IBM

AIX 5L Practical Performance Tools and Tuning Guide

- Updated performance information for IBM @server p5 and AIX 5L V5.3
- New tools for @server p5 with SMT and Micro-Partitioning
- Practical performance problem determination examples

Kumiko Hayashi
Kangkook Ji
Octavian Lascu
Hennie Pienaar
Susan Schreitmueller
Tina Tarquinio
James Thompson

Redbooks

ibm.com/redbooks

International Technical Support Organization

AIX 5L Practical Performance Tools and Tuning Guide

April 2005

SG24-6478-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (April 2005)

This edition applies to Version 5, Release 3, of AIX 5L (product number 5765-G03).

Contents

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:

@server®	Hypervisor™	POWER5™
ibm.com®	HACMP™	PTX®
pSeries®	IBM®	Redbooks™
AIX 5L™	Micro-Partitioning™	Redbooks (logo) ™
AIX®	Nways®	RS/6000®
DB2®	POWER™	Tivoli®
Enterprise Storage Server®	POWER3™	
ESCON®	POWER4™	

The following terms are trademarks of other companies:

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.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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 redbook is meant as a practical guide 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.

This book provides updated information about monitoring and tuning systems performance in an IBM @server® POWER5™ and AIX 5L V5.3 environment. Practical examples for the new and updated tools are provided, together with new information about using Resource Monitoring and to control part of RSCT for performance monitoring.

Also, in 10.1, "The performance status (Perfstat) API" on page 584, this book presents the Perfstat API for application programmers to have a better understanding of the new and updated facilities provided with this API.

The team that wrote this redbook

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

Kumiko Hayashi is an IT Specialist working at IBM Japan Systems Engineering Co., Ltd. She has four years of experience in AIX, RS/6000®, and IBM @server pSeries®. She provides pre-sales technical consultation and post-sales implementation support. She is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L.

Kangkook Ji is an IT Specialist at IBM Korea. He has four years of experience in AIX and pSeries. Currently as a Level 2 Support Engineer, he supports field engineers, and his main work is high availability solutions, such as HACMP™ and AIX problems. His interests vary in many IT areas, such as Linux® and middleware. He is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L and HACMP.

Octavian Lascu is a Project Leader at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM

classes worldwide in all areas of pSeries clusters and Linux. Before joining the ITSO, Octavian worked at IBM Global Services Romania as a Software and Hardware Services Manager. He holds a master's degree in Electronic Engineering from the Polytechnical Institute in Bucharest and is also an IBM Certified Advanced Technical Expert in AIX/PSSP/HACMP. He has worked with IBM since 1992.

Hennie Pienaar is a Senior Education Specialist in South Africa. He has eight years of experience in the AIX/Linux field. His areas of expertise include AIX, Linux and Tivoli®. He is certified as an Advanced Technical Expert. He has written extensively on AIX and Linux and has delivered classes worldwide on AIX and HACMP.

Susan Schreitmueller is a Sr. Consulting I/T Specialist with IBM. She joined IBM eight years ago, specializing in pSeries, AIX, and technical competitive positioning. Susan has been a Systems Administrator on zSeries, iSeries, and pSeries platforms and has expertise in systems administration and resource management. She travels extensively to customer locations, and has a talent for mentoring new hires and working to create a cohesive technical community that shares information at IBM.

Tina Tarquinio is a Software Engineer in Poughkeepsie, NY. She has worked at IBM for five years and has three years of AIX System Administration experience working in the pSeries Benchmark Center. She holds a bachelor's degree in Applied Mathematics and Computer Science from the University of Albany in New York. She is an IBM Certified pSeries AIX System Administrator and an Accredited IT Specialist.

James Thompson is a Performance Analyst for IBM Systems Group in Tucson, AZ. He has worked at IBM for five years, the first two years as a Level 2 Support Engineer for Tivoli Storage Manager and for the past three years he has provided performance support for the development of IBM Tape and NAS products. He holds a bachelor's degree in Computer Science from Utah State University.

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

Julie Peet, Certified IBM AIX System Administrator, pSeries Benchmark Center, Poughkeepsie, NY.

Nigel Griffiths, Certified IT Specialist, pSeries Advanced Technology Group, United Kingdom

Luc Smolders
IBM Austin

Andreas Hoetzel
IBM Austin

Gabrielle Velez
International Technical Support Organization, Rochester Center

Scott Vetter
IBM Austin

Dino Quintero
IBM Poughkeepsie

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'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll 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 email 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

Part 1

Introduction

Part 1 provides an overview of performance in an AIX 5L V5.3 environment and an introduction to performance analysis and tuning methodology. It also provides a description of overall performance metrics and expectations, together with the system components that should be considered for tuning in an IBM @server pSeries running AIX 5L.

1

Performance overview

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 make those expectations and their fulfillment match. The path to achieving this objective is a balance between appropriate expectations and optimizing the available system resources.

The performance-tuning process demands skill, knowledge, and experience, and cannot be performed by only analyzing statistics, graphs, and figures. If results are to be achieved, the human aspect of perceived performance must not be neglected. Performance tuning also takes into consideration problem determination aspects as well as pure performance issues.

1.1 Performance expectations

Performance tuning on a newly installed system usually involves setting the basic 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 may be necessary.

We therefore advise you 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.

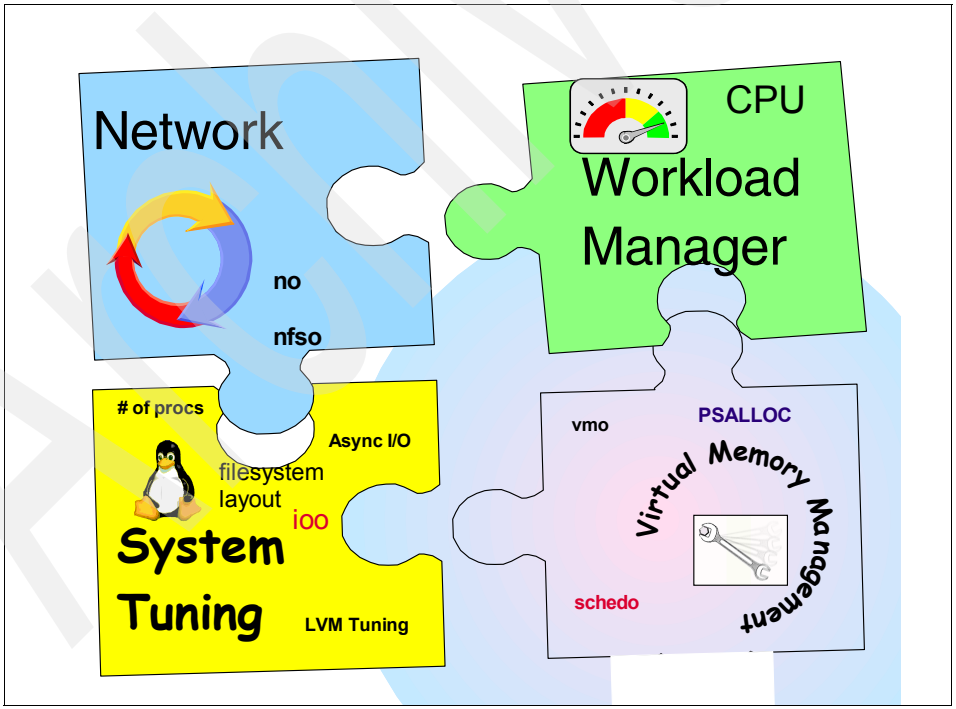

Figure 1-1 System tuning

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.

To evaluate if you have a performance issue, you can use the flow chart in Figure 1-2 as a guide.

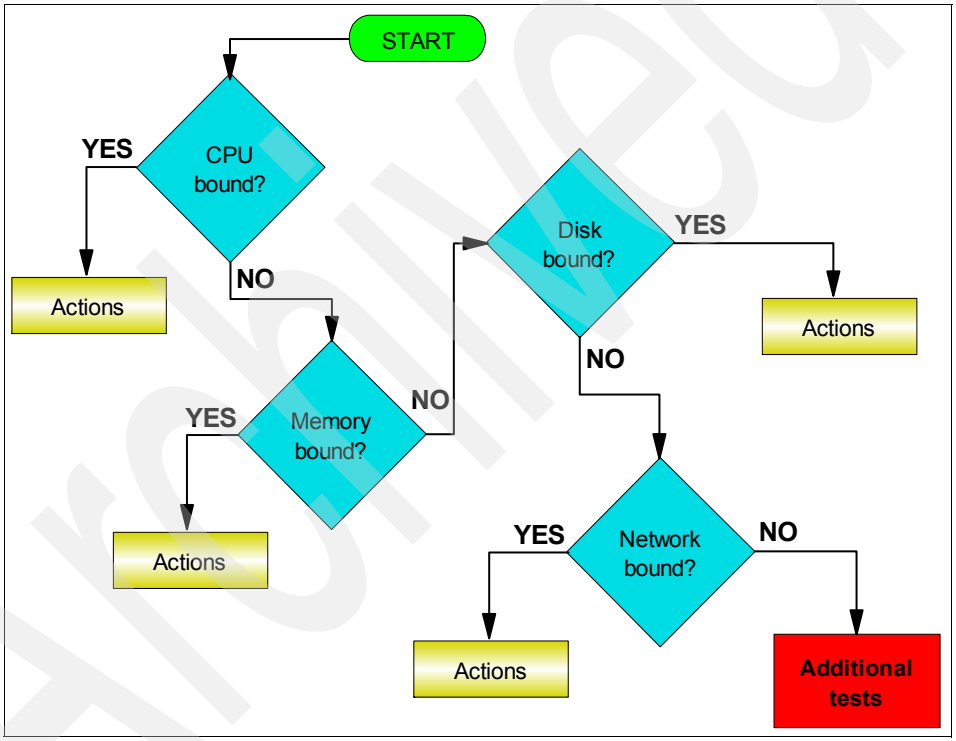

Figure 1-2 Performance problem determination flow chart

1.1.1 System workload

An accurate and complete definition of a system's workload is critical to understanding and/or predicting its performance. A difference in workload can cause far more variation in the measured performance of a system than differences in CPU clock speed or random access memory (RAM) size. The

workload definition must include not only the type and rate of requests sent to the system, but also the exact software packages and in-house application programs to be executed.

It is important to take into account the work that a system is performing in the background. For example, if a system contains file systems that are NFS-mounted and frequently accessed by other systems, handling those accesses is probably a significant fraction of the overall workload, even though the system is not a designated server.

A workload that has been standardized to allow comparisons among dissimilar systems is called a benchmark. However, few real workloads duplicate the exact algorithms and environment of a benchmark. Even industry-standard benchmarks that were originally derived from real applications have been simplified and homogenized to make them portable to a wide variety of hardware and software platforms.

The only valid use for industry-standard benchmarks is to narrow the field of candidate systems that will be subjected to a serious evaluation. Therefore, you should not solely rely on benchmark results when trying to understand the workload and performance of your system.

It is possible to classify workloads into the following categories:

Multiuser A workload that consists of a number of users submitting work through individual terminals. Typically, the performance objectives of such a workload are either to maximize system throughput while preserving a specified worst-case response time or to obtain the best possible response time for a constant workload.

Server A workload that consists of requests from other systems. For example, a file-server workload is mostly disk read and disk write requests. It is the disk-I/O component of a multiuser workload (plus NFS or other I/O activity), so the same objective of maximum throughput within a given response-time limit applies. Other server workloads consist of items such as math-intensive programs, database transactions, printer jobs.

Workstation A workload that consists of a single user submitting work through a keyboard and receiving results on the display of that system. Typically, the highest-priority performance objective of such a workload is minimum response time to the user's requests.

1.1.2 Performance objectives

After defining the workload that your system will have to process, you can choose performance criteria and set performance objectives based on those criteria. The

overall performance criteria of computer systems are response time and throughput.

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

- The amount of time a database query takes
- The amount of time it takes to echo characters to the terminal
- The amount of time it takes to access a Web page

Throughput is a measure of the amount of work that can be accomplished over some unit of time. Examples include:

- Database transactions per minute
- Kilobytes of a file transferred per second
- Kilobytes of a file read or written per second
- Web server hits per minute

The relationship between these metrics is complex. Sometimes you can have higher throughput at the cost of response time or better response time at the cost of throughput. In other situations, a single change can improve both. Acceptable performance is based on reasonable throughput combined with reasonable response time.

In planning for or tuning any system, make sure that you have clear objectives for both response time and throughput when processing the specified workload. Otherwise, you risk spending analysis time and resource dollars improving an aspect of system performance that is of secondary importance.

1.1.3 Program execution model

To clearly examine the performance characteristics of a workload, a dynamic rather than a static model of program execution is necessary, as shown in Figure 1-3 on page 8.

Program Execution Hierarchy

The figure is a triangle on its base. The left side represents hardware entities that are matched to the appropriate operating system entity on the right side. A program must go from the lowest level of being stored on disk, to the highest level being the processor running program instructions.

For instance, from bottom to top, the disk hardware entity holds executable programs; real memory holds waiting operating system threads and interrupt handlers; the translation lookaside buffer holds dispatchable threads; cache

contains the currently dispatched thread and the processor pipeline and registers contain the current instruction.

