43-26:

```
3dmon -i1 -cmyconfig
```

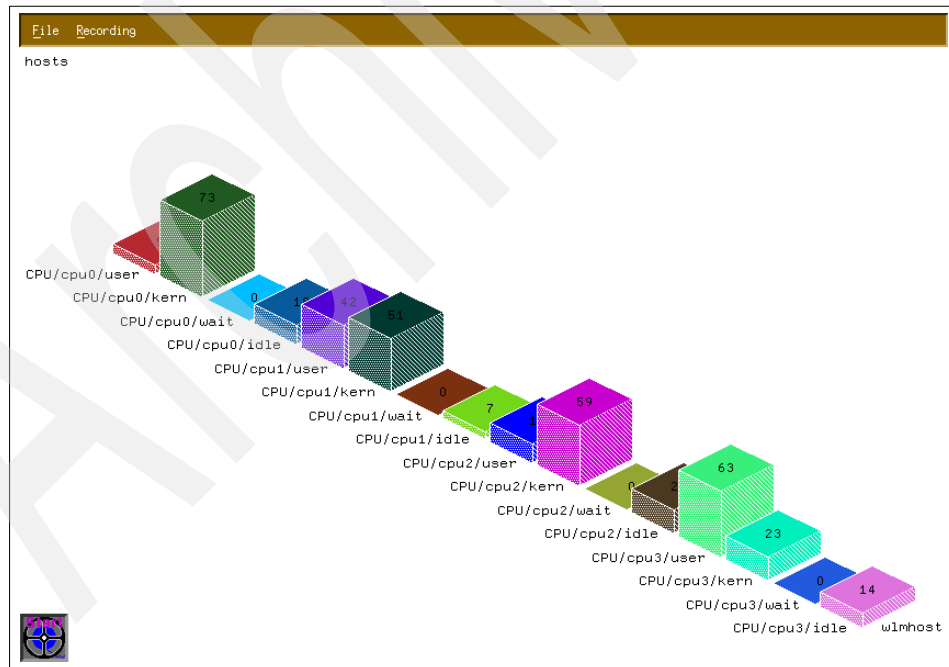


Figure 43-26 CPU statistics displayed by 3dmon after modifying 3dmon.cf

Figure 43-27 shows a typical example of a multiple-host graphical display showing the disk statistics.

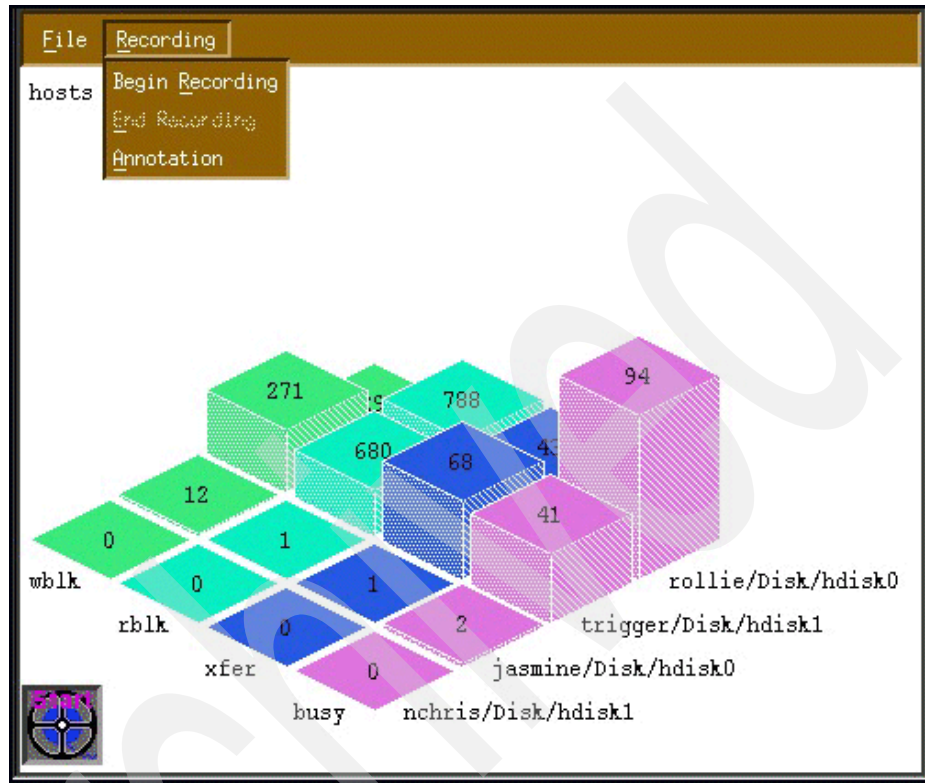


Figure 43-27 3dmon graph showing disk activity for multiple hosts

The configuration file for this can be seen in Example 43-3.

Example 43-3 3dmon.cf showing disk configuration for multiple hosts

```
# @(#)01 1.7 src/perf/perfmgr/usr/samples/perfmgr/3dmon.cf, perfmgr,
43perf30
0, 0101A_43perf300 12/3/96 06:52:31

#
# COMPONENT_NAME: (PERFMGR) - Performance Manager
#
# FUNCTIONS: Configuration file
#
# ORIGINS: 30
#
# (C) COPYRIGHT International Business Machines Corp. 1992, 1993
# All Rights Reserved
```

```
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
wildcard:  disk    hosts  # disks on remote hosts
Disk/*/busy
Disk/*/xfer
Disk/*/rblk
Disk/*/wblk
```

43.4 jazizo

The **jazizo** program is used to analyze system statistics over a long period of time. The **jazizo** program uses the **xmtrend** daemon to collect data. It can be configured to only show areas of interest in concise graphical form or table form. The output can be generated for specific time periods.

The **jazizo** program resides in `/usr/bin` and is part of the `perfmgr.analysis.jazizo` fileset, which is installable from the Performance Toolbox Version 3 for AIX media. The `perfmgr.common` and `perfagent.server` filesets are prerequisites for `perfmgr.analysis.jazizo`. They are also installable from the PTX media.

43.4.1 Syntax of xmtrend

The syntax of the **xmtrend** command is:

```
xmtrend {-f infile} {-d recording_dir} {-n recording_name} {-t trace_level}
```

Flags

- | | |
|--------------------------|---|
| -f infile | The name of the configuration file that is used by xmtrend to determine which parameters to monitor. The default file name is <code>/etc/perf/xmtrend.cf</code> . |
| -d recording_dir | This flag specifies the recording file output directory. The default directory is <code>/etc/perf</code> . |
| -n recording_name | This flag specifies the name of the recording file. By default, xmtrend creates a recording file named <code>xmtrend.date</code> . If <code>-n myfile</code> is specified, the recording files will be named <code>myfile.date</code> , where <code>date</code> is the system date at file creation time in the format <code>yymmdd</code> . |
| -t trace_level | Trace level can be any whole number from one to nine. The higher the value of the trace level, the greater the amount of trace information supplied. By default the log |

file is created in /etc/perf and is named xmtrend.log1, and xmtrend.log2 when xmtrend.log1 is full. If -d mydir is specified, then the log file will be created in mydir, and if -n myfile is specified, then the logfile is named as myfile.log1 and myfile.log2.

43.4.2 Syntax of jazizo

The syntax of the **jazizo** command is:

```
jazizo -r <recording> -c <configuration file>
```

Flags

- | | |
|--------------------------------------|---|
| -r <recording> | This is the name of the recording file. A directory name can be specified in which recording files will be located. |
| -c <configuration file> | This is the jazizo configuration file. A sample can be found in /usr/lpp/perfmgr/jazizo.cf |

The default configuration file can be copied into a working directory and modified as required. It is recommended that instead of modifying the original file, make and modify a copy and move it to the required directory.

43.4.3 Information about measurement and sampling

The **jazizo** program is dependant on the **xmtrend** daemon running on the system. To start the **xmtrend** daemon, perform the following functions:

Determine where the xmtrend.cf file is. This file determines which statistics or metrics will be monitored. The sample xmtrend configuration file can be found in the /usr/lpp/perfagent directory. It is recommended that the original is left untouched. Make a copy of the file in the working directory, for example /etc/perf, and make the required changes to that file.

Run the **xmtrend** daemon in background and use the **nohup** command as follows:

```
/etc/perf> nohup xmtrend -f /etc/perf/xmtrend.cf -d /etc/perf -n my_stats
```

The **xmtrend** daemon can also be started by placing an entry in the /etc/inittab file. This overcomes the problem of restarting the daemon each time the system is rebooted.

To confirm that the **xmtrend** daemon is running, use the **ps** command as in Example 43-4 on page 920.

Example 43-4 Checking that the xmtrend daemon is running

```
# ps -ef |grep xmtrend
  root 17544 17834  1 15:51:40 pts/1  0:26 xmtrend -f /etc/perf/xmtrend.cf
d /etc/perf -n stuart
  root 21414 14758  1 17:20:49 pts/2  0:00 grep xmtrend
```

If the daemon starts successfully, a reporting file will be created in the working directory. In this example, the file name will be `my_stats.date`, where `date` is the system date. The size of this file is dependant on the number of metrics that are being measured. A metric can be defined as a measurement of a resource such as CPU idle time.

If the daemon fails to start, it may be due to the `xmtrend` daemon being incorrectly stopped on a previous occasion. To correct this problem, refer to the Workload Manager (WLM) section in “xmtrend” on page 874.

To start the `jazizo` program, type the command:

```
jazizo -c /etc/perf/jazizo.cf -n my_stats.010531.
```

Exploring the jazizo windows

If the command had been issued without any flags, the `jazizo` program would have searched for a configuration file in order to determine which metrics are to be displayed. At this point it is important to remember that the `xmtrend` daemon gathers the data, while the `jazizo` program displays the results.

Figure 43-28 on page 921 shows the `jazizo` program issued without any flags.



Figure 43-28 The jazizo opening window

This is the first screen displayed by jazizo. Click **File** to open the menu in Figure 43-29.

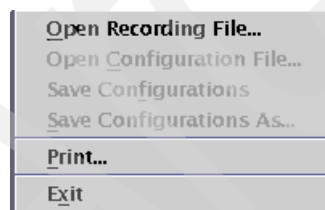


Figure 43-29 The File tab down menu

Click **Open Recording File** to select the recording file from which the results are to be obtained. Figure 43-30 on page 922 shows the list of files in the **jazizo** directory.

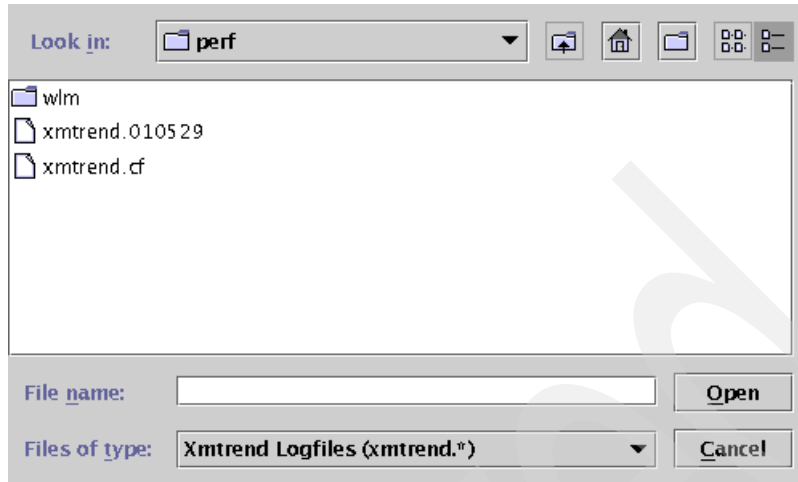


Figure 43-30 The Open Recording File window in jazizo

In this example, the directory name is /etc/perf, and xmtrend.010529 is the only file that contains monitored data. (All files that contain monitoring information have a six-digit date suffix based on creation date.) Select the recording file from which the results are required, and click **Open**.

The left-hand pane of the **Metric Selection** window shows a list of metrics that can be displayed. In Figure 43-31, the values for the CPU idle, CPU kernel, and CPU user metrics have been selected. You can select multiple values using the Ctrl or Shift keys.

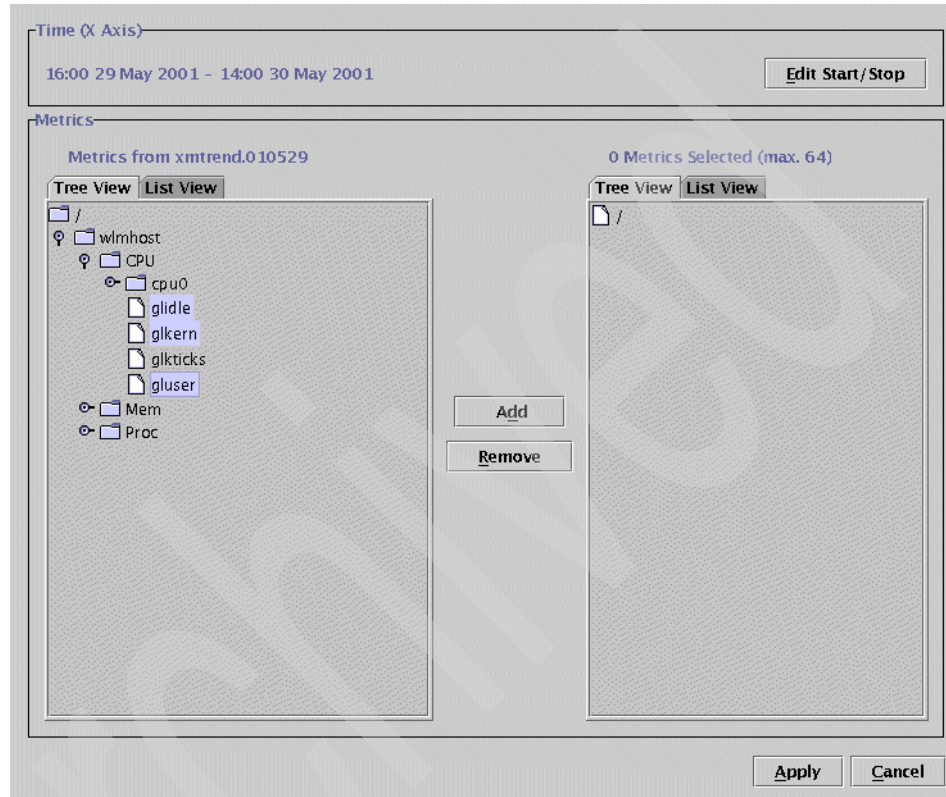


Figure 43-31 Metric Selection window

To display the metrics, they must be added to the right-hand pane. Click **Add** to move the selected metrics. In the same way, an incorrectly selected metric can be removed from the right-hand pane by selecting the metric and clicking **Remove**. Figure 43-32 on page 924 shows the result.

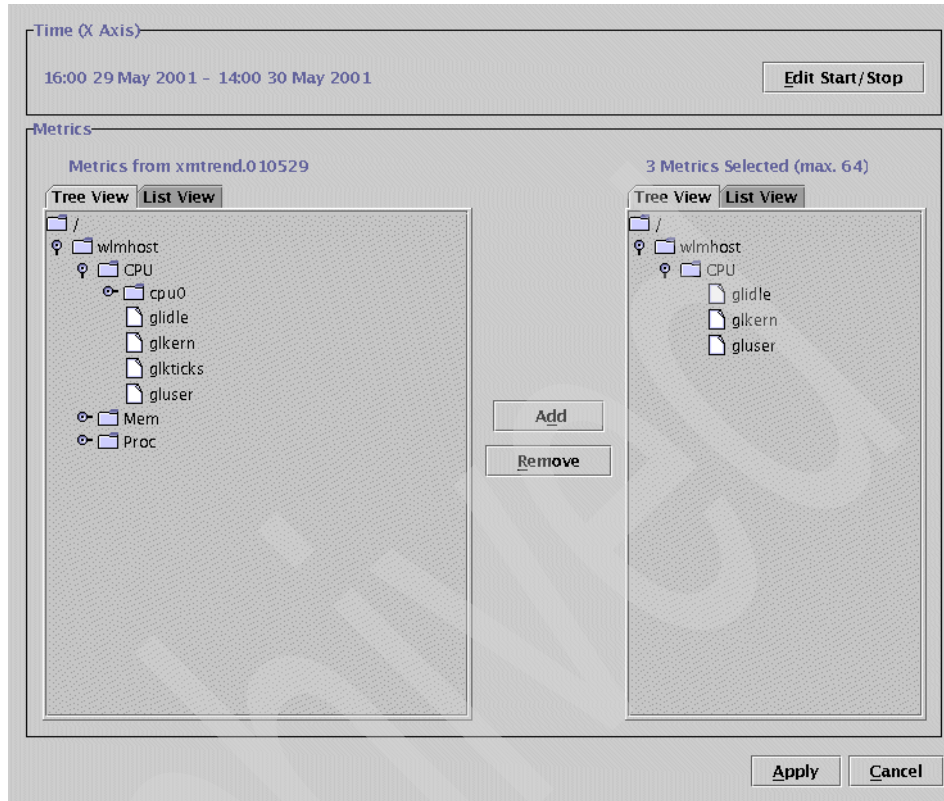


Figure 43-32 The Metric Selection window showing metric selections

The file containing the data on the metrics can have data spanning several months, up to a year, so date and time selections are available to crop the view to display only a required period. Click **Edit Start/Stop** to open the window in Figure 43-33 on page 925, where you can select the Start Hour and Stop Hour for the period to be displayed.

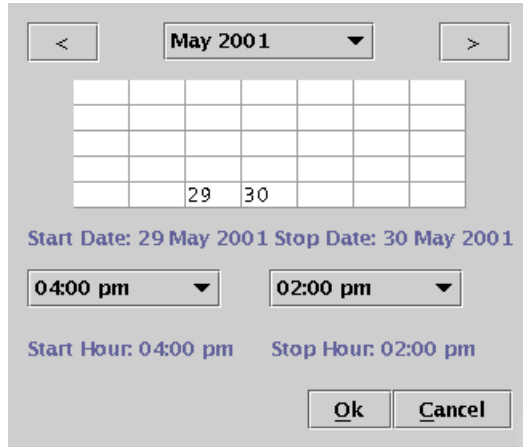


Figure 43-33 The Time Selection window

Click **Start Hour** or **Stop Hour** to select the respective times from the menu shown in Figure 43-34.

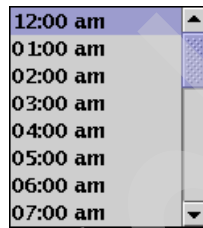


Figure 43-34 The Stop Hour and Start Hour tab down menus

The desired month can also be selected, as shown in Figure 43-35 on page 926.