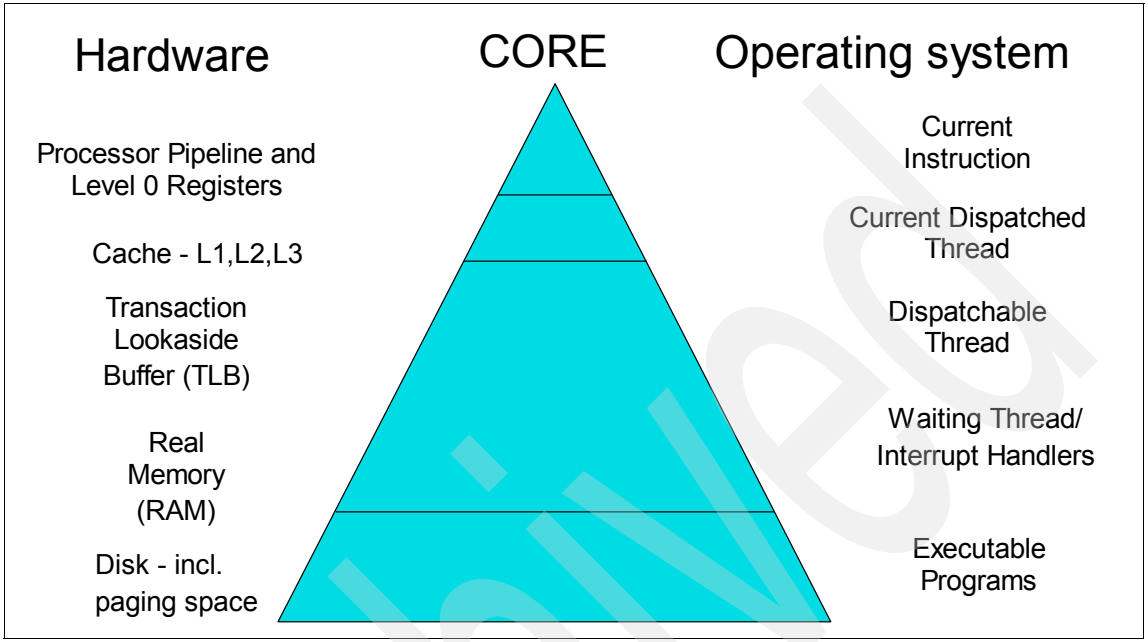

Figure 1-3 Program execution hierarchy

To run, a program must make its way up both the hardware and operating system hierarchies in parallel. Each element in the hardware hierarchy is more scarce and more expensive than the element below it. Not only does the program have to contend with other programs for each resource, the transition from one level to the next takes time. To understand the dynamics of program execution, you need a basic understanding of each of the levels in the hierarchy.

Hardware hierarchy

Usually, the time required to move from one hardware level to another consists primarily of the latency of the lower level (the time from the issuing of a request to the receipt of the first data).

Fixed disks

The slowest operation for a running program on a standalone system is obtaining code or data from a disk, for the following reasons:

- The disk controller must be directed to access the specified blocks (queuing delay).
- The disk arm must seek to the correct cylinder (seek latency).

- The read/write heads must wait until the correct block rotates under them (rotational latency).
- The data must be transmitted to the controller (transmission time) and then conveyed to the application program (interrupt-handling time).

Slow disk operations can have many causes besides explicit read or write requests in the program. System-tuning activities frequently prove to be hunts for unnecessary disk I/O.

Real memory

Real memory, often referred to as Random Access Memory, or RAM, is faster than disk, but much more expensive per byte. Operating systems try to keep in RAM only the code and data that are currently in use, storing any excess onto disk, or never bringing them into RAM in the first place.

RAM is not necessarily faster than the processor though. Typically, a RAM latency of dozens of processor cycles occurs between the time the hardware recognizes the need for a RAM access and the time the data or instruction is available to the processor.

If the access is going to a page of virtual memory that is stored over to disk, or has not been brought in yet, a page fault occurs, and the execution of the program is suspended until the page has been read from disk.

Translation Lookaside Buffer (TLB)

Programmers are insulated from the physical limitations of the system by the implementation of virtual memory. You design and code programs as though the memory were very large, and the system takes responsibility for translating the program's virtual addresses for instructions and data into the real addresses that are needed to get the instructions and data from RAM. Because this address-translation process can be time-consuming, the system keeps the real addresses of recently accessed virtual-memory pages in a cache called the translation lookaside buffer (TLB).

As long as the running program continues to access a small set of program and data pages, the full virtual-to-real page-address translation does not need to be redone for each RAM access. When the program tries to access a virtual-memory page that does not have a TLB entry, called a TLB miss, dozens of processor cycles, called the TLB-miss latency are required to perform the address translation.

Caches

To minimize the number of times the program has to experience the RAM latency, systems incorporate caches for instructions and data. If the required

instruction or data is already in the cache, a cache hit results and the instruction or data is available to the processor on the next cycle with no delay. Otherwise, a cache miss occurs with RAM latency.

In some systems, there are two or three levels of cache, usually called L1, L2, and L3. If a particular storage reference results in an L1 miss, then L2 is checked. If L2 generates a miss, then the reference goes to the next level, either L3, if it is present, or RAM.

Cache sizes and structures vary by model, but the principles of using them efficiently are identical.

Pipeline and registers

A pipelined, superscalar architecture makes possible, under certain circumstances, the simultaneous processing of multiple instructions. Large sets of general-purpose registers and floating-point registers make it possible to keep considerable amounts of the program's data in registers, rather than continually storing and reloading the data.

The optimizing compilers are designed to take maximum advantage of these capabilities. The compilers' optimization functions should always be used when generating production programs, however small the programs are. The Optimization and Tuning Guide for XL Fortran, XL C and XL C++ describes how programs can be tuned for maximum performance.

Software hierarchy

To run, a program must also progress through a series of steps in the software hierarchy.

Executable programs

When you request a program to run, the operating system performs a number of operations to transform the executable program on disk to a running program. First, the directories in the your current PATH environment variable must be scanned to find the correct copy of the program. Then, the system loader (not to be confused with the ld command, which is the binder) must resolve any external references from the program to shared libraries.

To represent your request, the operating system creates a process, or a set of resources, such as a private virtual address segment, which is required by any running program.

The operating system also automatically creates a single thread within that process. A thread is the current execution state of a single instance of a program. In AIX, access to the processor and other resources is allocated on a thread

basis, rather than a process basis. Multiple threads can be created within a process by the application program. Those threads share the resources owned by the process within which they are running.

Finally, the system branches to the entry point of the program. If the program page that contains the entry point is not already in memory (as it might be if the program had been recently compiled, executed, or copied), the resulting page-fault interrupt causes the page to be read from its backing storage.

Interrupt handlers

The mechanism for notifying the operating system that an external event has taken place is to interrupt the currently running thread and transfer control to an interrupt handler. Before the interrupt handler can run, enough of the hardware state must be saved to ensure that the system can restore the context of the thread after interrupt handling is complete. Newly invoked interrupt handlers experience all of the delays of moving up the hardware hierarchy (except page faults). Unless the interrupt handler was run very recently (or the intervening programs were very economical), it is unlikely that any of its code or data remains in the TLBs or the caches.

When the interrupted thread is dispatched again, its execution context (such as register contents) is logically restored, so that it functions correctly. However, the contents of the TLBs and caches must be reconstructed on the basis of the program's subsequent demands. Thus, both the interrupt handler and the interrupted thread can experience significant cache-miss and TLB-miss delays as a result of the interrupt.

Waiting threads

Whenever an executing program makes a request that cannot be satisfied immediately, such as a synchronous I/O operation (either explicit or as the result of a page fault), that thread is put in a waiting state until the request is complete. Normally, this results in another set of TLB and cache latencies, in addition to the time required for the request itself.

Dispatchable threads

When a thread is dispatchable but not running, it is accomplishing nothing useful. Worse, other threads that are running may cause the thread's cache lines to be reused and real memory pages to be reclaimed, resulting in even more delays when the thread is finally dispatched.

Currently dispatched threads

The scheduler chooses the thread that has the strongest claim to the use of the processor. When the thread is dispatched, the logical state of the processor is restored to the state that was in effect when the thread was interrupted.

Current machine instructions

Most of the machine instructions are capable of executing in a single processor cycle if no TLB or cache miss occurs. In contrast, if a program branches rapidly to different areas of the program and accesses data from a large number of different areas causing high TLB and cache-miss rates, the average number of processor cycles per instruction (CPI) executed might be much greater than one. The program is said to exhibit poor locality of reference. It might be using the minimum number of instructions necessary to do its job, but it is consuming an unnecessarily large number of cycles. In part because of this poor correlation between number of instructions and number of cycles, reviewing a program listing to calculate path length no longer yields a time value directly. While a shorter path is usually faster than a longer path, the speed ratio can be very different from the path-length ratio.

The compilers rearrange code in sophisticated ways to minimize the number of cycles required for the execution of the program. The programmer seeking maximum performance must be primarily concerned with ensuring that the compiler has all of the information necessary to optimize the code effectively, rather than trying to second-guess the compiler's optimization techniques The real measure of optimization effectiveness is the performance of an authentic workload.

1.1.4 System tuning

After efficiently implementing application programs, further improvements in the overall performance of your system becomes a matter of system tuning. The main components that are subject to system-level tuning are:

Communications I/O

Depending on the type of workload and the type of communications link, it might be necessary to tune one or more of the following communications device drivers: TCP/IP, or NFS.

Fixed disk

The Logical Volume Manager (LVM) controls the placement of file systems and paging spaces on the disk, which can significantly affect the amount of seek latency the system experiences. The disk device drivers control the order in which I/O requests are acted upon.

Real memory

The Virtual Memory Manager (VMM) controls the pool of free real-memory frames and determines when and from where to steal frames to replenish the pool.

Running thread

The scheduler determines which dispatchable entity should next receive control. In AIX, the dispatchable entity is a thread.

1.2 Introduction to the performance tuning process

Performance tuning is primarily a matter of resource management and correct system parameters setting. Tuning the workload and the system for efficient resource use consists of the following steps:

- Identifying the workloads on the system
- Setting objectives:
  - Determining how the results will be measured
  - Quantifying and prioritizing the objectives
- Identifying the critical resources that limit the system's performance
- Minimizing the workload's critical-resource requirements:
  - Using the most appropriate resource, if there is a choice
  - Reducing the critical-resource requirements of individual programs or system functions
  - Structuring for parallel resource use
- Modifying the allocation of resources to reflect priorities
  - Changing the priority or resource limits of individual programs
  - Changing the settings of system resource-management parameters
- Repeating above steps until objectives are met (or resources are saturated)
- Applying additional resources, if necessary

There are appropriate tools for each phase of system performance management Some of the tools are available from IBM; others are the products of third parties. The following figure illustrates the phases of performance management in a simple LAN environment.

1.2.1 Performance management phases

Figure 1-4 uses five weighted circles to illustrate the steps of performance tuning a system; plan, install, monitor, tune, and expand. Each circle represents the system in various states of performance; idle, unbalanced, balanced, and overloaded. Essentially, you need to expand a system that is overloaded, tune a system until it is balanced, monitor an unbalanced system and install for more resources when an expansion is necessary.

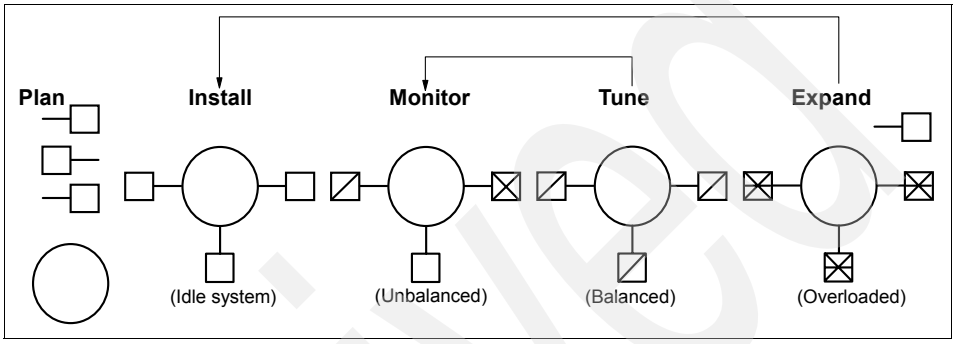

Figure 1-4 Performance phases

Identification of the workloads

It is essential that all of the work performed by the system be identified. Especially in LAN-connected systems, a complex set of cross-mounted file systems can easily develop with only informal agreement among the users of the systems. These file systems must be identified and taken into account as part of any tuning activity.

With multiuser workloads, the analyst must quantify both the typical and peak request rates. It is also important to be realistic about the proportion of the time that a user is actually interacting with the terminal.

An important element of this identification stage is determining whether the measurement and tuning activity has to be done on the production system or can be accomplished on another system (or off-shift) with a simulated version of the actual workload. The analyst must weigh the greater authenticity of results from a production environment against the flexibility of the nonproductive environment, where the analyst can perform experiments that risk performance degradation or worse.