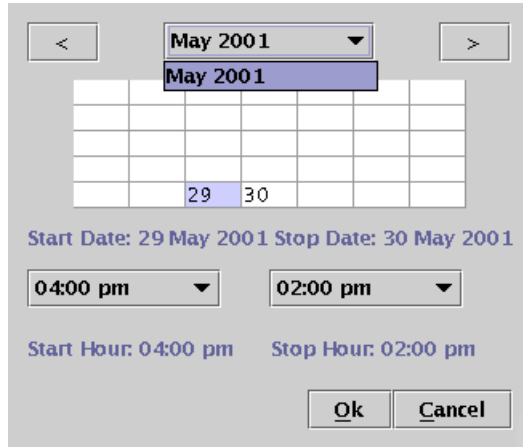


Figure 43-35 Adjusting the month in the jazizo Time Selection window

The day of the month can also be changed. Note that in Figure 43-36, only two of the blocks in the calendar have a day of the month number; this particular recording file contains statistics only for these days. Click one of the days in the calendar to open the **Set Start Date** and **Set End Date** menu and set the stop day for the monitoring period. Perform the same operation for the start time. There is no specific order in which the date and time is set.

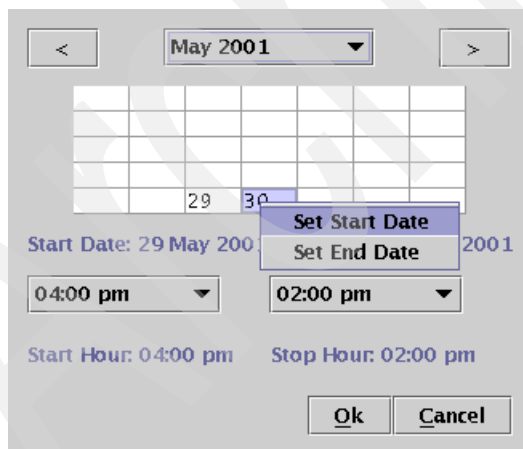


Figure 43-36 Adjusting days in the jazizo Time Selection window

When the time and date selections have been made, click **Ok** to close the **Time Selection** window. Click **Apply** in the **Metric Selection** window. The **jazizo** program now displays the selected metrics over the selected monitoring period, as shown in Figure 43-37 on page 927. The vertical axis of the graph is shown in

graduations of 10, and is in percent because this graphic is displaying CPU percentage statistics. The horizontal axis has a time graduation over the selected monitor period.

Each of the selected metrics in this example is represented by a colored line graph. At the bottom of the window the metrics are listed with the appropriate colored selection blocks. These selection blocks are the same color as the line of the graph. The selection block is used to select the specific metric on the graph. The name of the particular metric is followed by its range minimum and maximum in parentheses. (In the interest of clarity, the background color has been changed from black to gray.)

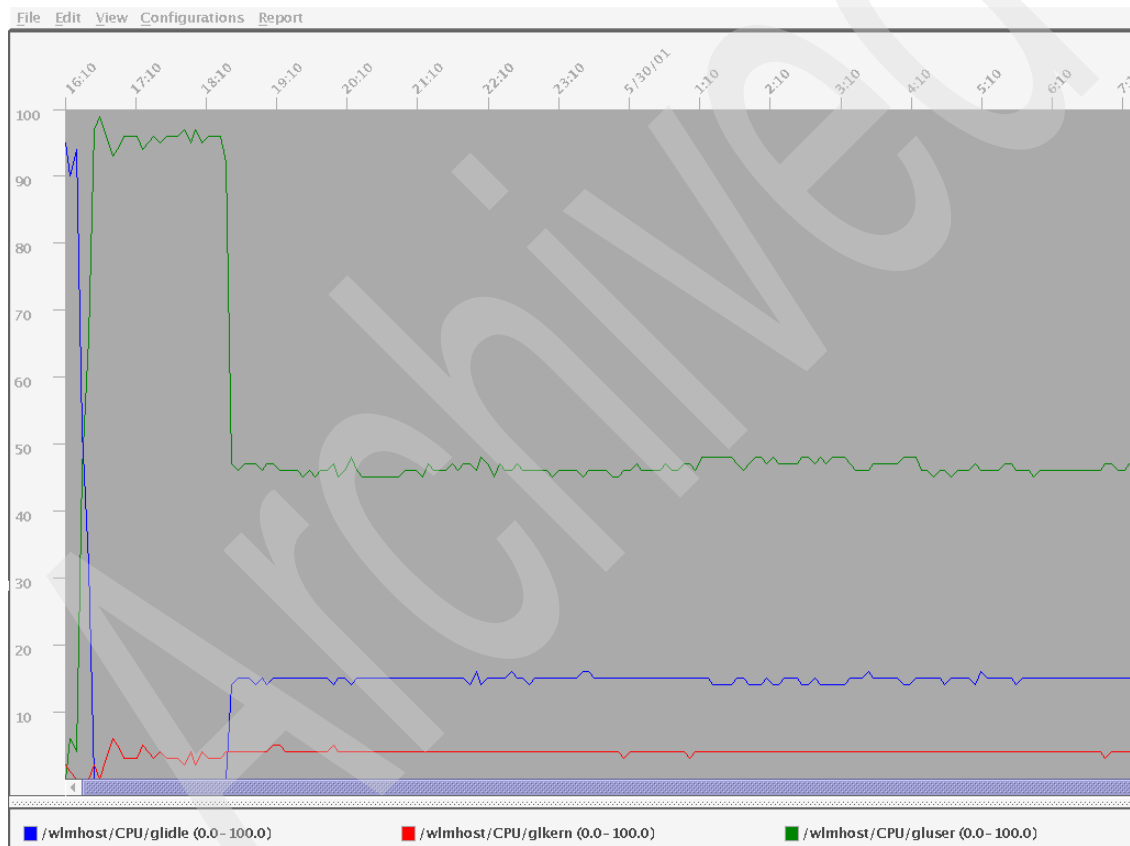


Figure 43-37 The jazizo window

From the **Edit** menu, select the **Metric Selection** option shown in Figure 43-38 to open the **Metric Selection** window.

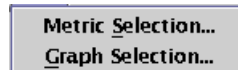


Figure 43-38 The jazizo Edit tab down menu

Choosing **Graph Selection** instead opens the window in Figure 43-39. Several options are available here, such as standard deviation and the trend line option.

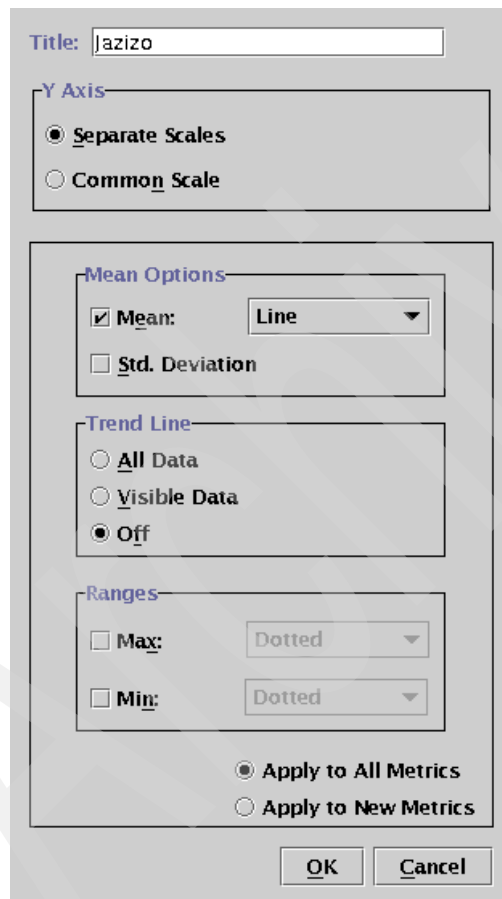


Figure 43-39 The Graph Selection window of the jazizo program

Figure 43-40 on page 929 shows the **jazizo** graphical output within which the trend lines have been added. This option is particularly useful when comparing the output for one month with another so overall performance for the measured

metric can be observed quickly. The trend lines are the same color in the graph as the metric with which they are associated.

Two Trend Line options are available. The first option, **All Data**, shows the trend for the entire measurement period, as shown in Figure 43-40. The second trend option, **Visible Data**, shows the trend for only the section that is currently visible in the display window. Alternately, the Trend Line option can be switched off using the **Off** radio button. Only one of the options can be selected at a time.

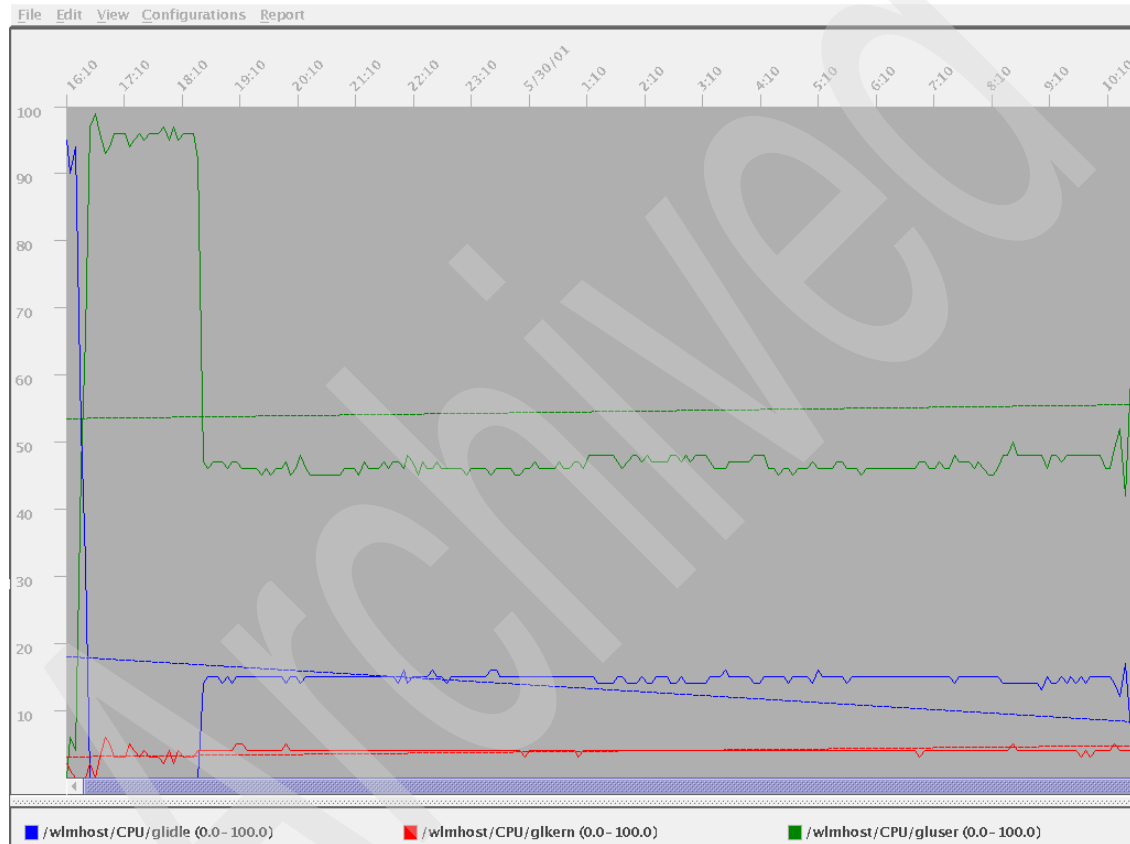


Figure 43-40 The trend of the metric can be displayed by jazizo

From the main window, select **View** to access the options in Figure 43-41 on page 930.

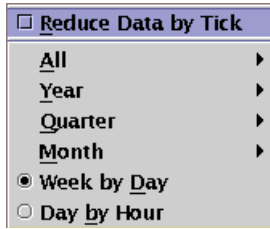


Figure 43-41 The View tab down menu

When the **Reduce Data by Tick** box in the tab down menu is checked, the output will show less data. For a full display showing all of the time intervals, ensure that this box is not checked. (It is checked by default.) The other options in the tab down menu determine the displayed time graduation. **Day by Hour** is the default.

Selecting **Report** in the main window displays the menu shown in Figure 43-42.

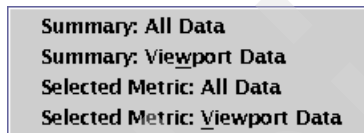


Figure 43-42 The Report tab down menu

These options supply statistical (non-graphical) information about the metrics. Click **Summary: All Data** for a table with the statistics for all of the currently displayed metrics. If specific metric options are required, choose either **Selected Metric: All Data** or **Selected Metric: Viewport Data** to display statistical data for the specific metric.

Timestamp	Mean	Max	Min	Std Dev
5/29/01 4:10 PM	95	95	0	0
5/29/01 4:14 PM	90	99	50	10
5/29/01 4:20 PM	94	99	72	7
5/29/01 4:25 PM	52	76	24	9
5/29/01 4:30 PM	32	52	0	24
5/29/01 4:34 PM	0	1	0	0
5/29/01 4:40 PM	0	0	0	0
5/29/01 4:45 PM	0	0	0	0
5/29/01 4:50 PM	0	0	0	0

Figure 43-43 Tabular statistical output that can be obtained from jazizo

The display is in the tabular format seen in Figure 43-43 with these headings:

- Timestamp The sample time interval; here it is five minutes between samples.
- Mean The mean value monitored over the time interval.
- Max The maximum value during the time interval.

Min The minimum value over the time period.
Std Dev The standard deviation during the time interval.

The output can be printed either to file or to a printer by selecting **Print**. Note that the table in Figure 43-43 on page 930 is an extract from the full table listing and hence does not show the **Print** or **Close** screen buttons that appear at the bottom of the table view.

To close the **jaz i zo** windows, open the jazizo main window's **File** menu, shown in Figure 43-44.

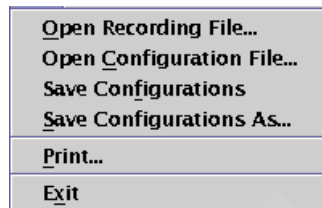


Figure 43-44 The File tab down menu when closing jazizo

If any configurations have been changed, they can be saved here using the **Save Configurations** or **Save Configurations As** options. To exit from the program, select the **Exit** option.

Archived



Part 9

Appendixes

Archiving

Archived

Source code examples

This appendix contains source code that was used to create the examples for these sections of this book:

- ▶ The `perfstat_dude.c` program was used in 41.1, “Perfstat API” on page 786.
- ▶ The programs `spmi_dude.c`, `spmi_data.c`, `spmi_file.c`, and `spmi_traverse.c` were used in 41.2, “System Performance Measurement Interface” on page 805.
- ▶ The `dudestat.c` program was used in 41.5, “Miscellaneous performance monitoring subroutines” on page 842.
- ▶ The `cwhet.c` program was used in Chapter 19.2.3, “Examples for `gprof`” on page 302 and 19.4.2, “Examples for `prof`” on page 322.

Unlike the examples in the chapters, the source code examples in this appendix do not have line numbers, making it easier to copy and paste from the online version of this book.

perfstat_dump_all.c

Example A-1 shows how to combine all examples from 41.1.2, “Subroutines” on page 787 to access data provided by AIX 5.2 Perfstat API subroutines. Note that the error checking and memory management in this example must be enhanced for a production-type program.

Example: A-1 AIX 5.2 Perfstat API complete example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>

4  cpu()
5  {
6      perfstat_id_t   name;
7      perfstat_cpu_t *ub;
8      int             ncpu,i;

9      ncpu = perfstat_cpu (NULL,NULL,sizeof(perfstat_cpu_t),0);
10     ub = malloc(sizeof(perfstat_cpu_t)*ncpu);

11     strcpy(name.name,"");

12     if (perfstat_cpu(&name,ub,sizeof(perfstat_cpu_t),ncpu) >= 0)
13         for (i = 0; i < ncpu; i++) {
14             printf("name      : %s\n",  ub[i].name);
15             printf("\tuser      : %llu\n",  ub[i].user);
16             printf("\tsys       : %llu\n",  ub[i].sys);
17             printf("\tidle      : %llu\n",  ub[i].idle);
18             printf("\twait      : %llu\n",  ub[i].wait);
19             printf("\tpswhch   : %llu\n",  ub[i].pswhch);
20             printf("\tsyscall  : %llu\n",  ub[i].syscall);
21             printf("\tsysread  : %llu\n",  ub[i].sysread);
22             printf("\tsyswrite : %llu\n",  ub[i].syswrite);
23             printf("\tsysfork  : %llu\n",  ub[i].sysfork);
24             printf("\tsysexec  : %llu\n",  ub[i].sysexec);
25             printf("\treadch   : %llu\n",  ub[i].readch);
26             printf("\twritech  : %llu\n",  ub[i].writech);
27         }
28 }

29 cpu_total()
30 {
31     perfstat_cpu_total_t   ub;

32     if (perfstat_cpu_total((perfstat_id_t*)NULL, &ub,
33         sizeof(perfstat_cpu_total_t), 1) >= 0) {
34         printf("ncpus      : %d\n", ub.ncpus);
```

```

34     printf("ncpus_cfg   : %d\n", ub.ncpus_cfg);
35     printf("description : %s\n", ub.description);
36     printf("processorHZ  : %llu\n", ub.processorHZ);
37     printf("user        : %llu\n", ub.user);
38     printf("sys         : %llu\n", ub.sys);
39     printf("idle        : %llu\n", ub.idle);
40     printf("wait         : %llu\n", ub.wait);
41     printf("pswitch      : %llu\n", ub.pswitch);
42     printf("syscall     : %llu\n", ub.syscall);
43     printf("sysread     : %llu\n", ub.sysread);
44     printf("syswrite    : %llu\n", ub.syswrite);
45     printf("sysfork     : %llu\n", ub.sysfork);
46     printf("sysexec    : %llu\n", ub.sysexec);
47     printf("readch     : %llu\n", ub.readch);
48     printf("writech    : %llu\n", ub.writech);
49     printf("devintrs   : %llu\n", ub.devintrs);
50     printf("softintrs  : %llu\n", ub.softintrs);
51     printf("lbolt      : %ld\n", ub.lbolt);
52     printf("loadavg T0 : %llu\n", ub.loadavg[0]);
53     printf("loadavg T-5 : %llu\n", ub.loadavg[1]);
54     printf("loadavg T-15: %llu\n", ub.loadavg[2]);
55     printf("runque     : %llu\n", ub.runque);
56     printf("swpque     : %llu\n", ub.swpque);
57 }
58 }

59 disk()
60 {
61     perfstat_id_t   name;
62     perfstat_disk_t *ub;
63     int             ndisk,i;

64     ndisk = perfstat_disk (NULL,NULL,sizeof(perfstat_disk_t),0);
65     ub = malloc(sizeof(perfstat_disk_t)*ndisk);

66     strcpy(name.name, "");