Importance of setting objectives

Although you can set objectives in terms of measurable quantities, the actual desired result is often subjective, such as satisfactory response time. Further, the

analyst must resist the temptation to tune what is measurable rather than what is important. If no system-provided measurement corresponds to the desired improvement, that measurement must be devised.

The most valuable aspect of quantifying the objectives is not selecting numbers to be achieved, but making a public decision about the relative importance of (usually) multiple objectives. Unless these priorities are set in advance, and understood by everyone concerned, the analyst cannot make trade-off decisions without incessant consultation. The analyst is also apt to be surprised by the reaction of users or management to aspects of performance that have been ignored. If the support and use of the system crosses organizational boundaries, you might need a written service-level agreement between the providers and the users to ensure that there is a clear common understanding of the performance objectives and priorities.

Identification of critical resources

In general, the performance of a given workload is determined by the availability and speed of one or two critical system resources. The analyst must identify those resources correctly or risk falling into an endless trial-and-error operation.

Systems have both real and logical resources. Critical real resources are generally easier to identify, because more system performance tools are available to assess the utilization of real resources. The real resources that most often affect performance are as follows:

- CPU cycles
- Memory
- I/O bus
- Various adapters
- Disk arms/heads/spindles
- Disk space
- Network access

Logical resources are less readily identified. Logical resources are generally programming abstractions that partition real resources. The partitioning is done to share and manage the real resource.

Some examples of real resources and the logical resources built on them are as follows:

CPU

- Processor time slice

Memory

- Page frames

- Stacks
- Buffers
- Queues
- Tables
- Locks and semaphores

Disk space

- Logical volumes
- File systems
- Files
- Partitions

Network access

- Sessions
- Packets
- Channels

It is important to be aware of logical resources as well as real resources. Threads can be blocked by a lack of logical resources just as for a lack of real resources, and expanding the underlying real resource does not necessarily ensure that additional logical resources will be created. For example, consider the NFS block I/O daemon, biod. A biod daemon on the client is required to handle each pending NFS remote I/O request. The number of biod daemons therefore limits the number of NFS I/O operations that can be in progress simultaneously. When a shortage of biod daemons exists, system instrumentation may indicate that the CPU and communications links are used only slightly. You may have the false impression that your system is underused (and slow), when in fact you have a shortage of biod daemons that is constraining the rest of the resources. A biod daemon uses processor cycles and memory, but you cannot fix this problem simply by adding real memory or converting to a faster CPU. The solution is to create more of the logical resource (biod daemons).

Logical resources and bottlenecks can be created inadvertently during application development. A method of passing data or controlling a device may, in effect, create a logical resource. When such resources are created by accident, there are generally no tools to monitor their use and no interface to control their allocation. Their existence may not be appreciated until a specific performance problem highlights their importance.

Minimizing critical resource requirements

Consider minimizing the workload's critical-resource requirements at three levels, as discussed below.

Using the appropriate resource

The decision to use one resource over another should be done consciously and with specific goals in mind. An example of a resource choice during application development would be a trade-off of increased memory consumption for reduced CPU consumption. A common system configuration decision that demonstrates resource choice is whether to place files locally on an individual workstation or remotely on a server.

Reducing the requirement for the critical resource

For locally developed applications, the programs can be reviewed for ways to perform the same function more efficiently or to remove unnecessary function. At a system-management level, low-priority workloads that are contending for the critical resource can be moved to other systems, run at other times, or controlled with the Workload Manager.

Structuring for parallel use of resources

Because workloads require multiple system resources to run, take advantage of the fact that the resources are separate and can be consumed in parallel. For example, the operating system read-ahead algorithm detects the fact that a program is accessing a file sequentially and schedules additional sequential reads to be done in parallel with the application's processing of the previous data. Parallelism applies to system management as well. For example, if an application accesses two or more files at the same time, adding an additional disk drive might improve the disk-I/O rate if the files that are accessed at the same time are placed on different drives.

Resource allocation priorities

The operating system provides a number of ways to prioritize activities. Some, such as disk pacing, are set at the system level. Others, such as process priority, can be set by individual users to reflect the importance they attach to a specific task.

Repeating the tuning steps

A truism of performance analysis is that there is always a next bottleneck. Reducing the use of one resource means that another resource limits throughput or response time. Suppose, for example, we have a system in which the utilization levels are as follows:

CPU: 90% Disk: 70% Memory 60%

This workload is CPU-bound. If we successfully tune the workload so that the CPU load is reduced from 90 to 45 percent, we might expect a two-fold improvement in performance. Unfortunately, the workload is now I/O-limited, with utilizations of approximately the following:

CPU: 45% Disk: 90% Memory 60%

The improved CPU utilization allows the programs to submit disk requests sooner, but then we hit the limit imposed by the disk drive's capacity. The performance improvement is perhaps 30 percent instead of the 100 percent we had envisioned.

There is always a new critical resource. The important question is whether we have met the performance objectives with the resources at hand.

Attention: Improper system tuning with vmtune, schedtune, and other tuning commands can result in unexpected system behavior like degraded system or application performance, or a system hang. Changes should only be applied when a bottleneck has been identified by performance analysis.

Applying additional resources

If, after all of the preceding approaches have been exhausted, the performance of the system still does not meet its objectives, the critical resource must be enhanced or expanded. If the critical resource is logical and the underlying real resource is adequate, the logical resource can be expanded at no additional cost. If the critical resource is real, the analyst must investigate some additional questions:

- How much must the critical resource be enhanced or expanded so that it ceases to be a bottleneck?
- Will the performance of the system then meet its objectives, or will another resource become saturated first?
- If there will be a succession of critical resources, is it more cost-effective to enhance or expand all of them, or to divide the current workload with another system?

A more detailed diagram of performance management and tuning is presented in Figure 1-5 on page 19.

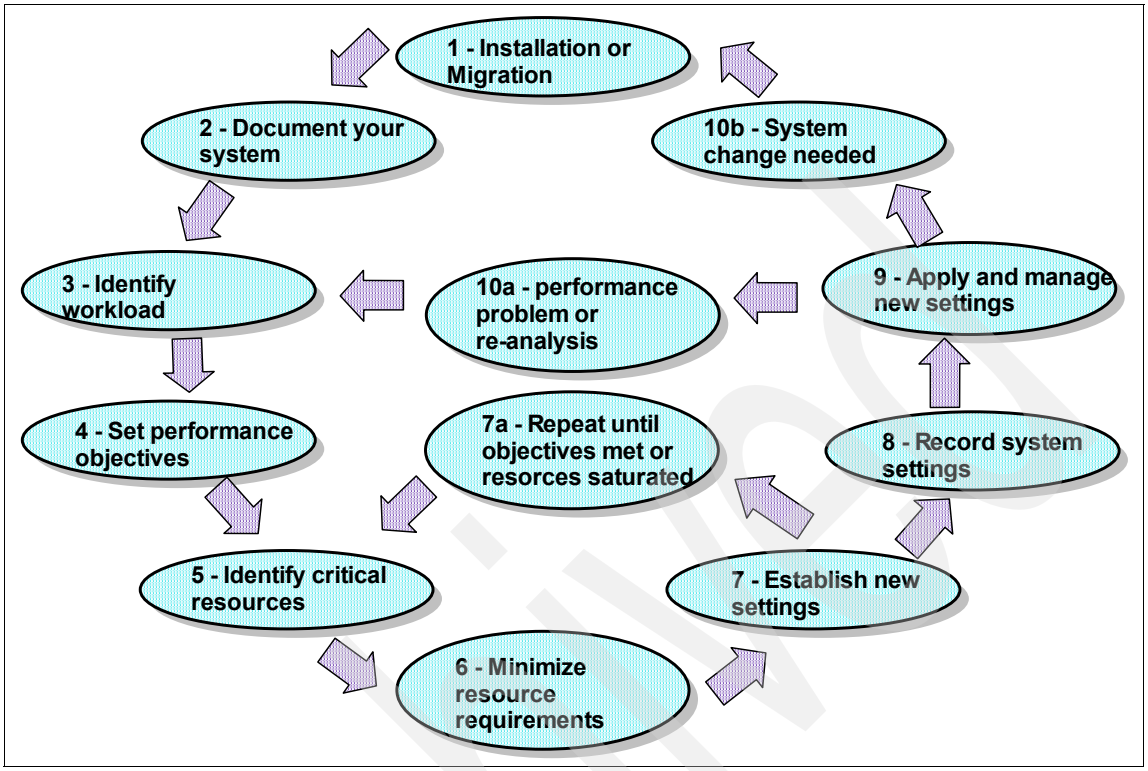

Figure 1-5 Performance management cycle

2

Performance analysis 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 can be actually tuned from the systems is categorized into CPU, memory, disk, and network, as discussed in:
  - “CPU performance” on page 22
  - “Memory overview” on page 31
  - “Disk I/O performance” on page 37
  - “Network performance” on page 48

2.1 CPU performance

To monitoring and tuning of CPU performance, it is important to know about process and scheduling. This section gives an overview of the process, thread, and scheduling which are closely related to the performance of CPU.

2.1.1 Processes and threads

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. 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 libpthreads. Multiple threads of control allow an application to service requests from multiple users at the same time. With the libpthreads implementation, user threads sit on top of virtual processors (VP) which are themselves on top of kernel threads. A multi threaded user process can use one of two models, as follows:

1:1 Thread Model: The 1:1 model indicates that each user thread will have exactly one kernel thread mapped to it. This is the default model in early AIX 4.3. In this model, each user thread is bound to a VP and linked to exactly one kernel thread. The VP is not necessarily bound to a real CPU (unless binding to a processor was done). A thread which is bound to a VP is said to have system scope because it is directly scheduled with all the other user threads by the kernel scheduler.

M:N Thread Model: The M:N model was implemented in AIX 4.3.1 and has been since then the default model. In this model, several user threads can share the same virtual processor or the same pool of VPs. Each VP can be thought of as a virtual CPU available for executing user code and system calls. A thread which is not bound to a VP is said to be a local or process scope because it is not directly scheduled with all the other threads by the kernel scheduler. The

pthreads library will handle the scheduling of user threads to the VP and then the kernel will schedule the associated kernel thread. As of AIX 4.3.2, the default is to have one kernel thread mapped to eight user threads. This is tunable from within the application or through an environment variable.

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 (see 4.3.5, "The nice command" on page 288, and 4.3.6, "The renice command" on page 289).

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 can be shown with the **ps -ef** command, looking at the "C" column of the output (see Example 4-19 on page 212).

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

```
CPU usage = CPU usage*(D/32)
```

Where D is the decay value as set by **schedo -o sched_D** (see 4.3.4, "The schedo command" on page 282).

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 *base priority* of a thread is 40.

nice value The *nice value* defaults to 20 for foreground processes and 24 for background processes. This can be changed using the nice or renice command.

r The *CPU penalty factor*. The default for r is 16. This value can be changed with the schedo command.

D The *CPU decay factor*. The default for D is 16. This value can be changed with the schedo command.

C CPU usage as Thread aging in preceding subsection.

p_nice This is called the *niced priority.* It is calculated as from:

```
p_nice = base priority + nice value
```

x_nice The "*extra nice*" value. If the niced priority for a thread (p_nice) is larger than 60, then the following formula applies:

```
x_nice = p_nice * 2 - 60
```

If the niced priority for a thread (p_nice) is equal or less than 60, the following formula applies:

```
x_nice = p_nice
```

X The xnice factor is calculated as:

```
(x_nice + 4) / 64.
```

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

```
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.

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**.

2.1.2 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 redispatched, 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 maxpspin 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.

  – 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. Others 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.

For more information about multiprocessing, refer to the *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905.

http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixbman/prftungd/prftungd.pdf

2.1.3 Initial advice for monitoring CPU

When you monitor the CPU usage, the **vmstat** command is a good tool use for this purpose. The **vmstat** command displays entire system performance statistics. Example 2-1 shows a sample of entire system performance statistics. The new **lparstat** command is also useful to measure the CPU usage of the whole system.

Example 2-1 Entire system performance statistics

```
[p630n06][/]> vmstat 1

System configuration: lcpu=4 mem=8192MB

kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  0 341382 240915   0   0   0   0    0   0  25  805 368 25  1 74  0
 1  0 341382 240915   0   0   0   0    0   0   5  195 297 25  0 74  0
 1  0 341382 240915   0   0   0   0    0   0   3  192 310 25  0 75  0
 1  0 341382 240915   0   0   0   0    0   0   2  602 305 25  0 75  0
 1  0 341382 240915   0   0   0   0    0   0   1  190 304 25  0 75  0
... lines omitted ...
```

You need to check the *kthr* and the *cpu* category. The *kthr* category reports the average number of thread on run queue (r column) and wait queue (b column).

The *cpu* category reports CPU statistics information.

us The us column shows the percent of CPU time spent in user mode.

sy The sy column details the percentage of time the CPU was executing a process in system mode.

id The ID column shows the percentage of time which the CPU is idle.

wa The wa column details the percentage of time the CPU was idle with pending local disk I/O and NFS-mounted disks.

In this example, the us (user) column of the *cpu* category shows 25% utilization. we should know that when the server has two or more processors, the **vmstat** displays the average of the total CPU utilization.

To determine the number of available CPUs, use the **lsdev** command as in Example 2-2. In this example, four CPUs are displayed as available.

Example 2-2 Determine the number of available CPUs

```
[p630n06][/]> lsdev -Cc processor
proc0 Available 00-00 Processor
proc1 Available 00-01 Processor
proc2 Available 00-02 Processor
proc3 Available 00-03 Processor
```

The **mpstat** command is a good tool to monitor each CPU utilization. Example 2-3 shows a sample of displaying each CPU utilization with the **mpstat** command. In this example, we can see that cpu0 keeps 100% busy and cpu2, cpu3, cpu4 are idle. Therefore, the average CPU usage of four processors becomes 25% shown in the "ALL" line. The **sar** command is also useful command to measure the each CPU usage.

Example 2-3 Displaying each CPU utilization

```
[p630n06][/]> mpstat 1

System configuration: lcpu=4

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id
  0    0    0    0  104    1    1    1    0 100    0 100  0  0  0
  1    0    0    0  106   46   29    0    0 100  461  0  0  0 100
  2    0    0    0  215  244  125    1    0 100  195  0  0  0 100
  3    0    0    0  103  131  124    0    0 100   51  0  0  0 100
ALL    0    0    0  528  422  279    2    0 100  707 25  0  0 75
------------------------------------------------------------------------------
  0    0    0    0  104    1    1    1    0 100    0 100  0  0  0
  1    0    0    0  104   37   25    0    0 100   28  0  0  0 100
  2    0    0    0  212  236  125    1    0 100  138  0  1  0 99
  3    0    0    0  111  145  128    0    1 100   82  0  0  0 100
ALL    0    0    0  531  419  279    2    1 100  248 25  0  0 75
------------------------------------------------------------------------------
  0    0    0    0  106    1    1    1    0 100    0 100  0  0  0
  1    0    0    0  105   43   29    0    0 100   31  0  0  0 100
  2    0    0    0  205  231  124    1    0 100  125  0  0  0 100
  3    0    0    0  115  147  126    0    0 100   97  0  0  0 100
ALL    0    0    0  531  422  280    2    0 100  253 25  0  0 75
------------------------------------------------------------------------------
```

The **topas** command reports statistics information about the activity on the local system on a character terminal. Using the **-P** flag, topas provides the lists of

busiest processes. Example 2-4 shows an example of the output of the busiest processes screen. By default, CPU% column is the sort key. In this example, we can see cpu_load (a program we used for our tests) process is the busiest process.

Then, it is necessary to investigate the process itself. AIX provides trace and profile tool such as **trace**, **tprof**, and some other commands.

Example 2-4 Displaying the lists of busiest processes

```
[p630n06][/]> topas -P
Topas Monitor for host:    p630n06     Interval:   2    Tue Oct 26 20:21:01 2004

                                 DATA  TEXT  PAGE              PGFAULTS
USER       PID    PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
Topas Monitor for host:    p630n06     Interval:   2    Tue Oct 26 20:21:15 2004

                                 DATA  TEXT  PAGE              PGFAULTS
USER       PID    PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root    630800  172076 135 24     3     1    14    8:59 25.0    0    0 cpu_loade
root    577628  540876  58 41   307    67   307    0:00  0.0    0    0 topas
root    475292  282774  60 20   163    84   243    0:00  0.0    0    0 sshd
root    471268  180458  60 20   299   223   667   70:44  0.0    0    0 IBM.CSMAg
root    123020       1  60 20    70     2   128   19:36  0.0    0    0 syncd
root    401584  180458  60 20   588     2  4198   15:36  0.0    0    0 java
root    254080  180458  60 20    43   383   316    8:34  0.0    0    0 i4llmd
root    274524  335892  60 20   113   138   347    6:21  0.0    0    0 dtfile
root     57372       0  37 41    24     0    28    5:54  0.0    0    0 gil
root    192608  180458  60 20    27   133   162    1:48  0.0    0    0 aixmibd
root    295060       1  60 20    41     0    51    1:02  0.0    0    0 rpc.lockd
root    364616  180458  60 20   215    46   215    0:49  0.0    0    0 pcmsrv
root     28686       0  16 41    10     0    13    0:42  0.0    0    0 lrud
root    217250  180458  60 20   546   131   847    0:36  0.0    0    0 rmcd
root    131154       0  60 20    39     0    44    0:32  0.0    0    0 kbiod
root    512068  180458  60 20   215   193   430    0:25  0.0    0    0 IBM.HostR
root    729164  274524  60 20    33   138   267    0:18  0.0    0    0 dtfile
root    282774  180458  60 20    30    84   110    0:16  0.0    0    0 sshd
root     49176       0  60 41    11     0    12    0:11  0.0    0    0 xmgc
root    393308  180458  60 20    39   164   214    0:10  0.0    0    0 ctcasd
```

2.2 Memory overview

To analyze memory, you first need to know how much memory you have. Example 2-5 on page 32 demonstrates how to find out how much memory your system has.

Example 2-5 Using lsattr

```
[node4][/]> lsattr -El mem0
goodsize 8192 Amount of usable physical memory in Mbytes False
size     8192 Total amount of physical memory in Mbytes  False
```

The **lsattr** command will report the amount of memory in MB, so in the example above the machine has 8GB of memory.

2.2.1 Virtual memory manager (VMM) overview

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 memory and paging space on the system.

The VMM services memory requests from the system and its applications. Virtual-memory segments are partitioned in units called pages; each page is either located in real physical memory (RAM) or stored on disk until it is needed. AIX uses virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM.

The amount of virtual memory used can exceed the size of 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.

In AIX, virtual-memory segments are partitioned into 4096-byte units called pages. The VMM maintains a free list of available page frames. The VMM also uses a page-replacement algorithm to determine which virtual-memory pages currently in RAM will have their page frames reassigned to the free list. The page-replacement algorithm takes into account the existence of persistent versus working segments, repaging, and VMM thresholds.

Free List

The VMM maintains a list of free (unallocated) page frames that it uses to satisfy page faults. AIX tries to use all of RAM all of the time, except for a small amount which it maintains on the free list. To maintain this small amount of unallocated pages the VMM uses page outs and page steals to free up space and reassign those page frames to the free list. The virtual-memory pages whose page frames are to be reassigned are selected using the VMM's page-replacement algorithm.

See “Paging space allocation policies” on page 34 for more information about paging space allocation policies.

Memory Segments

AIX distinguishes between different types of memory segments. To understand the VMM, it is important to understand the difference between persistent, working and client segments.

Persistent segments have a permanent storage location on disk. Files containing data or executable programs are mapped to persistent segments. When a JFS file is opened and accessed, the file data is copied into RAM. VMM parameters control when physical memory frames allocated to persistent pages should be overwritten and used to store other data.

For JFS2, the file pages will be cached as local client pages. File data will be copied into RAM, unless the file is accessed through Direct I/O (DIO) or Concurrent I/O (CIO).

Working segments are transitory and exist only during their use by a process. Working segments have no permanent disk storage location. Process stack and data regions are mapped to working segments and shared library text segments. Pages of working segments must also occupy disk storage locations when they cannot be kept in real memory. The disk paging space is used for this purpose. When a program exits, all of its working pages are placed back on the free list immediately.

Client 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.

Memory segments can be shared between processors or maintained as private.

Working Segments and Paging Space

Working pages in RAM that can be modified and paged out are assigned a corresponding slot in paging space. The allocated paging space is used only if the page needs to be paged out. However, an allocated page in paging space cannot be used by another page. It remains reserved for a particular page for as long as that page exists in virtual memory. Because persistent pages are paged out to the same location on disk from which they came, paging space does not need to be allocated for persistent pages residing in RAM.

The VMM has three modes for allocating paging space: early, late, and deffered. *Early* allocation policy reserves paging space whenever a memory request for a working page is made. *Late* allocation policy assigns paging space when the

working page is being touched. *Deferred* allocation policy assigns paging space when the working page is actually paged out of memory, which significantly reduces the paging space requirements of the system.

VMM Memory Load Control Facility

When a process references a virtual-memory page that is on disk, because it either has been paged out or has never been read, the referenced page must be paged in, and this might cause one or more pages to be paged out if the number of available (free) page frames is low. The VMM attempts to steal page frames that have not been recently referenced and, therefore, are not likely to be referenced in the near future, using a page-replacement algorithm. A successful page-replacement keeps the memory pages of all currently active processes in RAM, while the memory pages of inactive processes are paged out. However, when RAM is over-committed, it becomes difficult to choose pages for page out because, they will probably be referenced in the near future by currently running processes. The result is that pages that are likely to be referenced soon might still get paged out and then paged in again when actually referenced. When RAM is over-committed, continuous paging in and paging out, called thrashing, can occur. When a system is thrashing, the system spends most of its time paging in and paging out instead of executing useful instructions, and none of the active processes make any significant progress. The VMM has a memory load control algorithm that detects when the system is thrashing and then attempts to correct the condition.

2.2.2 Paging space overview

A paging space is a type of logical volume with allocated disk space that stores information which is resident in virtual memory but is not currently being accessed. This logical volume has an attribute type equal to paging, and is usually simply referred to as paging space or swap space. When the amount of free RAM in the system is low, programs or data that have not been used recently are moved from memory to paging space to release memory for other activities. The amount of paging space required depends on the type of activities performed on the system. If paging space runs low, processes can be lost, and if paging space runs out, the system can panic. When a paging space low condition is detected, define additional paging space. The logical volume paging space is defined by making a new paging space logical volume or by increasing the size of existing paging space logical volumes.The total space available to the system for paging is the sum of the sizes of all active paging space logical volumes.

Paging space allocation policies

AIX uses three modes for paging space allocation. The PSALLOC environment variable determines which paging space allocation algorithm is used: late or

early. You can switch to an early paging space allocation mode by changing the value of the PSALLOC environment variable, but there are several factors to consider before making such a change. When using the early allocation algorithm, in a worst-case scenario, it is possible to crash the system by using up all available paging space.

Comparing paging space allocation policies

The operating system supports three paging space allocation policies:

- **Late Allocation Algorithm** (LPSA) This paging space slot allocation method is intended for use in installations where performance is more important than the possibility of a program failing due to lack of memory. In this algorithm, the paging space disk blocks are not allocated until corresponding pages in RAM are touched.
- **Early Allocation Algorithm** (EPSA) This paging space slot allocation method is intended for use in installations where this situation is likely, or where the cost of failure to complete is intolerably high. Aptly called early allocation, this algorithm causes the appropriate number of paging space slots to be allocated at the time the virtual-memory address range is allocated, for example, with the malloc() subroutine. If there are not enough paging space slots to support the malloc() subroutine, an error code is set. 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.
- **Deferred Allocation Algorithm** This paging space slot allocation method is the default beginning with AIX 4.3.2 Deferred Page Space Allocation (DPSA) policy delays allocation of paging space until it is necessary to page out the page, which results in no wasted paging space allocation. This method can save huge amounts of paging space, which means disk space. On some systems, paging space might not ever be needed even if all the pages accessed have been touched. This situation is most common on systems with very large amount of RAM. However, this may result in overcommitment of paging space in cases where more virtual memory than available RAM is accessed. This method saves huge amounts of paging space. To disable this policy, use the **vmo** command and set the defps parameter to 0 (with **vmo -o defps=0**). If the value is set to zero then the late paging space allocation policy is used.

In AIX 5L V5.3 there are two paging space garbage collection (PSGC) methods

- Garbage collection on re-pagein
- Garbage collection scrubbing for in memory frames

Paging Space Default Size

The default paging space size is determined during the system customization phase of AIX installation according to the following standards:

- Paging space can use no less than 16 MB, except for hd6 which can use no less than 64 MB in AIX 4.3 and later.
- Paging space can use no more than 20% of total disk space.
- If real memory is less than 256 MB, paging space is two times real memory.
- If real memory is greater than or equal to 256 MB, paging space is 512 MB.

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.

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 **lsps** command. The **-s** flag should be issued rather than the -a flag of the lsps command because the former includes pages in paging space reserved by the EPSA policy.

2.3 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

2.3.1 Initial advice

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 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 software[1]. Note that many logical limitations on the disk I/O subsystem can be monitored and tuned with the `ioo` command.

For further information, refer to:

- *AIX 5L Version 5.3 Performance Management Guide,* SC23-4905
- *AIX 5L Version 5.3 System Management Concepts: Operating System and Devices,* SC23-4908
- *AIX 5L Version 5.3 System Management Guide: Operating System and Devices,* SC23-4910

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

2.3.2 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.

2.3.3 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. 2 Gb fibre channel adapters have a channel rate of 200 megabytes per second. The maximum likely to be realized by the system is 175 megabytes per second.

- 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 current generation of PCI-X slots have burst bandwidths from 533 - 1066 megabytes per second. To calculate, take the bit value (32 or 64) multiply by the MHz value and divide by 8. Dividing by 8 converts the number to bytes. So a PCI-X 64 bit slot running at 100 MHz has a burst bandwidth of (64 * 100 / 8 = 800 MB/s).

2.3.4 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.

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, up to a point, 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; however, pbufs are pinned so that allocating many pbufs increases the use of memory.

2.3.5 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 physical volumes, which can vary in capacity and performance. A physical volume can belong to only one volume group at a time.

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.

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. These policies affect locally attached physical disk. Disk LUNs from storage subsystems are not affected by these 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 2-1:

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

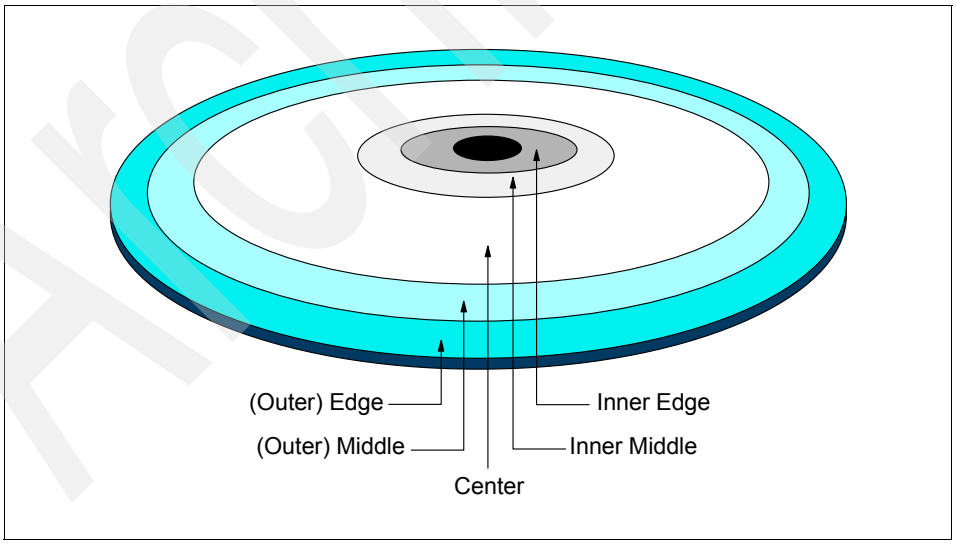

Figure 2-1 Physical partition mapping

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 storage consists of RAID LUNs, 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 LVDD[2] generates a write request for every mirror copy. If a logical volume is using mirror write consistency, the 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

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

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.

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 (not available for JFS2) 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 in-line 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 a in-line 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.

2.4 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 2-2 on page 53. Note that with network tuning, indiscriminately using buffers that are too large can reduce performance.

For more information about how the different protocols work, refer to:

- 6.7.1, “The no command” on page 396
- 6.7.3, “The nfso command” on page 416
- *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905
- *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- *AIX 5L Version 5.3 System Management Guide: Operating System and Devices,* SC23-4910
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *RS/6000 SP System Performance Tuning Update,* SG24-5340, at:

  `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 2.4.2, “TCP/IP protocol” on page 50. Information about the network tunables, including network adapter tunables, is provided in 2.4.3, “Network tunables” on page 51.

2.4.1 Initial advice

A good knowledge of your network topology 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 performance” on page 333. 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 5.3 Commands Reference, Volume 5*, SC23-4892, 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. In 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 3.3, “The perfpmr utility” on page 77.

- 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 3.3, “The perfpmr utility” on page 77 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 **traceroute** 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 6.4.1, "The entstat command" on page 351 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 **perfpmr** tool. However, the **perfpmr** tool may change, so it is advised to control the data gathered by **perfpmr** to ensure that the outputs of the **netstat** and **nfsstat** commands are included.

2.4.2 TCP/IP protocol

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 5.3 System Management Guide: Communications and Networks*, SC23-4909, 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. Table 2-1 contains a short, top-down reminder of TCP/IP protocols hierarchy.

Table 2-1 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

2.4.3 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 system-wide with the **no** command (refer to 6.7.1, “The no command” on page 396), or use the Interface Specific Network Options[3] (ISNO) for each network adapter. For more details about ISNO, see *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909, and *AIX 5L Version 5.3 Commands Reference*, SC23-4888.

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

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
```