67     if (perfstat_disk (&name,ub,sizeof(perfstat_disk_t),ndisk) >= 0)
68         for (i = 0; i < ndisk; i++) {
69             printf("name           : %s\n", ub[i].name);
70             printf("\tdescription: %s\n", ub[i].description);
71             printf("\tvgname      : %s\n", ub[i].vgname);
72             printf("\tsize       : %llu\n", ub[i].size);
73             printf("\tfree       : %llu\n", ub[i].free);
74             printf("\tbsize     : %llu\n", ub[i].bsize);
75             printf("\txrate     : %llu\n", ub[i].xrate);
76             printf("\txfers     : %llu\n", ub[i].xfers);
77             printf("\twblks     : %llu\n", ub[i].wblks);
78             printf("\trblks     : %llu\n", ub[i].rblks);

```

```

79         printf("\tqdepth      : %llu\n", ub[i].qdepth);
80         printf("\ttime        : %llu\n", ub[i].time);
81     }
82 }

83 disk_total()
84 {
85     perfstat_disk_total_t  ub;

86     if (perfstat_disk_total ((perfstat_id_t*)NULL, &ub,
sizeof(perfstat_disk_total_t), 1) >= 0) {
87         printf("number: %d\n", ub.number);
88         printf("size  : %llu\n", ub.size);
89         printf("free  : %llu\n", ub.free);
90         printf("xrate : %llu\n", ub.xrate);
91         printf("xfers : %llu\n", ub.xfers);
92         printf("wblks : %llu\n", ub.wblks);
93         printf("rblks : %llu\n", ub.rblks);
94         printf("time  : %llu\n", ub.time);
95     }
96 }

97 memory_total()
98 {
99     perfstat_memory_total_t ub;

100    if (perfstat_memory_total
((perfstat_id_t*)NULL,&ub,sizeof(perfstat_memory_total_t),1) >= 0) {
101        printf("virt_total: %llu\n", ub.virt_total);
102        printf("real_total: %llu\n", ub.real_total);
103        printf("real_free : %llu\n", ub.real_free);
104        printf("real_inuse: %llu\n", ub.real_inuse);
105        printf("pgbad   : %llu\n", ub.pgbad);
106        printf("pgexct  : %llu\n", ub.pgexct);
107        printf("pgins   : %llu\n", ub.pgins);
108        printf("pgouts  : %llu\n", ub.pgouts);
109        printf("pgspins : %llu\n", ub.pgspins);
110        printf("pgspouts : %llu\n", ub.pgspouts);
111        printf("scans   : %llu\n", ub.scans);
112        printf("cycles  : %llu\n", ub.cycles);
113        printf("pgsteals : %llu\n", ub.pgsteals);
114        printf("numperm  : %llu\n", ub.numperm);
115        printf("pgsp_total: %llu\n", ub.pgsp_total);
116        printf("pgsp_free : %llu\n", ub.pgsp_free);
117        printf("pgsp_rsvd : %llu\n", ub.pgsp_rsvd);
118    }
119}

120netinterface()

```



```

121 {
122     perfstat_id_t          name;
123     perfstat_netinterface_t *ub;
124     int                    nnetinterface,i;

125     nnetinterface = perfstat_netinterface (NULL,NULL,
sizeof(perfstat_netinterface_t), 0);
126     ub = malloc(sizeof(perfstat_netinterface_t)*nnetinterface);

127     strcpy(name.name,"");

128     if (perfstat_netinterface (&name,ub, sizeof(perfstat_netinterface_t),
nnetinterface) >= 0)
129         for (i = 0; i < nnetinterface; i++) {
130             printf("name      : %s\n",    ub[i].name);
131             printf("\tdescription: %s\n",  ub[i].description);
132             printf("\ttype      : %u\n",    ub[i].type);
133             printf("\tmtu      : %llu\n",   ub[i].mtu);
134             printf("\tipackets  : %llu\n",   ub[i].ipackets);
135             printf("\tibytes   : %llu\n",   ub[i].ibytes);
136             printf("\tierrors  : %llu\n",   ub[i].ierrors);
137             printf("\topackets : %llu\n",   ub[i].opackets);
138             printf("\tobytes  : %llu\n",   ub[i].obytes);
139             printf("\toerrors  : %llu\n",   ub[i].oerrors);
140             printf("\tcollisions : %llu\n", ub[i].collisions);
141         }
142 }

143 netinterface_total()
144 {
145     perfstat_netinterface_total_t  ub;

146     if (perfstat_netinterface_total ((perfstat_id_t*)NULL,&ub,
sizeof(perfstat_netinterface_total_t),1) >= 0) {
147         printf("number      : %d\n", ub.number);
148         printf("ipackets   : %llu\n", ub.ipackets);
149         printf("ibytes    : %llu\n", ub.ibytes);
150         printf("ierrors   : %llu\n", ub.ierrors);
151         printf("opackets  : %llu\n", ub.opackets);
152         printf("obytes   : %llu\n", ub.obytes);
153         printf("oerrors  : %llu\n", ub.oerrors);
154         printf("collisions: %llu\n", ub.collisions);
155     }
156 }

157 main()
158 {
159     cpu_total();
160     cpu();

```

```

161  disk_total();
162  disk();
163  memory_total();
164  netinterface_total();
165  netinterface();
166 }

```

perfstat_dude.c

The perfstat_dude.c program in Example A-2 makes one reading of a selected number of statistics, then waits for a specified amount of time before it takes the other reading.

Example: A-2 perfstat_dude.c program

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/var.h>
#include <libperfstat.h>

#defineNCPU1024
#defineNDISK1024
#defineNNETWORK1024

static intncpu = NCPU;
static intndisk = NDISK;
static intnnetwork = NNETWORK;

cpu_t(int t)
{
    perfstat_id_tname;
    perfstat_cpu_tub[NCPU];
    int i, rc;
    static u_longlong_ttime[NCPU];
    static u_longlong_tuser[NCPU];
    static u_longlong_tsys[NCPU];
    static u_longlong_tidle[NCPU];
    static u_longlong_twait[NCPU];
    static u_longlong_tsysfork[NCPU];
    static u_longlong_tsyscall[NCPU];
    static u_longlong_tpswitch[NCPU];

    strcpy(name.name, "");

    if (t) {
        if ((rc = perfstat_cpu (&name,ub,sizeof(perfstat_cpu_t),NCPU)) >= 0) {

```

```

printf("%6.6s %6.6s %6.6s %6.6s %3.3s %3.3s %3.3s %3.3s\n",
       "cpu", "fk", "sy", "cs", " us", " sy", "id", "wa");
for (i=0;i<rc;i++) {
    ttime[i] =
(ub[i].user-user[i])+(ub[i].sys-sys[i])+(ub[i].idle-idle[i])+(ub[i].wait-wait[i
]);

    printf("%6.6s ", ub[i].name);
    printf("%6lld ", ub[i].sysfork-sysfork[i]);
    printf("%6lld ", ub[i].syscall-syscall[i]);
    printf("%6lld ", ub[i].pswitch-pswitch[i]);
    printf("%3lld ", (ub[i].user-user[i])*100/ttime[i]);
    printf("%3lld ", (ub[i].sys-sys[i])*100/ttime[i]);
    printf("%3lld ", (ub[i].idle-idle[i])*100/ttime[i]);
    printf("%3lld ", (ub[i].wait-wait[i])*100/ttime[i]);
    printf("\n");
}
printf("\n");
} else {
    perror("perfstat_cpu 1");
}
} else {
    if ((rc = perfstat_cpu (&name,ub,sizeof(perfstat_cpu_t),NCPU) >= 0) {
        for (i=0;i<rc;i++) {
            user[i] = ub[i].user;
            sys[i] = ub[i].sys;
            idle[i] = ub[i].idle;
            wait[i] = ub[i].wait;
            sysfork[i] = ub[i].sysfork;
            syscall[i] = ub[i].syscall;
            pswitch[i] = ub[i].pswitch;
        }
    } else {
        perror("perfstat_cpu 0");
    }
}
}
}
cpu_total_t(int t)
{
    perfstat_cpu_total_tub;
    static int ncpus;
    static u_longlong_ttime;
    static u_longlong_trunq;
    static u_longlong_tswpque;
    static u_longlong_tdevintrs;
    static u_longlong_tsoftintrs;
    static u_longlong_tsysfork;
    static u_longlong_tsyscall;
    static u_longlong_tpswitch;
    static u_longlong_tuser;

```

```

static u_longlong_tsys;
static u_longlong_tidle;
static u_longlong_twait;

if (t) {
    if (perfstat_cpu_total
        ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_cpu_total_t),1) >= 0) {
        ttime = (ub.user-user)+(ub.sys-sys)+(ub.idle-idle)+(ub.wait-wait);
        printf("Que      Faults      Cpu\n");
        printf("%3.3s %3.3s %6.6s %6.6s %6.6s %6.6s %3.3s %3.3s %3.3s
%3.3s\n",
            "rq","sq","fk","in","sy","cs","us","sy","id","wa");
        printf("%31ld ", ub.runque-runque);
        printf("%31ld ", ub.swpque-swpque);
        printf("%61ld ", ub.sysfork-sysfork);
        printf("%61ld ", (ub.devintrs+ub.softintrs)-(devintrs+softintrs));
        printf("%61ld ", ub.syscall-syscall);
        printf("%61ld ", ub.pswitch-pswitch);
        printf("%31ld ", (ub.user-user)*100/ttime);
        printf("%31ld ", (ub.sys-sys)*100/ttime);
        printf("%31ld ", (ub.idle-idle)*100/ttime);
        printf("%31ld ", (ub.wait-wait)*100/ttime);
        printf("\n\n");
    } else {
        perror("perfstat_cpu_total 1");
    }
} else {
    if (perfstat_cpu_total
        ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_cpu_total_t),1) >= 0) {
        ncpus = ub.ncpus;
        runque = ub.runque;
        swpque = ub.swpque;
        sysfork = ub.sysfork;
        syscall = ub.syscall;
        devintrs = ub.devintrs;
        softintrs = ub.softintrs;
        pswitch = ub.pswitch;
        user = ub.user;
        sys = ub.sys;
        idle = ub.idle;
        wait = ub.wait;
    } else {
        perror("perfstat_cpu_total 0");
    }
}
}

disk_t(int t)

```

```

{
    perfstat_id_tname;
    perfstat_disk_tub[NDISK];
    int i,rc;
    static u_longlong_tqdepth[NDISK];
    static u_longlong_ttime[NDISK];
    static u_longlong_txrate[NDISK];
    static u_longlong_txfers[NDISK];
    static u_longlong_trblks[NDISK];
    static u_longlong_twblks[NDISK];

    strcpy(name.name, "");

    if (t) {
        if ((rc = perfstat_disk (&name,ub,sizeof(perfstat_disk_t),NDISK)) >= 0)
        {
            printf("%6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s\n",
                "disk","vg","qd","busy","KB/s","xfers","KBrd","KBwr");
            for (i=0;i<rc;i++) {
                printf("%6s ", ub[i].name);
                printf("%6s ", ub[i].vgname);
                printf("%6lld ", ub[i].qdepth-qdepth[i]);
                printf("%6lld ", ub[i].time-time[i]);
                printf("%6lld ", ub[i].xrate-xrate[i]);
                printf("%6lld ", ub[i].xfers-xfers[i]);
                printf("%6lld ", ub[i].rblks-rblks[i]);
                printf("%6lld ", ub[i].wblks-wblks[i]);
                printf("\n");
            }
            printf("\n");
        } else {
            perror("perfstat_disk 1");
        }
    } else {
        if ((rc = perfstat_disk (&name,ub,sizeof(perfstat_disk_t),NDISK)) >= 0)
        {
            for (i=0;i<rc;i++) {
                qdepth[i] = ub[i].qdepth;
                time[i] = ub[i].time;
                xrate[i] = ub[i].xrate;
                xfers[i] = ub[i].xfers;
                rblks[i] = ub[i].rblks;
                wblks[i] = ub[i].wblks;
            }
        } else {
            perror("perfstat_disk 0");
        }
    }
}

```

```

disk_total_t(int t)
{
    perfstat_disk_total_tub;
    static u_longlong_ttime;
    static u_longlong_txrate;
    static u_longlong_txfers;
    static u_longlong_trblks;
    static u_longlong_twblks;

    if (t) {
        if (perfstat_disk_total
            ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_disk_total_t),1) >= 0) {
            printf("%6.6s %6.6s %6.6s %6.6s %6.6s\n",
                "busy", " KB/s", "xfers", "KBrd", "KBwr");
            printf("%6lld ", ub.time-time);
            printf("%6lld ", ub.xrate-xrate);
            printf("%6lld ", ub.xfers-xfers);
            printf("%6lld ", ub.rblks-rblks);
            printf("%6lld ", ub.wblks-wblks);
            printf("\n\n");
        } else {
            perror("perfstat_disk_total 1");
        }
    } else {
        if (perfstat_disk_total
            ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_disk_total_t),1) >= 0) {
            time = ub.time;
            xrate = ub.xrate;
            xfers = ub.xfers;
            rblks = ub.rblks;
            wblks = ub.wblks;
        } else {
            perror("perfstat_disk_total 0");
        }
    }
}

memory_total_t(int t)
{
    perfstat_memory_total_tub;
    static u_longlong_treal_free;
    static u_longlong_treal_inuse;
    static u_longlong_tpgsp_free;
    static u_longlong_tpgspins;
    static u_longlong_tpgspouts;
    static u_longlong_tpgins;
    static u_longlong_tpgouts;
    static u_longlong_tpgexct;
}

```

```

static u_longlong_tpgsteals;
static u_longlong_tscans;
static u_longlong_tnumperm;

if (t) {
    if (perfstat_memory_total
((perfstat_id_t*)NULL,&ub,sizeof(perfstat_memory_total_t),1) >= 0) {
        printf("Real memory          Paging space Virtual\n");
        printf("%6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s\n",
%6.6s\n",
"free","use","free","psi","pso","pi","po","fault","fr","sr","num");
        printf("%6lld ", ub.real_free);
        printf("%6lld ", ub.real_inuse);
        printf("%6lld ", ub.pgsp_free);
        printf("%6lld ", ub.pgspins);
        printf("%6lld ", ub.pgspouts);
        printf("%6lld ", ub.pgins);
        printf("%6lld ", ub.pgouts);
        printf("%6lld ", ub.pgexct);
        printf("%6lld ", ub.pgsteals);
        printf("%6lld ", ub.scans);
        printf("%6lld ", ub.numperm);
        printf("\n\n");
    } else {
        perror("perfstat_memory_total 1");
    }
} else {
    if (perfstat_memory_total
((perfstat_id_t*)NULL,&ub,sizeof(perfstat_memory_total_t),1) >= 0) {
        real_free = ub.real_free;
        real_inuse = ub.real_inuse;
        pgsp_free = ub.pgsp_free;
        pgspins = ub.pgspins;
        pgspouts = ub.pgspouts;
        pgins = ub.pgins;
        pgouts = ub.pgouts;
        pgexct = ub.pgexct;
        pgsteals = ub.pgsteals;
        scans = ub.scans;
        numperm = ub.numperm;
    } else {
        perror("perfstat_memory_total 1");
    }
}
}

netinterface_t(int t)
{

```

```

perfstat_id_tname;
perfstat_netinterface_tub[NDISK];
int i,rc;
static u_longlong_tipackets[NDISK];
static u_longlong_tibytes[NDISK];
static u_longlong_terrors[NDISK];
static u_longlong_topackets[NDISK];
static u_longlong_tobytes[NDISK];
static u_longlong_toerrors[NDISK];
static u_longlong_tcollisions[NDISK];

strcpy(name.name, "");

if (t) {
    if ((rc = perfstat_netinterface
(&name,ub,sizeof(perfstat_netinterface_t),NNETWORK) >= 0) {
        printf("%7.7s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s\n",
            "network", "mtu", "ipack", "ibyte", "ierr", "opack", " obyte", "
oerr", "coll");
        for (i=0;i<rc;i++) {
            printf("%7s ", ub[i].name);
            printf("%6lld ", ub[i].mtu);
            printf("%6lld ", ub[i].ipackets-ipackets[i]);
            printf("%6lld ", ub[i].ibytes-ibytes[i]);
            printf("%6lld ", ub[i].ierrors-ierrors[i]);
            printf("%6lld ", ub[i].opackets-opackets[i]);
            printf("%6lld ", ub[i].obytes-obytes[i]);
            printf("%6lld ", ub[i].oerrors-oerrors[i]);
            printf("%6lld ", ub[i].collisions-collisions[i]);
            printf("\n");
        }
        printf("\n");
    } else {
        perror("perfstat_netinterface 1");
    }
} else {
    if ((rc = perfstat_netinterface
(&name,ub,sizeof(perfstat_netinterface_t),NNETWORK) >= 0) {
        for (i=0;i<rc;i++) {
            ipackets[i] = ub[i].ipackets;
            ibytes[i] = ub[i].ibytes;
            ierrors[i] = ub[i].ierrors;
            opackets[i] = ub[i].opackets;
            obytes[i] = ub[i].obytes;
            oerrors[i] = ub[i].oerrors;
            collisions[i] = ub[i].collisions;
        }
    } else {
        perror("perfstat_netinterface 1");
    }
}

```



```

    }
}

netinterface_total_t(int t)
{
    perfstat_netinterface_total_ub;
    static u_longlong_t_ipackets;
    static u_longlong_t_ibytes;
    static u_longlong_t_ierrors;
    static u_longlong_t_opackets;
    static u_longlong_t_obytes;
    static u_longlong_t_oerrors;
    static u_longlong_t_collisions;

    if (t) {
        if (perfstat_netinterface_total
            ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_netinterface_total_t),1) >= 0) {
            printf("%6.6s %6.6s %6.6s %6.6s %6.6s %6.6s %6.6s\n",
                "ipack","ibyte","ierr","opack"," obyte"," oerr","coll");
            printf("%6lld ", ub.ipackets-ipackets);
            printf("%6lld ", ub.ibytes-ibytes);
            printf("%6lld ", ub.ierrors-ierrors);
            printf("%6lld ", ub.opackets-opackets);
            printf("%6lld ", ub.obytes-obytes);
            printf("%6lld ", ub.oerrors-oerrors);
            printf("%6lld ", ub.collisions-collisions);
            printf("\n\n");
        } else {
            perror("perfstat_netinterface_total 1");
        }
    } else {
        if (perfstat_netinterface_total
            ((perfstat_id_t*)NULL,&ub,sizeof(perfstat_netinterface_total_t),1) >= 0) {
            ipackets = ub.ipackets;
            ibytes = ub.ibytes;
            ierrors = ub.ierrors;
            opackets = ub.opackets;
            obytes = ub.obytes;
            oerrors = ub.oerrors;
            collisions = ub.collisions;
        } else {
            perror("perfstat_netinterface_total 0");
        }
    }
}

main()
{

```

```

struct variovario;
int      rc;

if (!sys_parm(SYSP_GET,SYSP_V_NCPUS,&vario))
    ncpu = vario.v.v_ncpus_cfg.value;

if ((rc = perfstat_cpu (NULL,NULL,sizeof(perfstat_cpu_t),0)) > 0)
    ncpu = rc;

if ((rc = perfstat_disk (NULL,NULL,sizeof(perfstat_disk_t),0)) > 0)
    ndisk = rc;

if ((rc = perfstat_netinterface
(NULL,NULL,sizeof(perfstat_netinterface_t),0)) > 0)
    nnetwork = rc;

cpu_total_t(0);
cpu_t(0);
memory_total_t(0);
disk_total_t(0);
disk_t(0);
netinterface_total_t(0);
netinterface_t(0);

sleep(1);

cpu_total_t(1);
cpu_t(1);
memory_total_t(1);
disk_total_t(1);
disk_t(1);
netinterface_total_t(1);
netinterface_t(1);
}

```

Example A-3 shows a sample output from perfstat_dude program.

Example: A-3 Output from perfstat_dude

```

# perfstat_dude
Que      Faults
rq  sq    fk    in    sy    cs  us  sy  id  wa
0   0     0     0  2359  1473  0  0  86  13

      cpu    fk    sy    cs  us  sy  id  wa
cpu0    0   240  240  0  0  99  0
cpu1    0   289  300  0  0 100  0
cpu2    0   337  336  0  0  99  0
cpu3    0  1231  594  0  0  45  52

```

Real memory		Paging space Virtual									
free	use	free	psi	ps0	pi	po	fault	fr	sr	num	
1753170	343982	1046751	614369	4217286	716225	5114271	143100457	4224489	70493357	95299	

busy	KB/s	xfers	KBrd	KBwr
116	0	228	21054	256

disk	vg	qd	busy	KB/s	xfers	KBrd	KBwr
hdisk0	rootvg	0	22	0	55	4210	248
hdisk1	datavg	0	7	0	24	4210	0
hdisk2	ssavg	0	79	0	96	8420	0
hdisk4	None	0	0	0	0	0	0
hdisk6	None	0	0	0	0	0	0
hdisk5	None	0	0	0	0	0	0
hdisk7	ssavg2	0	3	0	48	4210	0
hdisk3	ssavg	0	3	0	4	4	0
cd0	not available	0	0	0	0	0	0

ipack	ibyte	ierr	opack	obyte	oerr	coll
14	1074	0	0	0	0	0

network	mtu	ipack	ibyte	ierr	opack	obyte	oerr	coll
en0	1500	7	537	0	0	0	0	0
en1	1500	7	537	0	0	0	0	0
lo0	16896	0	0	0	0	0	0	0

spmi_dude.c

Example A-4 shows the source code for the `spmi_dude.c` program.

Example: A-4 `spmi_dude.c` source code

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/Spmidef.h>

#if defined(DEBUG)
#define PDEBUG(x,y) printf(x,y)
#else
#define PDEBUG(x,y)
#endif

extern      errno;
extern charSpmiErrmsg[];
```

```

extern intSpmiErrno;
/*
 * Since we need this structure pointer in our cleanup() function
 * we declare it as a global variable.
 */
struct SpmiStatSet*SPMIset = NULL;
/*
 * These are the statistics we are interested in monitoring.
 * To the left of the last slash (/) is the context, to the
 * right of this slash (/) is the actual statistic within
 * the context. Note that statistics can have the same
 * name but belong to different contexts.
 */
char      *stats[] = {
    "CPU/glwait",
    "CPU/glidle",
    "CPU/glkern",
    "CPU/gluser",
    "Mem/Virt/scan",
    "Mem/Virt/steal",
    "PagSp/%totalfree",
    "PagSp/%totalused",
    "Mem/Virt/pagein",
    "Mem/Virt/pageout",
    "Mem/Virt/pgspgin",
    "Mem/Virt/pgspgout",
    "Proc/runque",
    "Proc/swpque",
    NULL
};

void
SPMIerror(char *s)
{
    /* We do not want the \n that the SpmiErrmsg have at the
     * end since we will use our own error reporting format.
     */
    SpmiErrmsg[strlen(SpmiErrmsg)-1] = 0x0;
    fprintf(stderr,"%s: %s (%d)\n",s,SpmiErrmsg,SpmiErrno);
}
/*
 * This subroutine is called when a user interrupts it or
 * when the main program exits. If called by a signal handler
 * it will have a value in parameter s. If s is not set, then
 * it is called when the main program exits. To not have this
 * subroutine called when calling exit() to terminate the
 * process, we use _exit() instead. Since exit() would call
 * _cleanup() and any atexit() registred functions, we call
 * _cleanup() ourselves.
 */

```

```

*/
void
cleanup(int s)
{
    if (SPMIset)
        if (SpmiFreeStatSet(SPMIset))
            SPMIError("SpmiFreeStatSet");
    SpmiExit();
    _cleanup();
    _exit(0);
}

#define MAXDELAY2
#define MAXCOUNT-1

main(int argc, char *argv[])
{
    struct SpmiStatVals*SPMIval = NULL;
    struct SpmiStat*SPMIstat = NULL;
    SpmiCxHdl SPMIcxhdl = 0;
    char        context[128];
    char        *statistic;
    float       statvalue;
    int         i, hardcore = 0, bailout = 0;
    int         maxdelay = MAXDELAY;
    uint        maxcount = MAXCOUNT;
    /*
     * Here we initialize the SPMI environment for our process.
     */
    if (SpmiInit(15)) {
        SPMIError("SpmiInit");
        exit(SpmiErrno);
    }
    if (argc == 2)
        maxdelay = atoi(argv[1]);
    else if (argc == 3) {
        maxdelay = atoi(argv[1]);
        maxcount = atoi(argv[2]);
    }
    /*
     * To illustrate enhanced durability of our simple program.
     */
    hardcore = atoi(getenv("HARDCORE"));
    /*
     * We make sure that we clean up the SPMI memory that we use
     * before we terminate the process. atexit() is called when
     * the process is normally terminated, and we trap signals
     * that a terminal user, or program malfunction could
     * generate and cleanup then as well.

```

```

*/
atexit(cleanup);
signal(SIGINT,cleanup);
signal(SIGTERM,cleanup);
signal(SIGSEGV,cleanup);
signal(SIGQUIT,cleanup);
/*
 * Here we create the base for our SPMI statistical data hierarchy.
 */
if ((SPMIset = SpmiCreateStatSet()) == NULL) {
    SPMIerror("SpmiCreateStatSet");
    exit(SpmiErrno);
}
/*
 * For each metric we want to monitor we need to add it to
 * our statistical collection set.
 */
for (i = 0; stats[i] != NULL; i++) {
    if (SpmiPathAddSetStat(SPMIset,stats[i],SPMIcxhdl) == NULL) {
        SPMIerror("SpmiPathAddSetStats");
        exit(SpmiErrno);
    }
}
printf ("%5s %5s %5s %5s %5s %5s %5s %5s %5s %5s %5s %5s %5s\n",
        "swpq", "runq", "pgspo", "pgspi", "pgout", "pgin",
        "%used", "%free", "fr", "sr", "us", "sy", "id", "wa");
/*
 * In this for loop we collect all statistics that we have specified
 * to SPMI that we want to monitor. Each of the data values selected
 * for the set is represented by an SpmiStatVals structure.
 * Whenever Spmi executes a request from the to read the data values
 * for a set all SpmiStatVals structures in the set are updated.
 * The application program will then have to traverse the list of
 * SpmiStatVals structures through the SpmiFirstVals() and SpmiNextVals()
 * function calls.
 */
for (i=0; i< maxcount; i++) {
again:
    /*
     * First we must request that SPMI refresh our statistical
     * data hierarchy.
     */
    if ((SpmiGetStatSet(SPMIset,TRUE)) != 0) {
        /*
         * if the hardcore variable is set (environment variable HARDCORE),
         * then we discard runtime errors from SpmiGetStatSet (up to three
         * times). This can happen some time if many processes use the SPMI
         * shared resources simultaneously.
         */
    }
}

```

```

        if (hardcore && (3 > bailout++)) goto again;
        SPMIerror("SpmiGetStatSet");
        exit(SpmiErrno);
    }
    bailout = 0;
    /*
     * Here we get the first entry point in our statistical data hierarchy.
     * Note that SPMI will return the values in the reverse order of the one
     * used to add them to our statistical set.
     */
    SPMIval = SpmiFirstVals(SPMIset);
    do {
        if ((statvalue = SpmiGetValue(SPMIset,SPMIval)) < 0) {
            SPMIerror("SpmiGetValue");
            exit(SpmiErrno);
        }
        printf("%5.0f ",statvalue);
        PDEBUG("\t%s\n",SpmiStatGetPath(SPMIval->context,SPMIval->stat, 0));
    /*
     * Finally we get the next statistic in our data hierarchy.
     * And if this is NULL, then we have retrieved all our statistics.
     */
    } while ((SPMIval = SpmiNextVals(SPMIset,SPMIval)));
    printf("\n");
    sleep(maxdelay);
}
}

```

spmi_data.c

Example A-5 shows the source code for the spmi_data.c program.

Example: A-5 spmi_data.c source code

```

/* The following statistics are added by the SpmiPathAddSetStat
 * subroutine to form a set of statistics:
 * CPU/cpu0/kern
 * CPU/cpu0/idle
 * Mem/Real/%free
 * PagSp/%free
 * Proc/runque
 * Proc/swpque
 * These statistics are then retrieved every 2 seconds and their
 * value is displayed to the user.
 */
#include <sys/types.h>
#include <sys/errno.h>

```

```

#include <signal.h>
#include <stdio.h>
#include <sys/Spmidef.h>

#define TIME_DELAY 2          /* time between samplings */

extern char  SpmiErrmsg[];    /* Spmi Error message array */
extern int   SpmiErrno;      /* Spmi Error indicator */

struct SpmiStatSet *statset; /* statistics set */

/*===== must_exit() =====*/
/* This subroutine is called when the program is ready to exit.
 * It frees any statsets that were defined and exits the
 * interface.
 */
/*=====*/

void must_exit()
{
    /* free statsets */
    if (statset)
        if (SpmiFreeStatSet(statset))
            if (SpmiErrno)
                printf("%s", SpmiErrmsg);

    /* exit SPMI */
    SpmiExit();
    if (SpmiErrno)
        printf("%s", SpmiErrmsg);
    exit(0);
}

/*===== getstats() =====*/
/* getstats() traverses the set of statistics and outputs the
 * statistics values.
 */
/*=====*/

void getstats()
{
    int          counter=20;    /* every 20 lines output
                                * the header
                                */

    struct SpmiStatVals *statvall;
    float         spmvalue;

    /* loop until a stop signal is received. */
    while (1) {

```



```

if(counter == 20) {
    printf("\nCPU/cpu0  CPU/cpu0  Mem/Real  PagSp    ");
    printf("Proc      Proc\n");
    printf("   kern      idle   %%free   %%free   ");
    printf("runque      swpque\n");
    printf("=====");
    printf("=====\n");
    counter=0;
}

/* retrieve set of statistics */
if (SpmiGetStatSet(statset, TRUE) != 0) {
    printf("SpmiGetStatSet failed.\n");
    if (SpmiErrno)
        printf("%s", SpmiErrmsg);
    must_exit();
}

/* retrieve first statistic */
statvall = SpmiFirstVals(statset);
if (statvall == NULL) {
    printf("SpmiFirstVals Failed\n");
    if (SpmiErrno)
        printf("%s", SpmiErrmsg);
    must_exit();
}

/* traverse the set of statistics */
while (statvall != NULL) {
    /* value to be displayed */
    spmivalue = SpmiGetValue(statset, statvall);
    if (spmivalue < 0.0) {
        printf("SpmiGetValue Failed\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }
    printf(" %6.2f  ",spmivalue);

    statvall = SpmiNextVals(statset, statvall);
} /* end while (statvall) */
printf("\n");
counter++;
sleep(TIME_DELAY);
}

/*===== addstats() =====*/
/* addstats() adds statistics to the statistics set. */

```

```

/* addstats() also takes advantage of the different ways a
 * statistic may be added to the set.
 */
/*=====*/
void addstats()
{
    SpmiCxHd1    cxhdl, parenthdl;

    /* initialize the statistics set */
    statset = SpmiCreateStatSet();
    if (statset == NULL)
    {
        printf("SpmiCreateStatSet Failed\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    /* Pass SpmiPathGetCx the fully qualified path name of the
     * context
     */
    if (!(cxhdl = SpmiPathGetCx("Proc", NULL)))
    {
        printf("SpmiPathGetCx failed for Proc context.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    /* Pass SpmiPathAddSetStat the name of the statistic */
    /* & the handle of the parent */
    if (!SpmiPathAddSetStat(statset,"swpque", cxhdl))
    {
        printf("SpmiPathAddSetStat failed for Proc/swpque statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    if (!SpmiPathAddSetStat(statset,"runque", cxhdl))
    {
        printf("SpmiPathAddSetStat failed for Proc/runque statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    /* Pass SpmiPathAddSetStat the fully qualified name of the

```

```

    * statistic
    */
    if (!SpmiPathAddSetStat(statset, "PagSp/%totalfree", NULL))
    {
        printf("SpmiPathAddSetStat failed for PagSp/%%free statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    if (!(parenthd1 = SpmiPathGetCx("Mem", NULL)))
    {
        printf("SpmiPathGetCx failed for Mem context.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    /* Pass SpmiPathGetCx the name of the context */
    /* & the handle of the parent context */
    if (!(cxhd1 = SpmiPathGetCx("Real", parenthd1)))
    {
        printf("SpmiPathGetCx failed for Mem/Real context.\n");
        if (SpmiErrmsg)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    if (!SpmiPathAddSetStat(statset, "%free", cxhd1))
    {
        printf("SpmiPathAddSetStat failed for Mem/Real/%%free statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    /* Pass SpmiPathGetCx the fully qualified path name of the
    * context
    */
    if (!(cxhd1 = SpmiPathGetCx("CPU/cpu0", NULL)))
    {
        printf("SpmiPathGetCx failed for CPU/cpu0 context.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    if (!SpmiPathAddSetStat(statset, "idle", cxhd1))
    {

```

```

        printf("SpmiPathAddSetStat failed for CPU/cpu0/idle statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    if (!SpmiPathAddSetStat(statset,"kern", cxhd1))
    {
        printf("SpmiPathAddSetStat failed for CPU/cpu0/kern statistic.\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        must_exit();
    }

    return;
}

/*-----*/
main(int argc, char **argv)
{
    int  spmierr=0;

    /* Initialize SPMI */
    if ((spmierr = SpmiInit(15)) != 0)
    {
        printf("Unable to initialize SPMI interface\n");
        if (SpmiErrno)
            printf("%s", SpmiErrmsg);
        exit(-98);
    }

    /* set up interrupt signals */
    signal(SIGINT,must_exit);
    signal(SIGTERM,must_exit);
    signal(SIGSEGV,must_exit);
    signal(SIGQUIT,must_exit);

    /* Go to statistics routines. */
    addstats();
    getstats();

    /* Exit SPMI */
    must_exit();
}

```

spmi_file.c

Example A-6 shows the source code for the `spmi_file.c` program.

Example: A-6 `spmi_file.c` source code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/Spmidef.h>

extern      errno;
extern charSpmiErrmsg[];
extern intSpmiErrno;

struct SpmiStatSet*SPMIset = NULL;

void
SPMIerror(char *s)
{
    /* We do not want the \n that the SpmiErrmsg have at the
     * end since we will use our own error reporting format.
     */
    SpmiErrmsg[strlen(SpmiErrmsg)-1] = 0x0;
    fprintf(stderr,"%s: %s (%d)\n",s,SpmiErrmsg,SpmiErrno);
}
/*
 * This subroutine is called when a user interrupts it or
 * when the main program exits. If called by a signal handler
 * it will have a value in parameter s. If s is not set, then
 * it is called when the main program exits. To not have this
 * subroutine called when calling exit() to terminate the
 * process, we use _exit() instead. Since exit() would call
 * _cleanup() and any atexit() registered functions, we call
 * _cleanup() ourselves.
 */
void
cleanup(int s)
{
    if (SPMIset)
        if (SpmiFreeStatSet(SPMIset))
            SPMIerror("SpmiFreeStatSet");
    SpmiExit();
    _cleanup();
    _exit(0);
}

main(int argc, char *argv[])
{
    struct SpmiStatVals*SPMIval = NULL;
```

```

struct SpmiStat*SPMIstat = NULL;
SpmiCxHdl SPMIcxhdl = 0;
FILE      *file;
char      stats[4096];
float     statvalue;
/*
 * Here we initialize the SPMI environment for our process.
 */
if (SpmiInit(15)) {
    SPMIerror("SpmiInit");
    exit(SpmiErrno);
}
/*
 * We make sure that we clean up the SPMI memory that we use
 * before we terminate the process. atexit() is called when
 * the process is normally terminated, and we trap signals
 * that a terminal user, or program malfunction could
 * generate and cleanup then as well.
 */
atexit(cleanup);
signal(SIGINT,cleanup);
signal(SIGTERM,cleanup);
signal(SIGSEGV,cleanup);
signal(SIGQUIT,cleanup);
/*
 * Here we create the base for our SPMI statistical data hierarchy.
 */
if ((SPMIset = SpmiCreateStatSet()) == NULL) {
    SPMIerror("SpmiCreateStatSet");
    exit(SpmiErrno);
}
/*
 * Open the file we have the SPMI metrics stored in
 */
if ((file = fopen("SPMI_METRICS", "r")) == NULL) exit(1);
/*
 * Read all lines in the file
 */
while (fscanf(file,"%s",&stats) != EOF) {
    /*
     * For each metric we want to monitor we need to add it to
     * our statistical collection set (assuming the input file syntax is
correct).
     */
    if ((SPMIval = SpmiPathAddSetStat(SPMIset,stats,SPMIcxhdl)) == NULL) {
        SPMIerror("SpmiPathAddSetStats");
        exit(SpmiErrno);
    }
}
}

```

```

fclose(file);
/*
 * First we must request that SPMI refresh our statistical
 * data hierarchy.
 */
if ((SpmiGetStatSet(SPMIset,TRUE)) != 0) {
    SPMIerror("SpmiGetStatSet");
    exit(SpmiErrno);
}
/*
 * Here we get the first entry point in our statistical data hierarchy.
 * Note that SPMI will return the values in the reverse order of the one
 * used to add them to our statistical set.
 */
SPMIval = SpmiFirstVals(SPMIset);
do {
    if ((statvalue = SpmiGetValue(SPMIset,SPMIval)) < 0) {
        SPMIerror("SpmiGetValue");
        exit(SpmiErrno);
    }
    printf("%-25s:
%.0f\n",SpmiStatGetPath(SPMIval->context,SPMIval->stat,0),statvalue);
    /*
     * Finally we get the next statistic in our data hierarchy.
     * And if this is NULL, then we have retrieved all our statistics.
     */
} while ((SPMIval = SpmiNextVals(SPMIset,SPMIval)));
}

```

spmi_traverse.c

Example A-7 shows the source code for the `spmi_traverse.c` program.