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 2-6 shows how to use the **lsattr** command to check the current settings for an network interface, in this case token-ring:

Example 2-6 Using lsattr to check adapter settings

```
# lsattr -H -El 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_mssdflt
```

The highlighted part in Example 2-6 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 (from /etc/inittab).

Network buffer tuning

The values in Table 2-2 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 2-2 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 2-3 shows some other network tunables that should be considered and other ways to calculate some of the values in shown in Table 2-2 on page 53.

Table 2-3 Other basic network tunables

tunable name	Comment
thewall	The thewall parameter is read only and cannot be changed. It is set at system boot to half the size of the memory, with a limit of 1GB on 32-bit kernel, and 65GB on a 64-bit kernel. **`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[a]. 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	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. 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_recvspce, udp_sendspace, and udp_recvspace parameters cannot be set larger than sb_max. 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. Once the total buffers on the socket reach the sb_max limit, no more buffers will be allowed to be queued to that socket.

tunable name	Comment
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.
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[b] 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)[c].
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. In a heterogeneous environment the value determined by MTU discovery can be way off.
b. 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.
c. 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 2-7.

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 2-7 is for an Ethernet network adapter interface using the **lsattr** command.

Example 2-7 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      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
slih_hog      10               Interrupt events processed per interrupt  True
```

Example 2-8 shows what it might look like on a token-ring network adapter interface using the **lsattr** command.

Example 2-8 Using lsattr on a token-ring network adapter interface

```
# lsattr -El 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 2-9.

Example 2-9 Using lsattr to find out attribute ranges for a network adapter interface

```
# lsattr -Rl ent0 -a tx_que_size
512...16384 (+1)
# lsattr -Rl ent0 -a rx_que_size
512
# lsattr -Rl tok0 -a xmt_que_size
32...16384 (+1)
# lsattr -Rl 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 2-10. Note that with the -P attribute, the changes will be effective after the next IPL.

Example 2-10 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.

The MTU sizes for a network adapter interface can be examined by using the **lsattr** command and the mtu attribute as in Example 2-11, which shows the tr0 network adapter interface.

Example 2-11 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 2-12 shows the allowable MTU sizes for Ethernet (en0).

Example 2-12 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 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 boot time value for the network option **extendednetstats** is 1 (one — the collection of extended network statistics is enabled). However, because these extra statistics may cause a reduction in system performance, **extendednetstats** is set to 0, for off, in /etc/rc.net. If you want to enable this option at system runtime, you should comment the corresponding line in /etc/rc.net. Keep in mind that you need to reboot the system for changing this variable.

Some high-speed adapters have ISNO parameters set by default in the ODM database. Review the *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909, for individual adapters default values, or use the **lsattr** command with the -D option as in Example 2-13.

Example 2-13 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
slih_hog      10               Interrupt events processed per interrupt True
```

The deflt column shows the default values for each attribute. Example 2-14 shows how to use them on an Ethernet network adapter interface.

Example 2-14 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_mssdflt     N/A                                 True
```

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

Part 2

Performance tools

In Part 2 we describe the performance monitoring and tuning tools for the four major subsystem components: CPU, memory, network I/O and disk I/O.

We also discuss some of the high level tools used as an entry point in performance analyzing and tuning methodology, as well as some in-depth tools for performance problem determination.

3

General performance monitoring tools

In this chapter we discuss the steps required to run general performance tools and how to interpret the output of the tools.

This chapter discusses the following tools:

- topas
- jtopas
- perfpmr (tool for collecting performance data when reporting a problem)
- Performance Diagnostics Tool (PDT)
- trace

3.1 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 usage
- CPU events and queues
- memory and paging use
- disk performance
- network performance
- WLM partitioning
- NFS statistics

Topas 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.

The **topas** command is located at /usr/bin/topas and is part of the bos.perf.tools fileset and provided since AIX Version 4.3.

The performance monitoring module in topas is implemented using the facility of System Performance Measurement Interface (SPMI). Therefore, like other tools using SPMI, you can see shared memory segment with address starting 0x78 in in shared memory address space while running **topas** command (see Example 3-1).

Example 3-1 Shared memory segment for topas

```
[p630n02][/]> ipcs -m
IPC status from /dev/mem as of Thu Oct 28 10:50:04 CDT 2004
T        ID     KEY        MODE       OWNER    GROUP
Shared Memory:
m         0 0x580010da --rw-rw-rw-     root   system
m         1 0x0d00051f --rw-rw-rw-     root   system
m    131074 0xffffffff --rw-rw----     root   system
m         3 0xffffffff --rw-rw----     root   system
m         4 0xffffffff --rw-rw----     root   system
m    655365 0x7800061b --rw-rw-rw-     root   system
```

Since SPMI is the part of Performance Toolbox (PTX®), every metric you can get from **topas** has the same semantics as the ones from PTX. For instance, you can get the description and the maximum/minimum values for EVENTS/QUEUES section of **topas** output shown in Example 3-4 on page 66 also by running the

program compiled from he source code presented in “Spmi_traverse.c” on page 691. This program traverses the data structure provided by SPMI and prints the brief information about each metric. Execution result of this program is listed in the following Example 3-2. For more information about SMPI, refer also to 10.2, “System Performance Measurement Interface” on page 620.

Example 3-2 Descriptions for SPMI metrics

```
...(lines omitted)...
CPU/cpu0/pswitch:Process context switches on this processor:Long/Counter:0-5000
CPU/cpu0/syscall:Total system calls on this processor:Long/Counter:0-2000
CPU/cpu0/read:Read system calls on this processor:Long/Counter:0-1000
CPU/cpu0/write:Write system calls on this processor:Long/Counter:0-1000
CPU/cpu0/fork:Fork system calls on this processor:Long/Counter:0-100
CPU/cpu0/exec:Exec system calls on this processor:Long/Counter:0-100
...(lines omitted)...
Proc/runque:Average count of processes that are waiting for the cpu:Float/Quantity:0-10
Proc/runocc:Number of samplings of runque:Long/Quantity:0-1000000
Proc/swpque:Average count of processes waiting to be paged in:Float/Quantity:0-10
...(lines omitted)...
```

If you need more information about the metrics provided by **topas**, refer to *Performance Toolbox Version 2 and 3 Guide and Reference*, SC23-2625. You can also find the basic command syntax and description of the command in *AIX 5L Version 5.3 Commands Reference, Volume 5*, SC23-4892.

3.1.1 Topas syntax

The following Example 3-3 shows the basic syntax of **topas** command.

Example 3-3 Syntax of topas

```
[p630n02][/]> topas -h

 Usage: topas [-d number_of_monitored_hot_disks]
              [-h show help information]
              [-i monitoring_interval_in_seconds]
              [-m Use monochrome mode - no colors]
              [-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]
              [-L show full-screen Logical Partition display]
              [-U username - show username owned processes with -P]
              [-W show full-screen WLM Display]
```

Tip: By not specifying any flags for the command, topas command runs as though invoked with the following command line:

```
topas -d20 -i2 -n20 -w20 -c20
```

The output of topas execution without flags is shown in Example 3-4.

Example 3-4 Output for topas without flags

```
Topas Monitor for host:    r33n05                EVENTS/QUEUES    FILE/TTY
Thu Oct 28 09:45:07 2004   Interval:  2          Cswitch      131 Readch        2
                                                 Syscall       52 Writech      98
Kernel    0.3   |#                          |    Reads          1 Rawin         0
User      0.0   |#                          |    Writes         1 Ttyout       96
Wait      0.0   |                           |    Forks          0 Igets         0
Idle     99.7   |###########################|    Execs          0 Namei         0
Physc =  0.00                    %Entc=   0.5    Runqueue    0.0 Dirblk        0
                                                 Waitqueue   0.0

Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out
en0       0.1      1.0     1.0     0.0     0.2  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults        0  Real,MB    7167
                                                 Steals        0  % Comp     10.7
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp   1.1
hdisk0    0.0      0.0     0.0     0.0     0.0  PgspOut       0  % Client    1.3
hdisk2    0.0      0.0     0.0     0.0     0.0  PageIn        0
hdisk1    0.0      0.0     0.0     0.0     0.0  PageOut       0  PAGING SPACE
cd0       0.0      0.0     0.0     0.0     0.0  Sios          0  Size,MB     512
                                                                 % Used      1.1
Name            PID  CPU%  PgSp Owner           NFS (calls/sec)  % Free     98.8
topas        557266   0.0   1.4 root            ServerV2       0
rpc.lockd    446510   0.0   0.2 root            ClientV2       0   Press:
netm         368820   0.0   0.0 root            ServerV3       0   "h" for help
IBM.CSMAg    528590   0.0   2.1 root            ClientV3       0   "q" to quit
```

With “-i” flag, you can specify updating interval and you can use the “+/-” keys to modify the sampling interval.

3.1.2 Basic topas output

The basic output of **topas** is composed of two sections. The one is variable (changeable) section in the left most part of the output and the other is static (non-changeable) section in the right most part of the output.

The variable part of the topas display can have one, two, three, four or five subsections. When the topas command is started, it displays all subsections for which hot entities are monitored. The exception to this is the WorkLoad

Management (WLM) Classes subsection. which is displayed only when WLM is active.

CPU Utilization This subsection displays a bar chart showing cumulative CPU usage. Pressing the **c** key only once will turn this subsection off. This output can display either global CPU utilization or a list of hot CPUs. You can toggle between these two outputs buy press **c** key twice.

Network Interfaces This subsection displays a list of hot network interfaces. The maximum number of interfaces displayed is the number of hot interfaces being monitored, as specified with the -n flag. Pressing the n key turns off this subsection. Pressing the **n** key again shows a one-line report summary of the activity for all network interfaces.

Physical Disks This subsection displays a list of hot physical disks. The maximum number of physical disks displayed is the number of hot physical disks being monitored as specified with the **-d** flag. Pressing the d key turns off this subsection. Pressing the **d** key again shows a one-line report summary of the activity for all physical disks.

WLM Classes This subsection displays a list of hot WorkLoad Management (WLM) Classes. The maximum number of WLM classes displayed is the number of hot WLM classes being monitored as specified with the **-w** flag. Pressing the **w** key turns off this subsection.

Processes This subsection displays a list of hot processes. The maximum number of processes displayed is the number of hot processes being monitored as specified with the **-p** flag. Pressing the **p** key turns off this subsection. The process are sorted by their CPU usage over the monitoring interval.

The Static section contains five subsections of statistics as follows:

EVENTS/QUEUES Display the per-second frequency of selected system-global events and the average size of the thread run and wait queues

FILE/TTY Displays the per-second frequency of selected file and tty statistics.

PAGING Display the per-second frequency of paging statistics.

MEMORY	Displays the real memory size and the distribution of memory in use.
NFS	NFS stats in calls per second

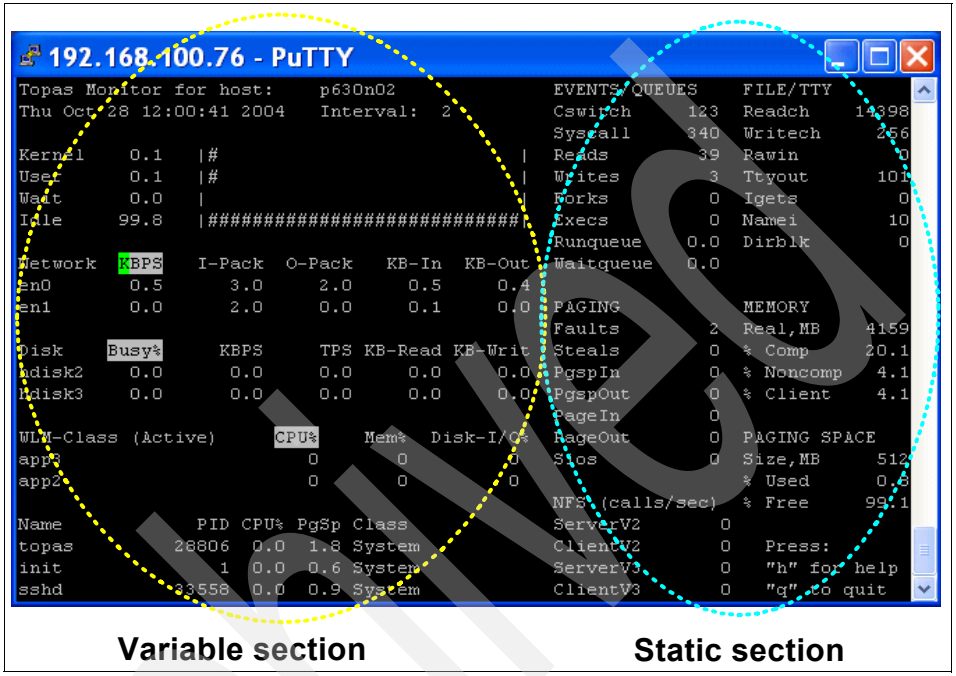

Figure 3-1 The basic output of Topas command

Topas provides you have additional screen outputs regarding to partition statistics, detailed WLM information and detailed process information (this output looks very similar to one of **top** command).

3.1.3 Partition statistics

Topas command in AIX 5L Version 5.3 supports Micro-Partitioning™ and simultaneous multi-threading (SMT) environments, and reports status of the partition. You can see sample screen output in a partitioned environment in Example 3-5. Pressing the **P** key from the basic topas screen switches to the partition statistics screen. Pressing the **P** key again gets out of this screen and goes back to the basic topas screen. You can also specify the **-L** flag when you run the **topas** command.

Example 3-5 Sample output for topas with partition statistics

```
Interval:    2    Logical Partition: r33n05          Thu Oct 28 09:56:16 2004
```

```
Psize:       -                Shared SMT  ON           Online Memory:   7168.0
Ent: 0.50                      Mode: UnCapped          Online Logical CPUs:  2
Partition CPU Utilization                              Online Virtual CPUs:  1
%usr %sys %wait %idle physc %entc %lbusy   app   vcsw phint %hypv   hcalls
   0    0     0   100   0.0   0.40  0.00     -    512     0   0.0        0
================================================================================
LCPU  minpf majpf  intr   csw icsw runq lpa scalls usr sys _wt idl   pc   lcsw
Cpu0      0     0   338   264  125    1 100     40   7  62   0  31 0.00    256
Cpu1      0     0    20     0    0    0   0      0   0   9   0  91 0.00    256
```

Detailed WorkLoad Management information

You can get more detailed information about WLM by using **topas** as well. This output also contains detailed process information. You can see sample screen output of this in Example 3-6. Pressing the **W** key from the basic **topas** screen switches to partition statistics screen. Pressing the **W** key get out of this screen and go back to the basic topas screen. You can also specify **-W** flag when you run **topas** command.

Example 3-6 Sample output for topas with detailed WLM information

```
Topas Monitor for host:    p630n02     Interval:  10    Thu Oct 28 09:57:25 2004
WLM-Class (Active)             CPU%      Mem%     Disk-I/O%
app3                           0         0        0
app2                           0         0        0
app1.app5                      0         0        0
app1.Shared                    0         0        0
app1.Default                   0         0        0
app1                           0         0        0
System                         0         10       0
Shared                         0         1        0
Default                        0         0        0
Unmanaged                      0=======  15=========  0====================
Unclassified                   0A  TEXT  0AGE         0     PGFAULTS
USER       PID    PPID PRI NI   RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root     17096   11726  60 20   537   158   548   13:18  0.0  163 1173 IBM.CSMAg
root      8536       1  60 20   125     2   125    2:10  0.0   1812909 syncd
root     14726   11726  60 20   147   417   309    1:08  0.0  335  893 i4llmd
root      3354       0  37 41    17     0    17    0:51  0.0    0   23 gil
root     33842   28372  60 20   200   145   468    0:17  0.0  18038992 dtfile
root     28874   35464  60 20 11305    13 11305    0:16  0.0    411526 java
root     12146   11726  60 20   160    78   160    0:13  0.0  106  532 aixmibd
root      7268       1  60 20     6     0     6    0:06  0.0    0    3 rt-fcparr
root     18604   11726  39 20   538   139   539    0:04  0.0  412 2661 rmcd
root      4470   11726  60 20    30    97   124    0:04  0.0  107 1270 sshd
```

Detailed process information

Topas provided the output more focused on process information. This output looks similar to the output of **top** command (see Example 3-7 on page 70). Pressing the **P** key from the basic **topas** screen switches to partition statistics screen. Press the **P** key again to get out of this screen and go back to the basic topas screen. You can also specify **-P** flag when you run **topas** command.

Example 3-7 Sample output for topas with detailed process information

```
Topas Monitor for host:    r33n05      Interval:  10    Thu Oct 28 15:30:58 2004

                                   DATA  TEXT  PAGE              PGFAULTS
USER       PID    PPID PRI NI    RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root    528590  536586  60 20    527   159   539    9:07  0.0  166 1126 IBM.CSMAg
root    548876       1  60 20    106    17   106    0:31  0.0    0  274 getty
root    372918       0  37 41     29     0    29    0:23  0.0    0   30 gil
root    426230       1  60 20    127     2   127    0:19  0.0 1831381 syncd
root    479340  536586  60 20    141    15   141    0:17  0.0   46  504 muxatmd
root    446510       1  60 20     50     0    50    0:03  0.0    0   56 rpc.lockd
root    364722       0  60 41     12     0    12    0:03  0.0    0    4 xmgc
root    487588  536586  39 20    537   133   538    0:01  0.0  511 2834 rmcd
root    368820       0  36 41     12     0    12    0:00  0.0    0 26902 netm
root    344232       0  40 41     12     0    12    0:00  0.0    0    4 vmptacrt
root    377016       0  16 41     12     0    12    0:00  0.0    0   14 wlmsched
root    389138       0  60 20     12     0    12    0:00  0.0    0    3 rtcmd
root    397522       0  60 20     12     0    12    0:00  0.0    0    4 lvmbb
root    401616       0  38 41     49     0    49    0:00  0.0    0   90 j2pg
root    409828       0  60 20     19     0    19    0:00  0.0    0   15 dog
root    413826  536586  60 20    322    59   322    0:00  0.0   58  591 IBM.HostR
root    418000  536586  60 20    279    25   279    0:00  0.0   44  517 IBM.DRMd
root    422114       1  39 41     12     0    12    0:00  0.0    0    3 aioserver
root    340134       0  16 41     15     0    15    0:00  0.0    0   11 lrud
root    430306       1  60 20    137    22   137    0:00  0.0   14  159 errdemon
```

3.2 The jtopas utility

The jtopas tool is a Java™ based system-monitoring tool that provides a console to view a summary of the overall system, as well as separate consoles to focus on particular subsystems. Top instruments are featured in the jtopas tool for various resources, such as processes, disks, etc. The data streams available are “Near Real-Time (NRT)” and “Playback” (PB). PB data can be viewed from the local host or a remote host, as long as Performance Toolbox for AIX has been installed and configured.

The jtopas tool interface displays a set of tabs that represent the various consoles. The main console provides a view of several resources and subsystems and lends itself to providing an overall view of a computer system, while the other consoles focus more on particular areas of the system. The main console contains several top instruments.

A top instrument is a monitoring window that displays a group of devices or processes. For instance, these top instruments can be sorted by the largest consumers of a system resource, such as memory, CPU, storage, or network adapters. Even though there might be thousands of processes, for example, only the top 10 or 20 are displayed by the jtopas tool.

Each of the other consoles is composed of one or more instruments. An instrument is similar to a window that can be resized, minimized, or moved. A divider bar is used to separate top instrument information from global information about the system, and the bar can be moved or either side of the bar can be made to use the entire console display area.

At initialization, the jtopas tool displays all consoles with their instruments. If a user configuration file is found, the consoles are constructed based on that file. Otherwise, the default configuration is used. By default, the jtopas tool tries to establish a communication link with the local host to drive the consoles.

To run the jtopas tool, type:

```
jtopas
```

When starting jtopas the Java interface is started you will see an image similar to Figure 3-2 on page 72.

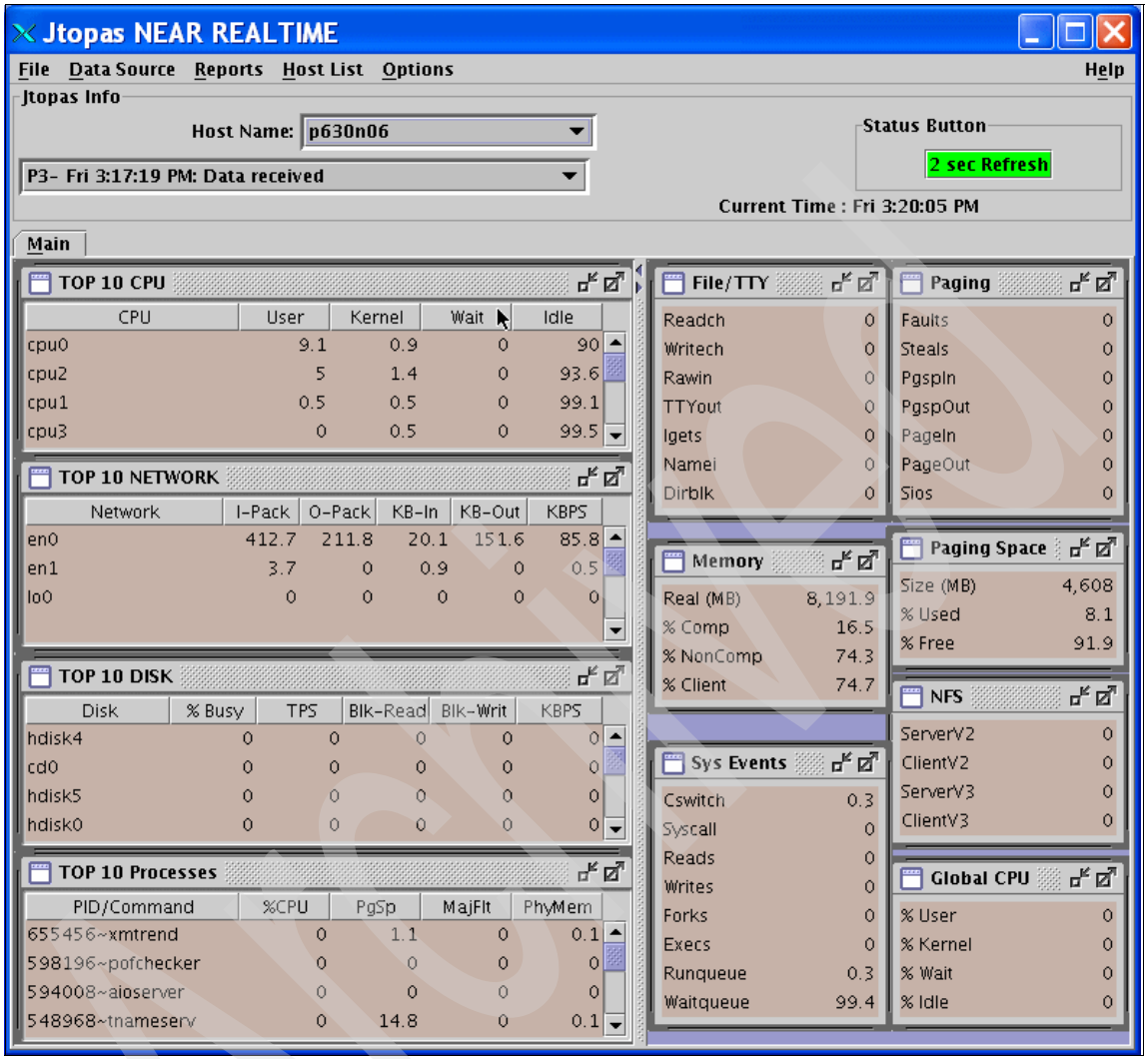

Figure 3-2 jtopas main screen

The jtopas tool uses recording files and a configuration file, as follows:

Recording Files

Recording files contain metric values recorded by an instance of the xmtrend agent, acting as the top agent. This xmtrend agent is directed to record metric data specifically for top data. The xmtrend agent creates a recording file of top metric data as defined in the jtopas.cf configuration file. This recording file can be used by the jtopas tool to display historical system events, or by the **`jazizo`** trend analysis tool.

Not all data and data rates are available to the jtopas tool during a playback. For top recordings and Near Real-Time data, the **xmtrend** daemon must be started with the -T option. The top recordings are placed in the /etc/perf/Top/ directory.

3.2.1 The jtopas configuration file

The jtopas tool uses a default configuration file that determines the size, location, and metrics viewed for each instrument. If any instrument is changed, upon exit, users are asked if they want to save the current configuration. If Yes is selected, a configuration file is placed in the user's HOME directory and is named ".jtopas.cfg". Users can return to using the default configuration by deleting the /$HOME/.jtopas.cfg file.

As can be seen in Figure 3-2 on page 72, jtopas has the following menus:

File Menu Closes all windows and exits the jtopas tool. If the configuration has changed, the user is asked whether to save the new configuration

Data Source Menu The data source menu contains two options:

Near Real-Time: Data Changes the data stream to near real-time data. Near real-time data is gathered from a machine in real time and then made available to the jtopas tool. The refresh rate, which can be changed in the jtopas tool, defines how often data is requested and displayed.

PlayBack Data: Changes the data stream to PlayBack data. The PlayBack control panel is displayed when users select this option. The jtopas tool continues to display data at the refresh rate. The data is gathered from the local or a remote machine. Recorded data is saved on a server by the xmtrend agent at 1-minute intervals. Although the refresh rate updates the console at a given interval by default, the clock associated with the data increments at the 1-minute interval. For example, if the refresh rate is every 5 seconds and the recording file is recorded every minute, the data and clock on the PlayBack panel refreshes every 5 seconds by 1 minute

Reports Menu The Reports menu provides a set of report formats. Each report summarizes the data in a tabular format that can be viewed and printed. The font and size of the data can be changed. Some reports might offer report options to change how the data is summarized and displayed.

Host List The Host List menu allows users to add or delete a host that can be monitored by jtopas.

Options Menu Options menu contains two options:

Refresh Rate: The jtopas tool cycles through at the refresh rate. The cycle includes requesting the data and updating the console. The refresh rate can be changed by either clicking the refresh rate/status button or selecting the menu option. The user can enter values of whole seconds. The jtopas tool uses the default refresh rate. The greater the refresh rate value, the less load the jtopas tool consumes on the CPU. If the jtopas tool is unable to complete an operation within the cycle time, the status button turns yellow and an appropriate message is displayed. If data cycles are consistently missed, the refresh rate should be adjusted to increase the time between updates.

Message Filter: The message filter option allows users to filter out and display messages based on a specific priority. The following are priorities for messages, each priority having a color associated with it:

Priority 1: Red - Critical message, such as losing a host connection

Priority 2 Yellow - Important message, such as losing a data cycle

Priority 3 Black - Informational messages The text of each message displayed is color-coded and is preceded by the priority and the timestamp.

3.2.2 The info section for the jtopas tool

The info section provides status information and allows users to select the host from which to gather the data. The following are the data fields:

Host Name: By default, the local host name is displayed. Host names can be added, deleted, or selected.

To add a new host, select Host List from the menu bar and then select Add Host. The new host is immediately contacted for a connection and is added to the host list pull-down. If the host list is modified in any way, upon exit, the user is asked whether to save the new configuration. If OK is selected, the new host list is saved in the

$HOME/.jtopas.cfg file and made available the next time the same user starts the jtopas tool.

To delete a host, select Host List from the menu bar and then select **Delete Host**. The old host will still remain selected until a new host is selected.

To select a new host from the host list, open the list and select the host name.

Message Section

The **`jtopas`** tool generates informational messages. These messages are assigned a priority to classify them by importance and to allow users to hide messages of a particular priority for easier viewing. As stated in the Message Filter section of the Options menu, the following priorities are assigned to messages: P1, P2, or P3. The highest in importance is P1, as it is used for critical messages. Messages can be filtered by selecting Message Filter under the Options menu. Status/Refresh Rate Button

The status button reflects the status of data acquisition per the selected refresh rate. The refresh rate defines how often the console data is updated. The value is in seconds. The refresh rate can be changed by selecting the button or selecting Refresh Rate under the Options menu. If data is not retrieved and updated within the refresh cycle, the button turns yellow and the button label changes to No Update. If the data connection is lost, the button turns red and the button label displays No Data. Appropriate messages are also added to the message section.

Current Time

This field reflects the current day and time.

3.2.3 The jtopas consoles

In Figure 3-2 on page 72 the instruments is displayed as a window that can be minimized, maximized, moved, and resized. If there are multiple columns with headers, the columns can be reorganized and resized. Some instruments implement a scroll bar to view additional data.

Top instruments monitor a group of common metrics ordered by a particular column metric. For example, CPUs are by default ordered highest to lowest by largest consumer of kernel CPU used. This default can be changed to largest consumer of user CPU by clicking the User header label. Even if there are 64 CPUs, only a subset is displayed.

3.2.4 The jtopas playback tool

When the PlayBack data source is selected. The Playback Control panel appears. Figure 3-3 shows the jtopas PlayBack panel. The panel allows a user to control the playback. Closing the PlayBack panel returns the user to the NRT data source.

Figure 3-3 jtopas playback control panel

Playbacks begin in a paused state. To begin displaying the playback, click Play. The PlayBack panel contains the following information:

Host Name The initial playback host is the host that was selected for the NRT data. This can be changed in the same manner as it is changed in the main console.

Start / Stop The available start and stop times of all recorded data on a particular host are displayed. By clicking Change, the start and stop date and times can be altered. The Time Selection panel displays dates and times of available recorded data. Select a date and indicate whether it is the start or stop date for the playback. Then select a start time and stop time. Click OK to use the dates and times selected.

PlayBack Time This time stamp represents the time stamp for the playback sample that is displayed.

Sample Interval Even though the recording frequency is in minutes, metric samples are taken at a much finer granularity. These samples are combined to determine the mean across the recording cycle. By default, sample updates to the jtopas tool in the playback mode are at the recording frequency. This is not the same as the refresh rate of the screen. The refresh rate represents how often the data in the jtopas console is refreshed. Having a refresh rate for the console, as well as a sample interval, allows the user to view a week's worth of data in hourly intervals and have

	the console refresh at a rate that is comfortable to view and analyze.
PlayBack Controls	The following are the playback controls:
Rewind	Plays the recording back in reverse. The sample interval value becomes negative, which indicates that the recording file is being traversed in reverse order and at the interval displayed. Each time Rewind is selected, the time interval increases. Clicking Play returns the playback to the default sample rate.
Play	Displays the recording file.
Fast Forward	Increases the time between data samples. The sample interval value increases, which indicates that the recording file is being traversed at greater intervals. Each time Fast Forward is selected, the time interval increases. Clicking Play returns the playback to the default or selected sample rate.
Pause	Stops the playback but maintains the current playback time in the recording file.
Stop	Stops the playback and resets the playback time to the beginning.
Step Forward	Moves the playback forward one time interval and pauses.
Step Backward	Moves the playback backward one time interval and pauses.

3.3 The perfpmr utility

perfpmr consists of a set of utilities that collect the necessary information to assist in analyzing performance issues. It is primarily designed to assist IBM software support, but is also useful to document your system during implementation and validation phases.

This tool contains a series of programs that use performance monitoring commands and tools existing on the system, and collect the data in a file which can be sent to IBM support, or saved for further reference.

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`

Download the version that is appropriate for your AIX level. In our case (AIX 5L V5.3), we downloaded the file from:

ftp://ftp.software.ibm.com/aix/tools/perftools/perfpmr/perf53/perf53.tar.Z

Syntax

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

-P preview only     - show scripts to run and disk space needed

-D                  run perfpmr the original way without a perfpmr cfg
                    file

-I                  get lock instrumented trace also

-g                  do not collect gennames output.

-f                  if gennames is run, specify gennames -f.

-n                  used if no netstat or nfsstat desired.

-p                  used if no pprof collection desired while
                    monitor.sh running.

-s                  used if no svmon 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              sec is time to wait before starting collection
                    period default is delay_seconds 0 monitor_seconds
                    is for the monitor collection period in seconds
```

For example, you can use **perfpmr.sh 600** for standard collection period of 600 seconds.

3.3.1 Information about measurement and sampling

The **perfpmr.sh 600** command executes the following shell scripts to obtain a test case. You can also run these scripts independently.

aiostat.sh Collects AIO information into a report called aiostat.int

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.

lpartstat.sh Builds a report on Logical partitioning information, two file are created lparstat.in and lparstat.sum

monitor.sh time Invokes system performance monitors and collects interval and summary reports:

mpstat Builds a report on Logical processor information into a report called mpstat.int

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 4.2.14, “The pprof command” on page 262 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.
svmon.sh	Builds a report on **svmon** data into two files svmon.out and svmon.out.S
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 collected by **trace**, 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** independently for a period of 15 seconds when the problem is present. For example, the command **trace.sh 15** runs a trace for 15 seconds.

An IBM @server pSeries system running AIX can produce a test case (the total data collected by perfpmr) 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*
```

3.3.2 Building and submitting a test case

You may be asked by IBM to supply a test case for a performance problem or you may want 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 want 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
- aiostat
- aiostat.sh
- config.sh
- emstat.sh
- filemon.sh
- getdate
- getevars
- iostat.sh
- iptrace.sh
- lparstat.sh
- lsc
- memfill
- monitor.sh
- mpstat.sh
- netstat.sh
- nfsstat.sh
- perf53.tar
- perfpmr.cfg
- perfpmr.sh

- pprof.sh
- ps.sh
- quicksnap.sh
- sar.sh
- setpri
- setsched
- svmon.sh
- 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**

The output of the installation procedure will be similar to Example 3-20.

Example 3-8 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 failover.

Tip: After you have installed **perfpmr** you can run it at any time to make sure that all of the files 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.

3.3.3 Examples for perfpmr

Example 3-9 shows the output of the data collected while running the **perfpmr.sh** program.

Example 3-9 Running perfpmr.sh

```
[p630n04][/home/hennie/perf/scripts]> perfpmr.sh 600

(C) COPYRIGHT International Business Machines Corp., 2000,2001,2002,2003,2004

    PERFPMR: perfpmr.sh Version 530 2004/10/06
    PERFPMR: current directory: /home/hennie/perf/scripts
    PERFPMR: perfpmr tool directory: /home/hennie/perf
    PERFPMR: Parameters passed to perfpmr.sh:
    PERFPMR: Data collection started in foreground (renice -n -20)

     TRACE.SH: Starting trace for 5 seconds
/bin/trace -k 10e,254,116,117  -f -n -C all -d -L 10000000 -T 10000000 -ao
trace.raw
     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

     TRACE.SH: Enabling locktrace
```

```
lock tracing enabled for all classes
     TRACE.SH: Starting trace for 5 seconds
/bin/trace -j 106,10C,10E,112,113,134,139,465,46D,606,607,608,609 -f -n -C all
-d -L 10000000 -T 10000000 -ao trace.raw.lock
     TRACE.SH: Data collection started
     TRACE.SH: Data collection stopped
     TRACE.SH: Trace stopped
     TRACE.SH: Disabling locktrace
lock tracing disabled for all classes
     TRACE.SH: Binary trace data is in file trace.raw

     MONITOR: Capturing initial lsps, svmon, and vmstat data
     MONITOR: Starting system monitors for 600 seconds.
     MONITOR: Waiting for measurement period to end....
iostat: 0551-157 Asynchronous I/O not configured on the system.