Example: A-7 `spmi_traverse.c` source code

```

#include <sys/types.h>
#include <sys/errno.h>
#include <stdio.h>
#include <sys/Spmidef.h>

extern      errno;
extern charSpmiErrmsg[];
extern intSpmiErrno;

SPMIerror(char *s)
{
    /* We do not want the \n that the SpmiErrmsg have at the

```

```

        * end since we will use our own error reporting format.
        */
    SpmiErrMsg[strlen(SpmiErrMsg)-1] = 0x0;
    fprintf(stderr,"%s: %s (%d)\n",s,SpmiErrMsg,SpmiErrno);
}
/*
 * This subroutine is called when a user interrupts it or
 * when the main program exits. If called by a signal handler
 * it will have a value in parameter s. If s is not set, then
 * it is called when the main program exits. To not have this
 * subroutine called when calling exit() to terminate the
 * process, we use _exit() instead. Since exit() would call
 * _cleanup() and any atexit() registred functions, we call
 * _cleanup() ourselves.
 */
void
cleanup(int s)
{
    SpmiExit();
    _cleanup ();
    _exit (0);
}
/*
 * This function that traverses recursively down a
 * context link. When the end of the context link is found,
 * findstats traverses down the statistics links and writes the
 * statistic name to stdout. findstats is originally passed the
 * context handle for the TOP context.
 */
findstats(SpmiCxHdl SPMIcxhdl)
{
    struct SpmiCxLink *SPMIcxlink;
    struct SpmiStatLink *SPMIstatlink;
    struct SpmiCx *SPMIcx, *SPMIcxparent;
    struct SpmiStat *SPMIstat;
    int instantiable;
    /*
     * Get the first context.
     */
    if (SPMIcxlink = SpmiFirstCx(SPMIcxhdl)) {
        while (SPMIcxlink) {
            SPMIcx = SpmiGetCx(SPMIcxlink->context);
            /*
             * Determine if the context's parent is instantiable
             * because we do not want to have to print the metrics
             * for every child of that parent, ie Procs/<PID>/metric
             * will be the same for every process.
             */
            SPMIcxparent = SpmiGetCx(SPMIcx->parent);

```



```

if (SPMIcxparent->inst_freq == SiContInst)
    instantiable++;
else
    instantiable = 0;
/*
 * We only want to print out the stats for any contexts
 * whose parents aren't instantiable. If the parent
 * is instantiable then we only want to print out
 * the stats for the first instance of that parent.
 */
if (instantiable > 1) {
    /*
     * Output the name of the metric with instantiable parents.
     */
    fprintf(stdout, "%s/%s/.....\n", SPMIcxparent->name, SPMIcx->name);
} else {
    /*
     * Traverse the stats list for the context.
     */
    if (SPMIstatlink = SpmiFirstStat(SPMIcxlink->context)) {
        while (SPMIstatlink) {
            SPMIstat = SpmiGetStat(SPMIstatlink->stat);
            /*
             * Output name of the statistic.
             */
            fprintf(stdout, "%s:%s",

SpmiStatGetPath(SPMIcxlink->context, SPMIstatlink->stat, 10),
                SPMIstat->description);
            /*
             * Output data type/value type about the metric
             */
            fprintf(stdout, ":%s/%s",
                (SPMIstat->data_type == SiLong?"Long":"Float"),
                (SPMIstat->value_type ==
SiCounter?"Counter":"Quantity"));
            /*
             * Output max/min information about the metric.
             */
            fprintf(stdout, ":%ld-%ld\n", SPMIstat->min, SPMIstat->max);
            /*
             * Get next SPMIstatlink
             */
            SPMIstatlink = SpmiNextStat(SPMIstatlink);
        }
    }
}
/*
 * Recursive call to this function, this gets the next context link

```

```

        */
        findstats(SPMIcxlink->context);
        /*
        * After returning from the previous link, we go to the next context
        */
        SPMIcxlink = SpmiNextCx(SPMIcxlink);
    }
}

main(int argc, char *argv[])
{
    int    spmierr=0;
    SpmiCxHdlSPMIcxhdl;
    /*
    * Here we initialize the SPMI environment for our process.
    */
    if ((spmierr = SpmiInit(15)) != 0) {
        SPMIerror("SpmiInit");
        exit(errno);
    }
    /*
    * We make sure that we clean up the SPMI memory that we use
    * before we terminate the process. atexit() is called when
    * the process is normally terminated, and we trap signals
    * that a terminal user, or program malfunction could
    * generate and cleanup then as well.
    */
    atexit(cleanup);
    signal(SIGINT,cleanup);
    signal(SIGTERM,cleanup);
    signal(SIGSEGV,cleanup);
    signal(SIGQUIT,cleanup);

    if ((SPMIcxhdl = SpmiPathGetCx(NULL, NULL)) == NULL)
        SPMIerror("SpmiPathGetCx");
    else
        /*
        * Traverse the SPMI statistical data hierarchy.
        */
        findstats(SPMIcxhdl);
}

```

dudestat.c

Example A-8 shows the source code for the dudestat.c program.

Example: A-8 dudestat.c source code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/var.h>
#include <sys/vminfo.h>
#include <sys/wlm.h>
#include <procinfo.h>
#include <sys/proc.h>
#include <usersec.h>

sys_param_dude()
{
    struct variovario;

    if (!sys_parm(SYSP_GET,SYSP_V_MAXUP,&vario))
        printf("v_maxup (max. # of user processes)           : %lld\n",
vario.v.v_maxup.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MAXPOUT,&vario))
        printf("v_maxpout (# of file pageouts at which waiting occurs): %lld\n",
vario.v.v_maxpout.value);
    if (!sys_parm(SYSP_GET,SYSP_V_MINPOUT,&vario))
        printf("v_minpout (# of file pageout at which ready occurs) : %lld\n",
vario.v.v_minpout.value);
    if (!sys_parm(SYSP_GET,SYSP_V_FILE,&vario))
        printf("v_file (# entries in open file table)           : %lld\n",
vario.v.v_file.value);
    if (!sys_parm(SYSP_GET,SYSP_V_PROC,&vario))
        printf("v_proc (max # of system processes)           : %lld\n",
vario.v.v_proc.value);

    if ((!sys_parm(SYSP_GET,SYSP_V_NCPUS,&vario)) !=
(!sys_parm(SYSP_GET,SYSP_V_NCPUS_CFG,&vario)))
        printf("Dude! v_ncpus %d (number of active CPUs) \
does not match v_ncpus_cfg %d (number of processor configured)\n",
vario.v.v_ncpus_cfg.value,
vario.v.v_ncpus_cfg.value);
}

vmgetinfo_dude()
{
    struct vminfovminfo;

    if (!vmgetinfo(&vminfo,VMINFO,sizeof(vminfo))) {
```

```

        printf("freewts (count of free frame waits)           :
%lld\n",vminfo.freewts);
        printf("extendwts (count of extend XPT waits)       :
%lld\n",vminfo.extendwts);
        printf("pendiowts (count of pending I/O waits)     :
%lld\n",vminfo.pendiowts);
        printf("numfrb (number of pages on free list)       :
%lld\n",vminfo.numfrb);
        printf("numclient (number of client frames)        :
%lld\n",vminfo.numclient);
        printf("numcompress (no of frames in compressed segments) :
%lld\n",vminfo.numcompress);
        printf("numperm (number frames non-working segments) :
%lld\n",vminfo.numperm);
        printf("maxperm (max number of frames non-working)  :
%lld\n",vminfo.maxperm);
        printf("maxclient (max number of client frames)     :
%lld\n",vminfo.maxclient);
        printf("memsizepgs (real memory size in 4K pages)   :
%lld\n",vminfo.memsizepgs);
    }
}

swapqry_dude()
{
    struct pginfopginfo;
    char    device[256];
    char    path[256];
    char    cmd[256];
    FILE    *file;

    bzero(cmd,sizeof(cmd));
    sprintf(cmd,"odmget -q \"value = paging\" CuAt|awk
'/name/{gsub(\"\\\\\"\\\\\",\\\"\",$3);print $3}'\n");
    if (file = popen(cmd,"r"))
        while (fscanf(file,"%s\n", &device)!=EOF) {
            sprintf(path,"/dev/%s", device);
            if (!swapqry(path,&pginfo)) {
                printf("paging space device           :
%s\n",path);
                printf("size (size in PAGESIZE blocks) :
%u\n",pginfo.size);
                printf("free (# of free PAGESIZE blocks) :
%u\n",pginfo.free);
                printf("iocnt (number of pending i/o's) :
%u\n",pginfo.iocnt);
            }
        }
    pclose(file);
}

```

```

}

getprocs_dude(char *dudes[])
{
    struct procsinfo ps[8192];
    int         uids[12];
    pid_t       index = 0;
    int         nprocs;
    int         i,j,k;
    char        *p;

    if (dudes[0] != NULL)
        if ((nprocs = getprocs(&ps, sizeof(struct procsinfo), NULL, 0, &index,
8192)) > 0)
            for (i = 0, k = 0; dudes[i] != NULL; i++)
                for (j=0; j<nprocs; j++) {
                    p = IDtouser(ps[j].pi_uid);
                    if (!strcmp(dudes[i],p)) {
                        printf ("The %s dude is online and
excellent!\n\n",dudes[i]);
                        uids[k++] = ps[j].pi_uid;
                        break;
                    }
                }
            if (i != k) {
                j = i - k;
                printf ("There %s %d dude%s
missing!\n\n", (j>1?"are":"is", j, (j>1)"s":""));
            }
        }

main(int argc, char *argv[])
{
    printf("PARTY ON!\n\n");
    getprocs_dude(argc>1?&argv[1]:NULL);
    printf("Dude, here are some excellent info for you today\n\n");
    sys_param_dude();
    vmgetinfo_dude();
    swapqry_dude();
}

```

cwhet.c

Example A-9 shows the source for the cwhet.c program. This Whetstone benchmark was written by Harold Curnow of CCTA, the British government computer procurement agency, based on work by Brian Wichmann of the National Physical Laboratory.

Example: A-9 The cwhet.c file

```
/* HARDENED WHETSTONE.
   Module 8 changed. Inlining will not throw Module 8 away now.
   Remove <#define HARD> to get the soft version
*/

/* Whetstone benchmark -- Double Precision.
   This program has a long history and is well described in "A Synthetic
   Benchmark" by H.J. Curnow and B.A. Wichman in Computer Journal, Vol.
   19 #1, February 1976.

   The number of ITERATIONS was increased from 10 to 10000 to minimize
   system overhead.
*/

#define ITERATIONS      10000
#define POUT
#define HARD

#include "math.h"
#include <stdio.h>

double  x1, x2, x3, x4, x, y, z, t, t1, t2;
double  e1[4];
int     i, j, k, l, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11;

main()
{
    t = 0.499975;
    t1 = 0.50025;
    t2 = 2.0;

    n1 = 0;
    n2 = 12 * ITERATIONS;
    n3 = 14 * ITERATIONS;
    n4 = 345 * ITERATIONS;
    n5 = 0;
    n6 = 210 * ITERATIONS;
    n7 = 32 * ITERATIONS;
    n8 = 899 * ITERATIONS;
    n9 = 616 * ITERATIONS;
```

```

n10 = 0;
n11 = 93 * ITERATIONS;

/**** Module 1: Simple Identifier ****/

x1 = 1.0;
x2 = x3 = x4 = -1.0;
for (i = 1; i <= n1; i++) {
    x1 = ( x1 + x2 + x3 - x4) * t;
    x2 = ( x1 + x2 - x3 + x4) * t;
    x3 = ( x1 - x2 + x3 + x4) * t;
    x4 = (-x1 + x2 + x3 + x4) * t;
}
#ifdef POUT
    pout(n1, n1, n1, x1, x2, x3, x4);
#endif

/**** Module 2: array elements ****/

e1[0] = 1.0;
e1[1] = e1[2] = e1[3] = -1.0;
for (i = 1; i <= n2; i++) {
    e1[0] = ( e1[0] + e1[1] + e1[2] - e1[3]) * t;
    e1[1] = ( e1[0] + e1[1] - e1[2] + e1[3]) * t;
    e1[2] = ( e1[0] - e1[1] + e1[2] + e1[3]) * t;
    e1[3] = ( -e1[0] + e1[1] + e1[2] + e1[3]) * t;
}
#ifdef POUT
    pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

/**** Module 3: Array as Parameter ****/

for (i = 1; i <= n3; i++) {
    mod3(e1);
}
#ifdef POUT
    pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
#endif

/**** Module 4: Conditional Jumps ****/

j = 1;
for (i = 1; i <= n4; i++) {
    if (j == 1) j = 2;
    else j = 3;
    if (j > 2) j = 0;
    else j = 1;
    if (j < 1) j = 1;
}

```

```

        else j = 0;
    }
#ifdef POUT
    pout(n4, j, j, x1, x2, x3, x4);
#endif

    /**** Module 6: Integer Arithmetic Using Arrays ****/

    j = 1; k = 2; l = 3;
    for (i = 1; i <= n6; i++) {
        j = j * (k - j) * (l - k);
        k = l * k - (l - j) * k;
        l = (l - k) * (k + j);
        e1[l - 2] = j + k + l;
        e1[k - 2] = j * k * l;
    }
#ifdef POUT
    pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif

    /**** Module 7 : Trigonometric functions ****/

    x = y = 0.5;
    for (i = 1; i <= n7; i++) {
        x = t * atan(t2 * sin(x) * cos(x) / (cos(x + y) + cos(x - y) - 1.0));
        y = t * atan(t2 * sin(y) * cos(y) / (cos(x + y) + cos(x - y) - 1.0));
    }
#ifdef POUT
    pout(n7, j, k, x, x, y, y);
#endif

    /**** Module 8 Procedure Call ****/

    x = y = z = 1.0;
    for (i = 1; i <= n8; i++) {
        mod8(x, y, &z);
#ifdef HARD
        x = z;
#endif
    }
#ifdef POUT
    pout(n8, j, k, x, y, z, z);
#endif

    /**** Module 9: Array References ****/

    j = 1;
    k = 2;
    l = 3;

```



```

    e1[1] = 1.0;
    e1[2] = 2.0;
    e1[3] = 3.0;
    for (i = 1; i <= n9; i++) {
        mod9();
    }
#ifdef POUT
    pout(n9, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif

    /**** Module 10: Integer Arithmetic ****/

    j = 2;
    k = 3;
    for (i = 1; i <= n10; i++) {
        j = j + k;
        k = j + k;
        j = k - j;
        k = k - j - j;
    }
#ifdef POUT
    pout(n10, j, k, x1, x2, x3, x4);
#endif

    /**** Module 11: Standard Functions ****/

    x = 0.75;
    for (i = 1; i <= n11; i++) {
        x = sqrt(exp(log(x) / t1));
    }
#ifdef POUT
    pout(n11, j, k, x, x, x, x);
#endif

} /* End of Main */

/**** Module 3 Routine ****/
mod3(a)
double a[4];
{
    register int j;
    for (j = 0; j < 6; j++) {
        a[0] = ( a[0] + a[1] + a[2] - a[3] ) * t;
        a[1] = ( a[0] + a[1] - a[2] + a[3] ) * t;
        a[2] = ( a[0] - a[1] + a[2] + a[3] ) * t;
        a[3] = (-a[0] + a[1] + a[2] + a[3] ) / t2;
    }
}

```

```
**** Module 8 Routine ****/
mod8(x, y, z)
double x, y, *z;
{
    x = t * (x + y);
    y = t * (x + y);
    *z = (x + y) / t2;
}

**** Module 9 Routine ****/
mod9()
{
    e1[j] = e1[k];
    e1[k] = e1[l];
    e1[l] = e1[j];
}

#ifdef POUT
pout(n, j, k, x1, x2, x3, x4)
int n, j, k;
double x1, x2, x3, x4;
{
    printf("%6d %6d %6d %5e %5e %5e %5e\n",
           n, j, k, x1, x2, x3, x4);
}
#endif
```

Trace hooks

This appendix contains a listing of the AIX 5L trace hook IDs. Trace hooks can be thought of as markers in a trace report that mark certain events. After creating the trace report, the trace hooks can then be used to search for these events.

A trace report can be taken with all trace hooks active, or with only certain trace hooks active. It is a particularly good idea to limit the number of events that are captured (by limiting the number of trace hooks) on systems that are very busy, especially large SMP systems. Because the trace buffers are limited in size and can grow extremely quickly, avoid filling the buffer by limiting the number of trace hooks. Refer to Chapter 40, “The trace, trcnm, and trcrpt commands” on page 759 for further information about **trace**. The trace hooks that are needed by AIX trace post-processing tools, such as **filemon**, **netpmon**, **tprof**, or **curt**, are specified in the AIX documentation that can be found at:

http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/

AIX 5L trace hooks

The following list of trace hooks and their respective hook IDs can be obtained by running the `trcrpt -j` command. It is recommended that you run `trcrpt -j` every time the operating system is updated to check for any modifications to the trace hooks that IBM may make.

Example: B-1 AIX 5.2 trace hooks using trcrpt -j

```
#uname -a
AIX lpar05 2 5 0021768A4C00
#trcrpt -j
001 TRACE ON
002 TRACE OFF
003 TRACE HEADER
004 TRACEID IS ZERO
005 LOGFILE WRAPAROUND
006 TRACEBUFFER WRAPAROUND
007 UNDEFINED TRACE ID
008 DEFAULT TEMPLATE
00a TRACE_UTIL
100 FLIH
101 SYSTEM CALL
102 SLIH
103 RETURN FROM SLIH
104 RETURN FROM SYSTEM CALL
105 LVM EVENTS
106 DISPATCH
107 FILENAME TO VNODE (lookupn)
108 FILE ORIENTED SYSTEM CALLS
10a KERN_PFS
10b LVM BUF STRUCT FLOW
10c DISPATCH IDLE PROCESS
10d FILE VFS AND INODE
10e LOCK OWNERSHIP CHANGE
10f KERN_EOF
110 KERN_STDERR
111 KERN_LOCKF
112 LOCK
113 UNLOCK
114 LOCKALLOC
115 SETRECURSIVE
116 XMMALLOC size,align,heap
117 XMFREE address,heap
118 FORKCOPY
119 SENDSIGNAL
11a KERN_RCVSIGNAL
11c P_SLIH
11d KERN_SIGDELIVER
```

11e ISSIG
11f SET ON READY QUEUE
120 ACCESS SYSTEM CALL
121 SYSC_ACCT
122 ALARM SYSTEM CALL
12e CLOSE SYSTEM CALL
130 CREAT SYSTEM CALL
131 DISCLAIM SYSTEM CALL
134 EXEC SYSTEM CALL
135 EXIT SYSTEM CALL
137 FCNTL SYSTEM CALL
139 FORK SYSTEM CALL
13a FSTAT SYSTEM CALL
13b FSTATFS SYSTEM CALL
13e FULLSTAT SYSTEM CALL
14c IOCTL SYSTEM CALL
14e KILL SYSTEM CALL
152 LOCKF SYSTEM CALL
154 LSEEK SYSTEM CALL
15b OPEN SYSTEM CALL
15f PIPE SYSTEM CALL
160 PLOCK
163 READ SYSTEM CALL
169 SBREAK SYSTEM CALL
16a SELECT SYSTEM CALL
16e SETPGRP
16f SBREAK
180 SIGACTION SYSTEM CALL
181 SIGCLEANUP
183 SIGRETURN
18e TIMES
18f ULIMIT SYSTEM CALL
195 USRINFO SYSTEM CALL
19b WAIT SYSTEM CALL
19c WRITE SYSTEM CALL
1a4 GETRLIMIT SYSTEM CALL
1a5 SETRLIMIT SYSTEM CALL
1a6 GETRUSAGE SYSTEM CALL
1a7 GETPRIORITY SYSTEM CALL
1a8 SETPRIORITY SYSTEM CALL
1a9 ABSINTERVAL SYSTEM CALL
1aa GETINTERVAL SYSTEM CALL
1ab GETTIMER SYSTEM CALL
1ac INCINTERVAL SYSTEM CALL
1ad RESTIMER SYSTEM CALL
1ae RESABS SYSTEM CALL
1af RESINC SYSTEM CALL
1b0 VMM_ASSIGN (assign virtual page to a physical page)
1b1 VMM_DELETE (delete a virtual page)

1b2 VMM_PGEXCT (pagefault)
1b3 VMM_PROTEXCT (protection fault)
1b4 VMM_LOCKEXCT (lockmiss)
1b5 VMM_RECLAIM
1b6 VMM_GETPARENT
1b7 VMM_COPYPARENT
1b8 VMM_VMAP (fault on a shared process private segment)
1b9 VMM_ZFOD (zero fill a page)
1ba VMM_PAGEIO
1bb VMM_SEGCREATE (segment create)
1bc VMM_SEGDELETE (segment delete)
1bd VMM_DALLOC
1be VMM_PFEND
1bf VMM_EXCEPT
1c8 PPDD
1ca TAPEDD
1cf C327DD
1d0 DDSPEC_GRAPHIO
1d1 ERRLG
1d2 DUMP
1d9 VMM_ZERO
1da VMM_MKP
1db VMM_FPGIN
1dc VMM_SPACEOK
1dd VMM_LRU
1f0 SETTIMER SYSTEM CALL
200 RESUME
201 KERN_HFT
202 KERN_KTSM
204 SWAPPER swapin process
205 SWAPPER swapout process
206 SWAPPER post process for suspension
207 SWAPPER sched stats
208 SWAPPER process stats
209 SWAPPER sched stats
20a MEMORY SCRUBBING disable
20b MEMORY SCRUBBING enable
20c MEMORY SCRUBBING choose segment of memory
20d MEMORY SCRUBBING report single bit errors
20e LOCKL locks a conventional process lock
20f UNLOCKL unlocks a conventional process lock
211 NFS: Client VNOP read/write routines
212 NFS: Client VNOP routines
213 NFS: Server read/write services
214 NFS: Server services
215 NFS: Server dispatch
216 NFS: Client call
217 NFS: RPC Debug
218 NFS: rpc.lockd hooks

220 FDDD
221 SCDISKDD
222 BADISKDD
223 SCSIDD
226 GIODD
228 SERDASDD
229 TMSCSIDD
22c tsdd
232 SCARRAYDD
233 SCARRAY
234 CLOCK
250
251 NETERR
252 SOCK
254 MBUF
255 NETIF_EN
256 NETIF_TOK
257 NETIF_802.3
258 NETIF_X25
259 NETIF_SER
25a TCPDBG
272 PSLA DR. OPEN(X) CALL
273 PSLA DR. CLOSE CALL
274 PSLA DR. READ CALL
275 PSLA DR. WRITE CALL
276 PSLA DR. IOCTL CALLS
277 PSLA INTERRUPT HANDLER
278 PSLA DR. CONFIG CALL
280 HIADD
292 VCA DEVICE DRIVER
2a1 IDEDISKDD
2a2 IDECDROMDD
2a4 kentdd
2a5 kentdd
2a6 kentdd
2a7 stokdd
2a8 stokdd
2a9 stokdd
2aa stokdd
2c7 chatmdd
2c8 chatmdd
2c9 chatmdd
2ca bbatmdd
2d9 NFS: krpc network hooks
2da cstokdd
2db cstokdd
2dc cstokdd
2e6 phxentdd
2e7 phxentdd

2e8 phxentdd
2ea gxentdd
2eb gxentdd
2ec gxentdd
2ed nbc
2f9 WLM
2fa ethchandd
2fb ethchandd
2fc VMM_VWAIT EVENT
2fd RPDP:
2fe System freeze:
300 ODM EVENTS
339 ATM SIGNALING-DD -
33a if_at
340
355 PDIAGEX
38d AIO: Asynchronous I/O
38e SISADD
38f DYNAMIC RECONFIG:
393 LVM NON-I/O EVENTS
3a0 atmcm
3a5 atmsock
3a7 jatmdd
3a8 SCSESDD
3a9 dpmpdd
3aa dpmpdd
3ab dpmpdd
3ac sciedd
3af NFS: cache fs hooks
3b0 AutoFS: Client VNOP read/write routines
3b4 TMSSA Device
3b5 ecpadd
3b6 ecpadd
3b7 SECURITY:
3b8 SEC DATA:
3b9 FCDD
3c0 ecpadd
3c1 ecpadd
3c2 ecpadd
3c4 FCPS
3c5 IPCACCESS EVENT
3c6 IPCGET EVENT
3c7 MSGCONV EVENT
3c8 MSGCTL SYSTEM CALL
3c9 MSGGET SYSTEM CALL
3ca MSGRCV SYSTEM CALL
3cb MSGSELECT SYSTEM CALL
3cc MSGSND SYSTEM CALL
3cd MSGXRCV SYSTEM CALL

3ce SEMCONV EVENT
3cf SEMCTL SYSTEM CALL
3d0 SEMGET SYSTEM CALL
3d1 SEMOP SYSTEM CALL
3d2 SEM EVENT
3d3 SHMAT SYSTEM CALL
3d4 SHMCONV EVENT
3d5 SHMCTL SYSTEM CALL
3d6 SHMDT SYSTEM CALL
3d7 SHMGET SYSTEM CALL
3d8 MADVISE SYSTEM CALL
3d9 MINCORE SYSTEM CALL
3da MMAP SYSTEM CALL
3db MPROTECT SYSTEM CALL
3dc MSYNC SYSTEM CALL
3dd MUNMAP SYSTEM CALL
3de MVALID SYSTEM CALL
3df MSEM_INIT SYSTEM CALL
3e0 MSEM_LOCK SYSTEM CALL
3e1 MSEM_REMOVE SYSTEM CALL
3e2 MSEM_UNLOCK SYSTEM CALL
3e3 ecpadd
3e4 ecpadd
3e8 bbatmdd
3e9 bbatmdd
3ea bbatmdd
3f7 J2 - VNODE
3f8 J2 - PAGER
3fd vlandd
3fe vlandd
3ff vlandd
400 STTY
401 STTY STRTTY
402 STTY LDTERM
403 STTY SPTR
404 STTY NLS
405 STTY PTY
406 STTY RS
407 STTY LION
408 STTY CXMA
409 STTY SF
417 STTY VCON
45a SSA Adapter
45b SSA DASD
460 ASSERT WAIT
461 CLEAR WAIT
462 THREAD BLOCK
463 EMPSLEEP
464 EWAKEUPONE

465 THREAD_CREATE SYSTEM CALL
 466 KTHREAD_START
 467 THREAD_TERMINATE SYSTEM CALL
 468 KSUSPEND
 469 THREAD_SETSTATE
 46a THREAD_TERMINATE_ACK
 46b THREAD_SETSCHED
 46c TIDSIG
 46d WAIT_ON_LOCK
 46e WAKEUP_LOCK
 470 scentdd
 471 scentdd
 472 scentdd
 473 goentdd
 474 goentdd
 475 goentdd
 502 GSC
 503 GSC
 522 ICA: IBM Crypto Accelerator Error Traces
 523 ICA: IBM Crypto Accelerator Verbose Traces
 524 ICA: IBM Crypto Accelerator Verbose Traces
 527 UDI MANAGEMENT AGENT
 528 UDI SCSI MAPPER
 529 UDI BRIDGE MAPPER
 52a UDI NETMAPPER (
 52b UDI GIO MAPPER
 52c UDI NETWORK DRIVER
 52d UDI SCSI DRIVER
 535 TCP
 536 UDP
 537 IP
 538 IP6
 539 PCB
 590 ATM DdMain trc,
 591 ATM ERROR trc,
 592 ATM Common trc,
 593 ATM ILMI trace,
 594 ATM QSAAL trc,
 595 ATM SVC trace,
 5a0 sysldr/mods
 5a1 load/kxent
 5a2 sysldr/execld
 5a3 sysldr/errs:
 5a4 sysldr/chkpt:
 600 Pthread user scheduler thread
 603 Pthread timer thread
 605 Pthread vp sleep
 606 Pthread condition variable
 607 Pthread mutex

608 Pthread read/write lock
609 General pthread library call
60a HKWD_LIBC_MALL_COMMON
60b HKWD_LIBC_MALL_INTERNAL
707 LFTDD:
709 INPUTDD:
71f BLDD:
722 SGIODD:
72d MIRDD:
730 SONDD:
733 MOJDD:
734 USBKBD:
735 USBMSE:
736 USBOHCD:
7ff STREAMS (PSE)

Archived

Abbreviations and acronyms

AIO	Asynchronous Input Output	EPSA	Early Paging Space Allocation
AIX	Advanced Interactive Executive	ERRM	Event Response Resource Manager
API	Application Programming Interface	ESS	Enterprise Storage System
ARP	Address Resolution Protocol	EXTSHM	Extended Shared Memory
ASCII	American Standard Code for Information Interchange	FDDI	Fiber Distributed Data Interface
ATM	Asynchronous Transfer Mode	FDPR	Feedback Directed Program Restructuring
BPF	Berkeley Packet Filter	FLIH	First Level Interrupt Handler
CCTA	Central Computer and Telecommunications Agency	FORTRAN	Formula Translation
CD-ROM	Compact Disk Read-Only Memory	FRCA	Fast Response Cache Accelerator
CDT	Central Daylight Saving Time	FSRM	File System Resource Manager
COBOL	Common Business Oriented Language	FTP	File Transfer Protocol
CPID	Channel Path ID	GSA	General Services Administration
CPU	Central Processing Unit	GUI	Graphical User Interface
CRC	Cyclic Redundancy Code	HACMP	High Availability Cluster Management Program
CSECT	Code Segment	HAEM	High Availability Event Management
CSMA	Carrier Sense Multiple Access	HAGS	High Availability Group Services
CWD	Current Working Directory	HATS	High Availability Topology Services
DASD	Direct Access Storage Device	HPC	High Performance Computing
DB2®	Database 2	HPM	Hardware Performance Monitor toolkit
DDS	Dynamic Data Supplier	HTTP	Hypertext Transfer Protocol
DLPI	Data Link Provider Interface	IBM	International Business Machine
DMA	Direct Memory Access	ICA	IBM Crypto Accelerator
DNS	Domain Name Service		
DPSA	Deferred Paging Space Allocation		
EBCDIC	Extended Binary Coded Decimal Instruction Code		
EOF	End of file		

ICMP	Internet Control Message Protocol	NLS	Network Language Translation
IGMP	Internet Group Management Protocol	NOP	NO-operation
IOCTL	Input Output Controller	ODM	Object Data Manager
IP	Internet Protocol	OSPF	Open Shortest Path First
IPX	Internetwork Packet Exchange	PCB	Program Control Block
ISNO	Interface Specific Network Option	PCI	Peripheral Component Interconnect
ITSO	International Technical Support Organization	PDT	Performance Diagnostic Tool
JFS	Journalled File Systems	PFS	Physical File System
KEX	Kernel Extension	PFT	Page Frame Table
LAN	Local Area Network	PGID	Page Identifier
LFS	Local File System	PID	Process Identifier
LLC	Logical Link Control	POSIX	Portable Operating System Interface
LPID	Logical Page Identifier	SSP	Parallel Systems Support Program
LPSA	Late Paging Space Allocation	PTX	Performance Toolbox
LRU	Least Recently Used	PV	Physical Volume
LSU	Logical Storage Unit	RAID	Redundant Array of Independent Drives
LTG	Linux Technology Group	RAM	Random Access Memory
LVDD	Logical Volume Device Driver	RFC	Request for Comment
LVM	Logical Volume Manager	RMC	Resource Monitoring and Control
LVMDD	LVM Device Driver	ROM	Read-only Memory
MB	Megabyte	RPC	Remote Procedure Call
MCA	Microchannel Architecture	RPM	Rotation per Minute
MP64	64-bit Multiprocessor	RS/6000	RISC Systems/6000
MPIO	Multi Path Input Output device	RSCT	Reliable Scalable Cluster Technology
MTU	Maximum Transmission Unit	RSS	Real Storage Size
MWC	Multi-write Consistency	SCSI	Small Computer System Interface
MWCC	Multi-write Consistency Cache	SHM	Shared Memory
NBC	Network Buffer Cache	SLIH	Second Level Interrupt Handler
NDD	Network Device Driver		
NFS	Network File Systems		

SMIT	Systems Management Interface Tools
SMP	Symmetric Multiprocessor
SPMI	Systems Performance Measurement Interface
SRC	System Resource Controller
SSA	Systems Storage Adapter
SVC	Supervisory Call
TCP/IP	Transmission Control Protocol/Internet Protocol
TID	Thread ID
TLB	Translation Look-aside Buffer
TOC	Table of Contents
TTY	Teletype
UDP	User Datagram Protocol
VFS	Virtual File Systems
VMM	Virtual Memory Manager
VSID	Virtual Segment Identifier
WLM	Workload Manager

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 989. Note that some of the documents referenced here may be available only in softcopy.

- ▶ *AIX 5L Differences Guide Version 5.2 Edition*, SG24-5765
- ▶ *AIX 5L Workload Manager (WLM)*, SG24-5977
- ▶ *RS/6000 and Asynchronous Transfer Mode*, SG24-4796
- ▶ *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155
- ▶ *RS/6000 SP System Performance Tuning Update*, SG24-5340
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376
- ▶ *Understanding IBM @server pSeries Performance and Sizing*, SG24-4810

Other publications

These publications are also relevant as further information sources:

- ▶ AIX 5L publication, available in softcopy only:
 - *AIX 5L Version 5.1 Commands Reference*, SBOF-1877
 - *AIX 5L Version 5.1 Commands Reference, Volume 5*, SBOF-1857
 - *AIX 5L Version 5.1 Files Reference*
 - *AIX 5L Version 5.1 General Programming Concepts*
 - *AIX 5L Version 5.1 Kernel Extensions and Device Support Programming Concepts*
 - *AIX 5L Version 5.1 Performance Management Guide*
 - *AIX 5L Version 5.1 Performance Management Guide: Communications and Networks*

- *AIX 5L Version 5.2 Performance Management Guide: Operating System and Devices*
- *AIX 5L Version 5.1 System Management Concepts: Operating System and Devices*
- *AIX 5L Version 5.2 System User's Guide: Communications and Networks*
- *AIX 5L Version 5.2 System User's Guide: Operating System and Devices*
- *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 1*
- *AIX 5L Version 5.2 Technical Reference: Base Operating System and Extensions, Volume 2*
- *AIX 5L Version 5.2 Technical Reference: Communications, Volume 2*
- *AIX 5L Version 5.2 Technical Reference: Kernel and Subsystems, Volume 1*
- *AIX 5L Version 5.2 Technical Reference: Kernel and Subsystems, Volume 2*
- ▶ Other IBM publications
 - *Event Management Programming Guide and Reference, SA22-7354*
 - *Performance Toolbox Version 2 and 3 Guide and Reference*
 - *Resource Monitoring and Control Guide and Reference, SC23-4345*
- ▶ RFC 1180 A TCP/IP Tutorial

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM @server pSeries support page
<https://techsupport.services.ibm.com/server/support?view=pSeries>
- ▶ Request for comment pages for TCP/IP protocol specification
<http://www.rfc-editor.org>
<http://www.ietf.org/rfc.html>
- ▶ IBM Networking product page
<http://www.networking.ibm.com/netprod.html>
- ▶ AIX software support FTP site
<ftp://ftp.software.ibm.com/aix>
<ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr>
<ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/perf52/perf52.tar.Z>

- ▶ AIX 5L documentation page
http://publib16.boulder.ibm.com/pseries/en_US/infocenter/base/aix.htm
- ▶ TCP dump Web page
<http://www.tcpdump.org>
- ▶ Toolkit page from Alphaworks
<http://www.alphaworks.ibm.com/tech/hpmtoolkit>
- ▶ IBM Redbooks homepage
<ftp://www.redbooks.ibm.com/redbooks>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Index

Symbols

.files file 108
.nodes file 109
.SM_RAW_REPORT file 112
.thresholds file 109

Numerics

3dmon command 875, 909

A

access time 21
 rotational 21
 seek 21
 transfer 21
accumulated CPU time 135
active socket connections 622
Active Virtual Memory 214
adapter throughput report 91
Address Resolution Protocol, see ARP
address space 13
address space map 271
adm user 108
AIX maintenance level 120
AIX Performance Toolbox 891
AIX processes 6
AIX thread 6
alignment exception 281
allocated page range 412
allocation policies 15
alstat command 283
 examples 283
analysis interval 733
API 181, 785, 891
application profiling 335
Application Program Interface, see API
Application Programming Interfaces, see API
ARP 42
ARP cache thrashing 605
ARP handling 605
as file 49
ASCII 568
Asynchronous Transfer Mode, see ATM

ATM 539
atmstat command 33, 540, 622
 examples 541
 fields of interest 543

B

base priority 7
baseline performance information 60
bc command 593
benchmarking program 302
Berkeley Packet Filter, see BPF
bindintcpu command 9, 290
bindprocessor command 9, 137, 292
biod 218
block device activity 153
block I/O daemon 218
bos.acct 81, 211, 355
bos.adt.prof 298
bos.adt.samples 165, 229
bos.mp 289
bos.net.nfs.client 655
bos.net.tcp.server 567
bos.perf.diag_tool 105
bos.perf.tools 93, 139, 179, 281, 298–299, 379,
387, 457, 677, 703, 729
bos.perf.tune 165, 229, 255
bos.rte.control 349, 365
bos.rte.lvm 501, 519
bos.rte.misc_cmds 355
bos.sysmgmt.serv_aid 191
bosboot command 45, 230, 719
BPF 571
buffer activity 150
buffer size 671
buffer utilization 150

C

C program 786, 806, 820
cache coherency 9
caveat 819
cc command 288, 786
CC_IN 602
chcondition command 825

chdev command 544, 629
 child processes 276
 chresponse command 825
 chsrc command 825
 client segments 13, 392, 409
 clock interrupt 6
 clock-algorithm 225
 commands
 3dmon 875, 909
 alstat 283
 atmstat 33, 540, 622
 bc 593
 bindintcpu 9, 290
 bindprocessor 137, 292
 bosboot 45, 230, 719
 cc 288, 786
 chcondition 825
 chdev 544, 629
 chresponse 825
 chsrc 825
 cronadm 108
 ctsnap 825
 curt 678
 date 197
 dd 198, 450, 494
 defragfs 481
 df 47, 494
 Driver_ 108
 du 493
 emstat 285
 entstat 33, 546, 622
 estat 33, 552
 fddistat 33, 555, 622
 fdpr 71–72
 filemon 219, 458, 508, 512, 705
 fileplace 64, 464, 480, 512
 frcactrl 623
 fsck 29
 ftp 33, 98
 genkex 713
 genkld 712
 genld 710
 gennames 94, 678, 704, 730–731, 734
 gensyms 327, 715, 731, 734
 gprof 300, 689
 inetd 45
 ioo 18, 239
 iostat 64, 82, 340, 505, 660, 787
 ipcrm 365, 808
 ipcs 366, 808
 ipfilter 34, 573
 ipreport 34, 572
 iptrace 34, 569, 633
 jazizo 918
 kill 808
 locktrace 720
 logform 29