MONITOR: Capturing final lsps, svmon, 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 40086.
0513-044 The iptrace Subsystem was requested to stop.
     IPTRACE: iptrace collected....
     IPTRACE: Binary iptrace data is in file iptrace.raw

     TCPDUMP: Starting tcpdump for 10 seconds....
kill: 41054: no such process
     TCPDUMP: tcpdump collected....
     TCPDUMP: Binary tcpdump data is in file tcpdump.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
     CONFIG.SH: Report is in file config.sum

PERFPMR: Data collection complete.
[p630n04][/home/hennie/perf/scripts]>
```

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.

3.4 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.

PDT Syntax

To start the PDT configuration, enter:

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

The **pdt_config** is a menu-driven program. Refer to 3.4.1, “Examples for PDT” on page 87 for PDT usage.

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.

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

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.

3.4.1 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 3-10 shows the PDT main menu.

Example 3-10 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 3-11.

Example 3-11 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 3-11 on page 88 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 3-12.

Example 3-12 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 3-13.

Example 3-13 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.

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 3-14.

Example 3-14 .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. Each line in the .nodes file should only contain either a hostname or an IP address. In the following example, we will monitor the connection to the Domain Name Server (DNS). Example 3-15 shows how to check which nameserver a DNS client is using by examining the /etc/resolv.conf file.

Example 3-15 ./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 3-16 on page 90.

Example 3-16 .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 3-17 is the content of the default file.

Example 3-17 .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 MEMORY_FACTOR * 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 last_value * 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.

3.4.2 Using reports generated by PDT

Example 3-18 shows the default-configured level 3 report. It is an example of what will be delivered by e-mail every day.

Example 3-18 PDT sample e-mail report

```
Performance Diagnostic Facility 1.0

 Report printed: Fri Nov 5 11:14:27 2004

 Host name: lpar05
 Range of analysis includes measurements
 from: Hour 10 on Friday, November 5th, 2004
 to: Hour 11 on Friday, November 5th, 2004


 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 3-19 shows the raw information from the .SM_RAW_REPORT file that is used for creating the PDT_REPORT file.

Example 3-19 .SM_RAW_REPORT file

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

H 1  | Report printed: Fri Nov 5 10:00:00 2004
H 1  |

H 1  | Host name: lpar05

H 1  | Range of analysis includes measurements

H 1  |    from: Hour 10 on Friday, November 5th, 2004

H 1  |    to: Hour 11 on Friday, November 5th, 2004
H 1  |
...(lines omitted)...
```

The script in Example 3-20 shows how to extract report subsections from the PDT_REPORT file. In this example it displays all subsections in turn.

Example 3-20 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 3-21 shows a sample output from the script in Example 3-20 using the same data as in Example 3-18 on page 92.

Example 3-21 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 (hdis

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 %
```

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.

3.4.3 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
```

3.5 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.

The **curt** command is located at in /usr/bin/curt and is part of the bos.perf.tools fileset that is obtained from the AIX base installation media.

Syntax

The syntax for the **curt** command is:

```
curt -i inputfile [-o outputfile] [-n gensymsfile] [-m trcnmfile] [-a
      pidnamefile] [-f timestamp] [-l timestamp] [-r PURR] [-ehpstP]
```

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.
-r PURR	Uses the PURR register to calculate CPU times.
-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.

3.5.1 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 3-1 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 3-1 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.
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.

3.5.2 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 perfpmr command for details, or alternatively, you can run **trace** as shown in Example 3-34 on page 117 (see 3.7.3, “How to start and stop trace” on page 155 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 3-12 on page 89, 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 3-12 on page 89.
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 **curt** to output more names, especially interrupt handlers. **gennames** requires root authority to run. **trcnm** can be run by any user.
4. Generate the **curt** output.

Example 3-22 Creating a trace file for curt 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 curt” can be used in place of “-j $HOOKS” for the **trace** command from Example 3-12 on page 89.

3.5.3 Overview of the reports generated by curt

The following is an overview of the reports that can be generated by the curt 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 119 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 3-34 on page 113 for this report.

-s Produces a report that includes the statistics displayed in 3.5.4, "The default report" on page 101, and includes a report on errors returned by system calls. Refer to Example 3-35 on page 115 for this report.

-t Produces a report that includes the statistics displayed in 3.5.4, "The default report" on page 101, 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 3-36 on page 116 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 3-37 on page 118.

3.5.4 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 3-23.

Example 3-23 General information from curt.out

```
Run on Mon Nov 15 17:26:06 2004
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 Nov 15 17:12:14 2004

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 3-24.

Example 3-24 The System Summary report from curt.out

```
                System Summary
                --------------

  processing       percent       percent
  total time     total time     busy time
      (msec)  (incl. idle)  (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 3-24 on page 102 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 3-29 on page 108, 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 3-35 on page 115.

Example 3-25 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               IDLE
  -----------    ----------
      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 3-26.

Example 3-26 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.

  `syacall` 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 3-26 on page 105. 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 3-26 on page 105 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 4, “CPU analysis and tuning” on page 171) or other profiling tools to profile the process, or look directly at the formatted trace file from the **`trcrpt`** command. (See 3.7.12, “The trcrpt command” on page 165.)

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 3-27, 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 3-27 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 3-28.

Example 3-28 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 3-29.

Example 3-29 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 3-30. The list is sorted by the total time in milliseconds consumed by each type of system call.

Example 3-30 The System Calls Summary from curt.out

```
                    System Calls Summary
                    --------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  SVC (Address)
               (msec)    time    (msec)    (msec)    (msec)
========  ===========  ======  ========  ========  ========  ================
     605     355.4475   1.74%    0.5875    0.0482    4.5626  kwrite(4259c4)
     733     196.3752   0.96%    0.2679    0.0042    2.9948  kread(4259e8)
       3       9.2217   0.05%    3.0739    2.8888    3.3418  execve(1c95d8)
      38       7.6013   0.04%    0.2000    0.0051    1.6137  __loadx(1c9608)
    1244       4.4574   0.02%    0.0036    0.0010    0.0143  lseek(425a60)
      45       4.3917   0.02%    0.0976    0.0248    0.1810  access(507860)
      63       3.3929   0.02%    0.0539    0.0294    0.0719  _select(4e0ee4)
       2       2.6761   0.01%    1.3380    1.3338    1.3423  kfork(1c95c8)
     207       2.3958   0.01%    0.0116    0.0030    0.1135  _poll(4e0ecc)
     228       1.1583   0.01%    0.0051    0.0011    0.2436  kioctl(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 3-31 displays the pending system calls summary.

Example 3-31 Pending System Calls Summary from curt.out

```
                    Pending System Calls Summary
                    ----------------------------
Accumulated   SVC (Address)              Procname (Pid  Tid)
Time (msec)
============  ========================  ==========================
      0.0656  _select(4e0ee4)           sendmail(7844 5001)
      0.0452  _select(4e0ee4)           syslogd(7514 8591)
      0.0712  _select(4e0ee4)           snmpd(5426 9293)
      0.0156  kioctl(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 3-32.

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 3-32 The Flih summaries from curt.out

```
                    Global Flih Summary
                    -------------------
      Count  Total Time    Avg Time    Min Time    Max Time  Flih Type
                 (msec)      (msec)      (msec)      (msec)
     ====== =========== =========== =========== ===========  =========
       2183    203.5524      0.0932      0.0041      0.4576   31(DECR_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    Avg Time    Min Time    Max Time  Flih Type
                 (msec)      (msec)      (msec)      (msec)
     ====== =========== =========== =========== ===========  =========
         635     39.8413      0.0627      0.0041      0.4576   31(DECR_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    Avg Time    Min Time    Max Time  Flih Type
                 (msec)      (msec)      (msec)      (msec)
     ====== =========== =========== =========== ===========  =========
           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 3-32 on page 111, 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 3-33.

The `Global Slih Summary` lists the total of second level interrupts on the system, while the `Per CPU Slih Summary` lists the second level interrupts per CPU.

Example 3-33 The Slih summaries from curt.out

```
                Global Slih 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 3.5.4, “The default report” on page 101, and also includes additional information in the System Calls Summary report as shown in Example 3-34. The additional information pertains to the total, average, maximum, and minimum elapsed times a system call was running.

Example 3-34 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)         (msec) (msec) (msec)   (msec)    (msec)    (msec)   (msec)
===== ======== ===== ====== ====== ====== ========== ========= ========= ========= ====================
  605 355.4475 1.74% 0.5875 0.0482 4.5626 31172.7658   51.5252    0.0482  422.2323 kwrite(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 kioctl(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 3-30 on page 109:

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 3.5.4, “The default report” on page 101 and includes reports on errors returned by system calls, as shown in Example 3-35.

Example 3-35 curt output with the -s flag

```
# curt -s -i trace.r -m trace.nm -n gennames.out -o curt.out
# cat curt.out
...(lines omitted)...
                    Errors Returned by System Calls
                    ------------------------------

Errors (errorno : count : description) returned for System call:
socket_aio_dequeue(0x11e0d8)
     11 :      485 : "Resource temporarily unavailable"
  Errors (errorno : 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 3.5.4, “The default report” on page 101, 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 3-36.

Example 3-36 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

                            Thread System Call Data
     Count   Total Time     Avg Time     Min Time     Max Time     SVC (Address)
                 (msec)       (msec)       (msec)       (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   SVC (Address)
  Time (msec)
  ============  ==========================
        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 redispatches 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 3-42 on page 128.) 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 undispatched 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 3-37.

Example 3-37 curt 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)
```

```
========  ===========  ======  ========  ========  ========  ================
      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.

3.6 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.

3.6.1 splat syntax

The syntax for the **splat** command is:

```
splat -i file [-n file] [-o file] [-d [ bfta ] ] [-l address] [-c class]
   [-s [ acelmsS ] ] [-C cpus] [-S count] [-t start] [-T stop][-p]
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.

-p Specifies the use of the PURR register to calculate CPU times.

-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 /usr/include/sys/lockname.h.

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

3.6.2 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**. 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 3-2.

Table 3-2 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.

Hook ID	Event name	Event explanation
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.
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.

3.6.3 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 3-50 on page 144 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**.

3.6.4 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.

3.6.5 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 3-40 on page 127; 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 3-41 on page 130 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.

3.6.6 splat examples

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.

Execution summary

Example 3-38 shows a sample of the Execution summary. This report is generated by default when using **splat**.

Example 3-38 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: Wed Apr 21 09:55:22 2004


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.

Gross lock summary

Example 3-39 shows a sample of the gross lock summary report. This report is generated by default when using **splat**.

Example 3-39 Gross lock summary

```
                    Unique          Acquisitions    Acq. or Passes  % Total System
                Total Addresses     (or Passes)     per Second      '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 *acquired* during the analysis interval; for PThread condition-variables, the total number of times the condition *passed* 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 *cumulative CPU time*, 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 (*).

Per-lock summary

Example 3-40 shows a sample of the per-lock summary report. This report is generated by default when using **`splat`**.

Example 3-40 Per-lock summary report

```
100 max entries, Summary sorted by Acquisitions:

                      T  Acqui-
                      y  sitions                                  Locks or           Percent Holdtime
Lock Names,           p  or                                       Passes     Real    Real      Comb      Kernel
Class, or Address     e  Passes  Spins   Wait    %Miss     %Total  / CSec     CPU     Elapse    Spin      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 ... )
000000002FF22B70        L   368838  0       N/A     0.0000    100.000   9622.964   99.9865   99.9865   0.0000
00000000F00C3D74        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 **`splat`** could map the address from a name file. See 3.6.5, “Address-to-name resolution in splat” on page 124 for an explanation.

`Type`
: The type of the lock, identified by one of the following letters:

  - `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 *wait* state for that lock or condition-variable during the analysis interval.

`%Miss` The percentage of access attempts that resulted in a *spin* 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 *cumulative CPU time*. This is a measure of the acquisition frequency of the lock.

`Real CPU` The percentage of the *cumulative CPU time* 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 *elapsed real time* 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 *cumulative CPU time* that executing threads spent spinning on the lock. Note that the PThreads library currently uses *waiting* 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 *not recoverable* for PThreads.

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 3-41 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 3-41 AIX SIMPLE lock

```
[AIX SIMPLE Lock]                CLASS:  NETISR_LOCK_FAMILY.FFFFFFFF
ADDRESS: 0000000000535378         KEX: unix
NAME:        netisr_slock
================================================================================
        |                                 |                 | Percent Held ( 18.330873s )
Acqui-  |  Miss  Spin   Wait   Busy   |    Secs Held     |  Real  Real    Comb  Real
sitions |  Rate  Count  Count  Count  |CPU      Elapsed  |  CPU  Elapsed  Spin  Wait
471     |  0.000 0      0      0      |0.002584 0.002584 |  0.01   0.01   0.00   0.00
--------------------------------------------------------------------------------
%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 *spinning* on the lock, whether executing or suspended, across the analysis interval.

`WaitQ` The minimum, maximum, and average number of threads *waiting* 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 3-4 on page 132 shows the states that a thread can be in with respect to the given simple or complex lock.