 lsactdef 825
 lsattr 35, 39, 91, 544, 629
 lsaudrec 825
 lscondition 825
 lscondresp 825
 lsfs 464, 525
 lslv 87, 463, 502, 523
 lspv 212, 463, 502
 lsresponse 825
 lsrsrc 825, 830
 lsrsrcdef 825
 lsvg 463, 503
 lvmstat 508, 520
 make 208, 805
 migratelp 523
 mkcondition 825
 mkcondresp 826
 mkresponse 825
 mkrsrc 825
 mkvg 24, 503
 mount 47, 663
 ncheck 396
 netpmon 94
 netstat 33, 620
 nfso 646
 nfsstat 33, 656
 nice 6, 350
 no 34, 623, 666
 odmget 540
 pdt_config 106
 perfpmr 116
 ping 33
 pprof 309
 proccred 270
 procfiles 268
 procflags 270
 procldd 272
 procmmap 271
 procrun 275
 procsig 273
 procstack 274

procstop 275
proctree 276
procwait 276
procwdx 268
prof 305, 320
ps 6, 49, 128, 351
refrsrc 825
renice 6, 352
rmaudrec 825
rmcctrl 825, 829
rmcondition 825, 841
rmcondresp 826
rmresponse 825, 841
rmrsrc 825
rmss 380, 399
route 620, 628
rstatd 849
sadc 142
sar 9, 62, 140, 219, 375
schedo 7, 15, 166, 219
schedtune 177
slibclean 365, 808
snap 32
splat 730
ssaraid 517
ssaxlate 516
startcondresp 826
stopcondresp 826
stripnm 704, 715, 724
svmon 369, 388
svmon_back 387
sync 481
syncd 244
syncvg 28
tcpdump 34, 587, 633
timex 357
tokstat 33, 560, 622
topas 53, 180, 369
tprof 61, 283, 324, 376, 704–705, 715
trace 62, 760
traceroute 33
trcevgrp 95
trcnm 775
trcoff 760
trcon 760
trcrpt 680, 777
trcstop 760
truss 192, 376
tunchange 44
tuncheck 44, 256
tundefault 44
tunrestore 44, 258
tunsave 44
vmo 12, 16, 215, 230
vmstat 157, 212, 245, 291, 340, 544, 660, 787
vmtune 54, 251
wlmmon 872
wlmperf 872
wlmstat 862
xmperf 875, 894
xmtrend 874, 918
xmwlm 873
communication direction 590
compiling 786, 806, 820
complex kernel lock 738
computational pages 233
condition 836
condition-variable 739
context switching 162, 217
CPU consuming processes 131
CPU context switch 226
CPU decay factor 7
CPU overhead 764
CPU penalty factor 7
CPU performance 5
CPU throughput 5
CPU ticks 356
CPU usage 335, 339
CPU Usage Reporting Tool 677
CPU usage value 6–7
cred data structure 52
credentials 270
cronadm command 108
crontab 106, 146
crontab file 108
ctsnap command 825
cumulative CPU time 737
curt command 678
 additional information 696
 application summary by process ID 689
 application summary by process type 689
 application summary by thread ID 687
 default report 683
 detailed process information 700
 detailed thread status 698
 errors by system calls 697
 FLIH summary 693
 general information report 683

- Kproc summary by thread ID 690
- pending system calls summary 692
- processor summary report 686
- SLIH summary 694
- system calls summary 691
- system summary report 684
- trace hooks 679

cwhet.c 968

Cylinder 21

D

Data Link Provider Interface, see DLPI

date command 197

dd command 198, 450, 494

DDS 808

Dead Man Switch, see DMS

decrementer interrupt 326

Deferred Paging Space Allocation, see DPSA

defps 16

defragfs command 481

destination address 615

devices.chrp.base.rte 289

devices.common.IBM.atm.rte 539

devices.common.IBM.ethernet.rte 539

devices.common.IBM.fddi.rte 539

devices.common.IBM.tokenring.rte 539

df command 47, 494

Direct Memory Access, see DMA

disk I/O 18

- access time 21
- design approach 19

disk placement 523

disk striping 249

disk utilization report 89

dispatching priority 349

DLPI 623

DMA 546

DMS 123

DNS 577

DNS tracing 577

document organization 2

Domain Name Server, see DNS

DPSA 16, 234

Driver_ command 108

du command 493

dudestat.c 965

duplex communication 590

dynamic data supplier, see DDS

dynamic libraries 272

E

Early Paging Space Allocation, see EPSA

EBCDIC 568

ECHO_REPLY package 574

ECHO_REQUEST package 574

emstat command 285

- examples 286

emulation exception 281

Enhanced Journaled File System, see JFS2

Enterprise Storage Server, see ESS

entstat command 33, 546, 622

- examples 548
- fields of interest 549

environment variables 205

- PSALLOC 15
- SPINLOOPTIME 11
- YIELDLOOPTIME 11

EPSA 15

ERRM 827

ESS 90

estat command 33, 552

- examples 552
- fields of interest 553

Ethernet device driver 539

Event Response resource manager, see ERRM

event response script 833

execution interval 733

expectations 4

Extended Common Object File Format, see XCOFF

extended shared memory 17

extendednetstats 630

extra nice value 7

F

F80 2

facility access modes 368

failovers 123

fault name 197

FD 459

FDDI 539

fddistat command 33, 555, 622

- examples 556
- fields of interest 558

fdpr command 71–72

- example 76
- instrumentation 76

- phases 75
 - source code 76
- Fiber Distributed Data Interface, see FDDI
- file access system routines 149
- file descriptor, see FD
- file descriptors 268
- file pages 233
- file system caching 233
- File System resource manager, see FSRM
- filemon command 219, 458, 508, 512, 705
 - access pattern analysis 460
 - detailed file report 470
 - examples 462
 - file report 468
 - file summary section 469
 - fragmentation analysis 461
 - I/O activity 459
 - interpreting reports 460
 - logical volume detail 463
 - monitoring 462
 - most active files 464
 - most active logical volumes 463, 472
 - most active physical volumes 462
 - Most Active Segments report 461
 - physical volume detail 466
 - physical volume reports 464
 - physical volume summary 466
 - trace hooks 460
 - virtual memory segments report 475
- filemon.out file 462
- fileplace command 64, 464, 480, 512
 - examples 481
 - indirect block report 486
 - logical report 482
 - physical address 484
 - physical mapping 499
 - physical report 483
 - volume report 488
- filesystems 507
- First Level Interrupt Handler, see FLIH
- fixed_pri_global 8, 170
- FLIH 98, 682
- FLIH CPU statistic 101
- floating point double load 282
- fork 219
- fork report 219
- four way close 599
- fragment size 485
- fragmentation 461

- fractrl command 623
- free list 14
- fsck command 29
- FSRM 827
- ftp command 33, 98

G

- gaps 491
- genkex command 713
- genkld command 712
- genld command 710
- gennames command 94, 678, 704, 730–731, 734
 - file system information 709
 - loaded kernel extensions 707
 - loaded libraries per process 711
 - loaded processes 708
 - loaded shared libraries 707
 - name-to-address mapping 706
- gensyms command 327, 715, 731, 734
 - name-to-address mapping 715
- gprof command 300, 689
 - cross-reference index 306
 - detailed function report 302
 - flat profile report 305
- Graphical User Interface, see GUI
- GUI 298, 824

H

- HACMP 123
- HAEM 826
- hd_pbuf_cnt 22, 246
- hdisk 517
- Head 21
- hit ratios 140
- HKWD_KERN_PIDSIG 680
- HKWD_KERN_SVC 680
- hook ID 765
- human expectations 3

I

- I/O activity levels 459
- I/O bottleneck 87
- I/O operations 359
- ICMP 96, 634
- ICMP packets 609
- ICMP tracing 586
- ifconfig command 35

- iget routines 149
- IGMP 634
- illegal instruction program 282
- indirect block 487
- inetd command 45
- inittab 258
- inode 493
- inode lookup routines 149
- inode number 50
- inode table 480
- instantiation 807
- Inter Process Communication, see IPC
- inter-disk allocation policy 26
- Interface Specific Network Options, see ISNO
- Internet Control Message Protocol, see ICMP
- Internet Group Multicast Protocol, see IGMP
- interrupt handlers 9
- interrupt level 290
- interrupt priority 220
- interrupt redirection 290
- intra-disk allocation policy 25
- ioo command 18, 239
- ioo tunable 239
 - hd_pbuf_cnt 22, 246
 - j2_maxRandomWrite 245
 - j2_nBufferPerPagerDevice 246
 - j2_nPagesPerWriteBehindCluster 244
 - j2_nRandomCluster 245
 - lvm_bufcnt 246
 - maxpgahead 241–242
 - maxrandwrt 244
 - minpgahead 242
 - numclust 241
 - numfsbuf 241
 - numfsbufs 245
 - pd_npages 246
 - sync_release_ilock 245
- iostat command 64, 82, 340, 505, 660, 787
 - adapter throughput report 91
 - disk utilization report 89
 - reports 83
 - system throughput report 84
 - tty and CPU utilization report 88
- IPC 155
- IPC message queue 367
- ipcrm command 365, 808
- ipcs command 366, 808
 - examples 366
 - IPC message queues 367

- semaphores 375
 - shared memory 368
- ipfilter command 34, 573
- ipforwarding 671–672
- ipreport command 34, 572
- iptrace command 34, 569, 633
- IPX tracing 586
- ISNO 31, 34, 637
- IY43857 711

J

- j2_maxRandomWrite 245
- j2_nBufferPerPagerDevice 246
- j2_nPagesPerWriteBehindCluster 244
- j2_nRandomCluster 245
- jazizo command 918
- JBOD 517
- JFS 87, 392, 507
- JFS inode table 480
- JFS log 29
- JFS superbloc 480
- JFS2 507, 516
- JFS2 client pages 235
- Journalized File System, see JFS
- Just a Bunch Of Disks, see JBOD

K

- kernel 5
 - operation 5
- kernel block buffer cache 140
- kernel extension 9, 142, 713
 - name 705
- kernel mode 9
- kernel process activity 154
- kernel processes 9, 217
- kernel scheduling queue 156
- kernel service 292
- kernel thread state changes 214
- kernel threads 309
- kernel table utilization 160
- kill command 808

L

- Late Paging Space Allocation, see LPSA
- Least Recently Used, see LRU
- lgpg_regions 238
- lgpg_size 238

- libperfstat.a 786
- libperfstat.h 786
- libpmapi.a 820
- libSpmi.a 806
- limbo state 545
- linking 786, 806, 820
- loaded kernel extension 705
- loader entry 712
- loading dynamic library 272
- lock 10
- lock classes 720
- lock types 10, 742
 - mutual exclusion locks 11
 - read-write locks 11
 - sleeping locks 11
 - spin locks 10
- lockname.h file 720
- locks usage 720
- locktrace command 720
 - lock usage 720
- logform command 29
- logical file system 459
- logical fragment numbers 488
- logical fragmentation 481
- logical partitions 523
- logical processor 292
- Logical Track Groups 28
- Logical Volume Device Driver, see LVDD
- Logical Volume Manager Device Driver, see LVM-DD
- Logical Volume Manager, see LVM
- logical volume report 471
- logical volume striping 249
- logical volume utilization 522
- logical volumes 459, 507, 705
- LPSA 15, 234
- LRU 171, 225
- lrud 446
- lrud kernel process 14
- lsactdef command 825
- lsattr command 35, 39, 91, 544, 629
- lsaudrec command 825
- lscondition command 825
- lscondresp command 825
- lsfs command 464, 525
- lslv command 87, 463, 502, 523
 - examples 505
 - usage 512
- lspv command 212, 463, 502
 - examples 505
 - usage 513
- lsresponse command 825
- lsrsrc command 825, 830
- lsrsrcdef command 825
- lsvg command 463, 503
 - examples 505
 - usage 515
- LVDD 28, 484
- LVM 22, 24, 501
- lvm_bufcnt 246
- LVMD 510, 520
- lvmstat command 508, 520
 - examples 521
 - logical volume utilization 522
 - monitoring logical volume 526
 - monitoring logical volumes 524
 - summarizing I/O utilization 528

M

- MAC 581
- MAC address 606
- machine fault numbers 194
- Mail Handler, see MH
- maintenance level 120
- major device number 50
- make command 208, 805
- mapping segment 415
- mapping segments 392
- maxclient% 235
- maxfree 14, 219, 231–232
- Maximum Segment Size, see MSS
- Maximum Transfer Unit, see MTU
- maxperm 233
- maxperm% 231
- maxpgahead 241–242
- maxpin 233
- maxrandwrt 244
- maxspin 11, 170
- MCA 541
- Media Access Control, see MAC
- memory consuming processes 132
- memory leak 394
- memory leaks 17
- memory pinning 233
- memory pools 232
- memory segment 13, 392

- types 13
- memory utilization 393
 - per user 400
- mempools 232
- message utilization 155
- MH 840
- MicroChannel Adapter, see MCA
- microprofiling 324, 336
- migratelp command 523
- minfree 14, 219, 231–232, 446
- minor device number 50
- minperm 233
- minperm% 231
- minpgahead 242
- mirror write consistency 28
- mkcondition command 825
- mkcondresp command 826
- mkresponse command 825
- mkrsrc command 825
- mkvlg command 24, 503
- mode switching 9
- module handler 221
- mon.sum file 323
- mount command 47, 663
- mounted file system 662
- MPIO 90
- msgget subroutine 367
- msgrcv system call 368
- msgsnd system call 368
- MSS 607
- MTU 31, 671
- MTU sizes 671
- multi-path input-output, see MPIO
- mutex 739
- mutual exclusion locks 11
- MWC Check 28
- MWC record 28

N

- name resolution 611
- name-to-address mapping 705, 715
- NBC 622, 642
- nbc_limit 642
- nbc_max_cache 643
- nbc_min_cache 643
- nbc_pseg 643
- nbc_pseg_limit 643
- ncheck command 396
- NDD 540, 547, 556, 561
- ndd_genstats 547
- netpmn command 94
 - detailed statistics 103
 - example 96
 - FLIH and SLIH CPU statistics 101
 - process statistics 99
 - TCP socket call statistics 102
 - trace hooks 95
- netstat command 33, 620
 - communications subsystems statistics 638
 - examples 624
 - kernel malloc statistics 628
 - network buffer cache 642
 - network interfaces 624
 - network routing 627
 - protocol statistic 632
 - state of all sockets 640
- network buffer cache, see NBC
- network buffer size 671
- Network Device Driver, see NDD
- Network File System, see NFS
- network interface layer 584
- network layer 584
- network MTU sizes 671
- network ports 570
- network protocol statistics 632
- network routes 622
- network services 570
- network traffic 568
- network tuning 31
- NFS 95, 218, 645, 655
- NFS client pages 235
- NFS clients 662
- NFS tracing 584
- nfs_dynamic_retrans 651
- nfso command 646
 - examples 648
- nfso tunable
 - nfs_dynamic_retrans 651
- nfstat command 33, 656
 - client NFS statistics 661
 - mounted file systems 662
 - NFS statistics 659
 - RPC statistics 657, 660
- nice command 6, 350
 - decreasing the nice value 352
 - improving priority 352
 - increasing the nice value 352

- reducing priority 352
- nice value 6, 350, 352
- niced priority 7
- no command 34, 623, 666
 - buffer size 671
 - MTU sizes 671
 - permanent change 671
- no fragment flag 597
- no tunable
 - extendednetstats 630
 - ipforwarding 672
 - mtu 671
 - nbc_limit 642
 - nbc_max_cache 643
 - nbc_min_cache 643
 - nbc_pseg 643
 - nbc_pseg_limit 643
 - rfc1323 671
 - sb_max 671
 - sockthresh 631
 - subnetsarelocal 609
 - tcp_nagle_limit 637
 - tcp_pmut_discover 628
 - tcp_recvspace 670–671
 - tcp_sendspace 671
 - udp_pmtu_discover 628
 - use_isno 34
- nointegrity 29
- nokilluid 16
- non-arbitrated loop protocol 20
- non-preemptive scheduling 8
- npskill 16, 231
- npswarn 16, 231
- numclust 241
- numfsbuf 241
- numfsbufs 245

O

- Object Data Manager, see ODM
- ODM 501, 540, 547
- odmget command 540
- offline processing 703
- Open Shortest Path First, see OSPF
- optimized executable 75
- OSPF 610
- Other subroutines 842

P

- pacefork 16, 170
- packet-sequencing information 616
- page fault 14
- Page Frame Table, see PFT
- page frames 13
- page in 136
- page reclaims 215
- page replacement algorithm 14, 232
- page stealer 14, 216
- page-replacement algorithm 216
- pages out 215
- paging space 12, 30
- paging space usage 394
- paging statistic 157
- Parallel System Support Programs, see PSSP
- Path Maximum Transfer Unit, see PMTU
- pbuf 22
- PCB, see protocol control block
- PCI 541
- PCI bus 20
- pd_npages 246
- PDT 105
 - .files file 108
 - .nodes file 109
 - .SM_RAW_REPORT file 112
 - .threshold file 109
 - configuration files 108
 - manual collection 114
 - report 111
- PDT directories 108
- PDT files 108
- pd_config command 106
 - interface 107
- penalized processes 133
- perfagent.tools 71
- perfmgr.analysis.jazizo 872
- performance
 - response time 4
 - throughput 4
- Performance AIDE 892
- performance concept 3
- Performance Diagnostic Tool, see PDT
- Performance Monitor, see PM
- Performance Toolbox, see PTX
- performance tuning 3
- perfpmr command 32, 116
 - filesets 120
 - installation 122

- PROBLEM.INFO file 124
- perfpnr files
 - config.sh 116
 - emstat.sh 116
 - filemon.sh 117
 - iostat.sh 117
 - iptrace.sh 117
 - monitor.sh 117
 - netstat.sh 118
 - nfsstat.sh 118
 - pprof.sh 118
 - ps.sh 118
 - sar.sh 119
 - tcpdump.sh 119
 - tprof.sh 119
 - trace.sh 119
 - vmstat.sh 119
- Perfstat API
 - compiling and linking 786
 - interface types 801
- perfstat kernel extension 786
- perfstat_cpu subroutine 788
- perfstat_cpu_total subroutine 790
- perfstat_disk subroutine 795
- perfstat_disk_total subroutine 797
- perfstat_diskadapter subroutine 805
- perfstat_diskpath subroutine 805
- perfstat_dude.c 940
- perfstat_dump_all.c 936
- perfstat_memory_total subroutine 793
- perfstat_netbuffer subroutine 805
- perfstat_netinterface subroutine 799
- perfstat_netinterface_total subroutine 802
- perfstat_pagingspace subroutine 805
- perfstat_protocol subroutine 805
- perfstat_reset subroutine 805
- Peripheral Component Interconnect, see PCI
- persistent segments 13, 392, 410
- PFS 765, 774
- PFT 14, 215, 446
- PGIN value 136
- physical disk 517
- Physical File System, see PFS
- physical fragment numbers 490
- physical fragmentation 481
- physical partition 24, 507, 528
- physical processor 292
- physical volume 24, 459, 705
- PID 102
- ping command 33
- pinned pages 407
- pinning memory 233
- PM 785
- PM API 818
 - compiling and linking 820
- PMTU 627
- ppe.xprofiler 298
- pprof command 309
 - pprof.famcpu report 319
 - pprof.famind report 316
 - pprof.namecpu report 315
 - pprof.start report 313
 - reports 310
 - trace hooks 311
- pprof.cpu 310
- pprof.famcpu 311
- pprof.famind 310
- pprof.namecpu 310
- pprof.start 310
- PR_REQUESTED event 275
- proccred command 270
- process address space map 271
- process addresses 274
- process credentials 270
- process dynamic libraries 272
- process file descriptors 268
- process identification, see PID
- process scheduling 5
- process stack frames 274
- process tracing flags 270
- process tree 276
- process working directory 268
- processes 6
- processor affinity 10
- processor utilization 158
- procfiles command 268
- procflags command 270
- procldd command 272
- procmap command 271
- procrun command 275
- procsig command 273
- procstack command 274
- proctree command 275
- proctree command 276
- procwait command 276
- procwdx command 268
- prof command 305, 320
 - mon.sum 323

- summary report 323
- Program Temporary Fix, see PTF
- protocol control block 613, 617
- protocol types 570
- ps command 6, 49, 128, 351
 - Berkeley standard 128
 - CPU consuming processes 131
 - displaying threads 138
 - memory consuming processes 132
 - penalized processes 133
 - PGIN value 136
 - RSS value 135
 - X/Open standard 128
- PSALLOC 15
- PSSP 539, 824
- PTF 120
- Pthread condition-variable 738
- Pthread mutex 738
- Pthread read/write lock 738
- PTX 872

R

- RAID 24
- random access 460
- Random write-behind 244
- RAW I/O 246
- raw trace 680
- read file descriptors 202
- read-locking 746
- read-write locks 11
- real memory 12, 392
- real memory map 392
- real memory pages 408
- real memory usage 394
- recursive locking 746
- Reduced-Memory System Simulator 379
- Redundant Array of Independent Disks, see RAID
- refsrc command 825
- Regatta 1
- Remote Procedure Call, see RPC
- renice command 6, 352
- reserved paging space 406
- resource bottleneck 340
- resource class 827
- resource manager 826
- Resource Monitoring and Control, see RMC
- resource utilization 862
- response time 4

- rfc1323 671
- rmaudrec command 825
- RMC 824
 - activating monitoring 838
 - active WLM classes 854
 - associating response with condition 837
 - condition
 - creation 836
 - event response creation 837
 - event response script 833
 - examples 843
 - resource class 827
 - resource manager 826
 - RMC
 - listing 836
 - vmgetinfo 845
- rmctrl command 825, 829
- rmcondition command 825, 841
- rmcondresp command 826
- rmresponse command 825, 841
- rmsrc command 825
- rmss command 380, 399
 - changing memory size 383
 - displaying memory size 383
 - examples 382
 - resetting memory size 383
 - testing executable run time 383
- route command 620, 628
- RPC 96, 572, 655
- rpoolsz 554
- rsct.basic 824
- rsct.compat.basic 824
- rsct.compat.clients 824
- RSS value 135
- rstatd command 849
- run queue 8
- RunQ lock 738

S

- sadc command 142
- sar command 9, 62, 140, 219, 375
 - combining option 144
 - file access system routines 149
 - monitoring block device 153
 - monitoring buffer activity 150
 - monitoring buffer utilization 150
 - monitoring context switching 162
 - monitoring kernel process 154

- monitoring kernel scheduling 156
- monitoring kernel table 160
- monitoring message utilization 155
- monitoring one CPU 142
- monitoring processor utilization 158
- monitoring semaphore utilization 155
- monitoring system calls 151
- monitoring tty device 160
- paging statistic 157
- using crontab entries 146
- sb_max 671
- sched_D 167
- SCHED_FIFO 8
- SCHED_FIFO2 8
- SCHED_FIFO3 8
- SCHED_OTHER 8
- sched_R 167
- SCHED_RR 7
- schedo command 7, 15–16, 166, 219
- schedo tunable 166
 - fixed_pri_global 8, 170
 - maxspin 11, 170
 - pacefork 170
 - sched_D 167
 - sched_R 167
 - timeslice 8, 170
 - v_exempt_secs 172
 - v_min_process 167, 172
 - v_repage_hi 167, 171, 381
 - v_repage_proc 172
 - v_sec_wait 168, 172
- schedtune command 177
- scheduling 5
- scheduling policies 7
- scheduling queue 156
- SCSI 20
- SD 835
- Second Level Interrupt Handler, see SLIH
- Sector 20
- segment 13, 392
- segmentation layout 393
- SEM_LOCK_CLASS lock type 721
- semaphore utilization 155
- semaphores 375
- sequential access 460
- sequential reads and writes 774
- Sequential write-behind 244
- sequentiality 489
- Serial Storage Architecture, see SSA
- serialize access 12
- SF_SYNC_CACHE 622
- SF_SYNC_CACHE flag 642
- shared memory 17, 368
- shared memory program 372
- shared memory segment 369–370, 395
- shared objects 712
- shared segment 403
- SHM_LGPAGE 234
- SHM_PIN flag 233
- short IP packets 607
- showing process tree 276
- SIGDANGER signal 16
- SIGKILL signal 16
- signal actions 273
- signal name 197
- signals 195
- simple kernel lock 738
- Simple Performance Lock Analysis Tool 729
- simplex communication 590
- sleeping locks 11
- slibclean command 365, 808
- SLIH 98, 682
- SLIH CPU statistic 101
- SMIT 545, 781
- SMP 9, 289
- SMP machines 5
- snap command 32
- sockthresh 631
- source address 615
- source code 935
- source segments 415
- SP Switch 571
- SP switch device driver 539
- space efficiency 489
- sparse file 490, 492
 - creation 494
 - determining 494
 - finding 498
 - large file enabled filesystems 496
- spin locks 10
- SPINLOOPTIME 11
- splat command 730
 - AIX kernel lock details 739
 - analysis interval 733
 - complex-lock report 746
 - execution interval 733
 - execution summary 735
 - function detail report 743

- gross lock summary report 736
- lock details 739
- mutex reports 750
- per-lock summary report 737
- PThread synchronizer reports 749
- read/write lock reports 752
- thread detail report 745
- trace discontinuities 733
- trace hooks 732
- trace interval 733
- SPMI 181, 369, 805, 891
- SPMI API
 - basic program layout 821
 - compiling and linking 806
 - data organization 806
 - makefile 817
- SPMI hierarchy 816
- spmi_data.c 953
- spmi_dude.c 949
- spmi_file.c 959
- spmi_traverse.c 961
- SpmiCreateStatSet subroutine 809
- SpmiFreeStatSet subroutine 812
- SpmiGetValue subroutine 811
- Spmilnit subroutine 809
- SpmiNextVals subroutine 812
- SpmiPathAddSetStat subroutine 810
- SpmiPathGetCx subroutine 809
- spoolsize 554
- SRC 571, 583
- SSA 20, 22
- ssaraid command 517
- ssaxlate command 516
- ssp.css 539
- stack frames 274
- stale partition 509
- standard deviation 460
- startcondresp command 826
- stopcondresp command 826
- Strict 26
- strict_maxperm 234
- stripnm command 704, 715, 724
 - examples 725
- structured data, see SD
- subnetsarelocal 609
- superblock 480
- superstrict 26
- svmon command 17, 369, 388
 - allocated page ranges 412
 - allocated page ranges by process 421
 - allocated page ranges for a command 436
 - client segments 409
 - client segments by process 419
 - client segments for a command 432
 - detailed reports 446
 - displaying persistent segment 396
 - examples 393
 - frame reports 448
 - global report 398
 - mapping segment 415
 - mapping segment for a command 424
 - memory utilization per user 400
 - monitoring frame reuse 450
 - most utilized segments 395
 - non-system segments 411
 - non-system segments by process 421
 - non-system segments for a command 435
 - paging space usage 394
 - persistent segments 410
 - persistent segments by process 419
 - persistent segments for a command 433
 - pinned pages 407
 - pinned pages by process 418
 - pinned pages for a command 430
 - processes by user 403
 - processes reports 413
 - real memory pages 408
 - real memory pages by process 418
 - real memory pages for a command 431
 - real memory usage 394
 - reserved paging space by process 417
 - reserved paging space for a command 429
 - reserved paging space pages 406
 - segment utilization 438
 - segments usage of paging space 396
 - source segment 415
 - source segment for a command 424
 - system segments 411
 - system segments by process 420
 - system segments for a command 435
 - time interval monitoring 447
 - virtual pages by process 417
 - virtual pages for a command 428
 - virtual space 405
 - WLM class memory usage 395
 - working segment 410
 - working segments by process 420
 - working segments for a command 433

- svmon_back command 387
- swapqry 848
- symbol names
 - list 776
- symbolic names 274
- Symmetrical Multiprocessor, see SMP
- sync command 481
- sync_release_ilock 245
- syncd command 244
- syncd daemon 245
- syncvg command 28
- Syscall name 197
- system call statistics 151
- System Management Interface Tool, see SMIT
- system memory 12
- System Performance Measurement Interface, see SPMI
- System Resource Controller, see SRC
- SYSTEM SEGMENT 425
- system segments 411
- system throughput report 84

T

- TCP 33, 96, 568, 656
- TCP socket call statistic 102
- TCP tracing 575, 585
- TCP/IP traffic tracing 583
- tcp_nagle_limit 637
- tcp_pmut_discover 628
- tcp_recvspace 670–671
- tcp_sendspace 671
- tcpdump command 34, 587, 633
 - abbreviations 592
 - device types primitives 591
 - examples 594
 - expressions 590
 - ICMP packet 609
 - interpreting link-level headers 604
 - monitoring all packets 603
 - monitoring ARP packets 604
 - monitoring of TCP connections 607
 - monitoring TCP 595
 - monitoring UDP packets 599
 - packet data 593
 - protocol abbreviation 592
 - relational expressions 592
 - transfer direction primitives 591
 - using expressions 606
- thrashing 15
- thread 6
 - base priority 7
- thread aging 6
- thread information 138
- thread scheduling 5
- thread state 742
- thread_create kernel service 292
- threads concept 5
- Threads ID, see TID
- three-way handshake 597
- throughput 4
- throughput data 33
- TID 138
- timeslice 8, 170
- timex command 357
- TLB 234
- Token-ring device driver 539
- tokstat command 33, 560, 622
 - examples 562
 - fields of interest 563
- topas command 53, 180, 369
 - CPU statistics 188
 - CPU utilization 182
 - disk drive statistics 183
 - examples 181
 - file statistic 184
 - hot processes 183
 - memory statistics 185
 - monitoring CPU usage 187
 - monitoring disk problem 189
 - network statistics 182
 - paging statistics 184
 - system global events 183
 - system queues 183
 - tty statistic 184
 - WLM statistics 183
- tprof command 61, 283, 324, 376, 704–705, 715
 - application profiling 335
 - detecting resource bottleneck 340
 - examples 329
 - global profiling 330
 - manual offline processing 333
 - offline profiling 332
 - online profiling 330
 - post-processing 333
 - process level profiling 335
 - report 329
 - trace hook 234 327

- trace buffer 764–766
- trace command 62, 760
 - CPU overhead 764
 - data collection 764
 - examples 770
 - INTERRUPT signals 763
 - return times 770
 - running asynchronously 768
 - running interactively 768
 - sequential read and write 774
 - subcommands 763
 - tracing a command 769
 - tracing to log file 769
- trace facility 765
- trace hook 765
 - list 973
- trace hook function 766
- trace interval 733
- trace log file 766
- traceroute command 33
- tracing
 - DNS 577
 - ICMP 586
 - IPX 586
 - NFS 584
 - TCP 575, 585
 - TCP/IP traffic 583
 - UDP 576, 586
- tracing flags 270
- Track 20
- Translation Look-Aside Buffer, see TLB
- Transmission Control Protocol, see TCP
- transport layer 584
- trcevgrp command 95
- trcnm
 - symbol names 776
- trcnm command 775
 - examples 776
- trcoff command 760
- trcon command 760
- trcrpt command 680, 777
 - combining trace buffers 781
- trcstop command 760
- trpt command
 - stored trace records 614
- truss command 192, 376
 - analyzing file descriptor I/O 202
 - checking environment variable 205
 - checking library call 209

- combining flags 204
- examples 197
- machine fault list 194
- monitoring running processes 200
- read file descriptors 202
- signal list 195
- summary output 198
- tracking child processes 206
- write file descriptors 203
- tty and CPU utilization report 88
- tty device utilization 160
- tunable 166, 230, 239
- tunchange command 44
- tuncheck command 44, 256
 - validation 256
- tundefault command 44
- tunrestore command 44, 258
 - limitation 258
- tunsave command 44

U

- UDP 33, 96, 656
- UDP tracing 576, 586
- udp_pmtu_discover 628
- UltraSCSI 20
- unallocated logical blocks 483
- unknown files 461
- unused shared memory segment 370
- unverified 819
- use_isno 34
- User Datagram Protocol, see UDP
- user library call 209
- user mode 9

V

- v_exempt_secs 172
- v_min_process 167, 172
- v_pinshm 238
- v_repage_hi 167, 171, 381
- v_repage_proc 172
- v_sec_wait 168, 172
- vario structure 843
- verified 819
- virtual file system 705
- virtual memory 392
- virtual memory activity 213
- Virtual Memory Manager, see VMM
- virtual memory system 459

- vmgetinfo 845
- VMM 12, 215, 231, 234, 242, 403, 842
- VMM write-behind 244
- vmo command 12, 14, 16, 215, 230
- vmo tunable 230
 - defps 16
 - lgpg_regions 238
 - lgpg_size 238
 - maxclient% 235
 - maxfree 14, 219, 231–232
 - maxperm 233
 - maxperm% 231
 - maxpin 233
 - mempools 232
 - minfree 14, 219, 231–232, 446
 - minperm 233
 - minperm% 231
 - nokilluid 16
 - npskill 16, 231
 - npswarn 16, 231
 - strict_maxperm 234
 - v_pinshm 238
- vmstat command 9, 157, 212, 245, 291, 340, 544, 660, 787
 - examples 213
 - fork report 219
 - I/O report 226
 - interrupt report 220
 - sum structure report 224
 - virtual memory activity 213
 - VMM statistics 221
- vmtune command 54, 251
- vnode address 149
- volume group 24, 507

W

- Whetstone benchmark 968
- WLM 128, 137, 774, 861
 - active mode 862
 - memory usage 395
 - passive mode 862
- wlm_bio_class_info_t 856
- wlm_get_bio_stats subroutine 856
- wlm_get_info subroutine 853, 865
- wlmmon command 872
- wlmperf command 872
- wlmstat command 862
 - examples 865

- working directory 268
- working segment 13, 410
- working segments 392
- Workload Manager, see WLM
- write activity throughput 251
- write file descriptors 203
- write-locking 746
- write-verify policy 28

X

- XCOFF 727
- XLATE ioctl operation 484
- xmperf command 875, 894
- xmtrend command 874, 918
- xmtrend daemon 872–873
- xmwlm command 873
- xmwlm daemon 873
- xnice factor 7

Y

- YIELDLOOPTIME 11



Redbooks

AIX 5L Performance Tools Handbook

(1.5" spine)

1.5" x 1.998"

789 <-> 1051 pages



AIX 5L Performance Tools Handbook



Redbooks

Efficient use of AIX 5L performance monitoring and tuning tools

This IBM Redbook takes an insightful look at the performance monitoring and tuning tools that are provided with AIX 5L. It discusses the use of the tools as well as the interpretation of the results in many examples.

In-depth understanding of AIX system performance issues

This book is meant as a reference for system administrators and AIX technical support professionals so they can use the performance tools efficiently and interpret the outputs when analyzing AIX system performance.

Statistical report interpretation explained

A general concept and introduction to the tools is presented to introduce the reader to the process of AIX performance analysis.

The individual performance tools discussed in this book fall into these categories:

- Multi-resource monitoring and tuning tools
- CPU-related performance tools
- Memory-related performance tools
- Disk I/O-related performance tools
- Network-related performance tools
- Performance tracing tools
- Additional performance topics, including performance monitoring API, Workload Manager tools, and performance toolbox for AIX.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks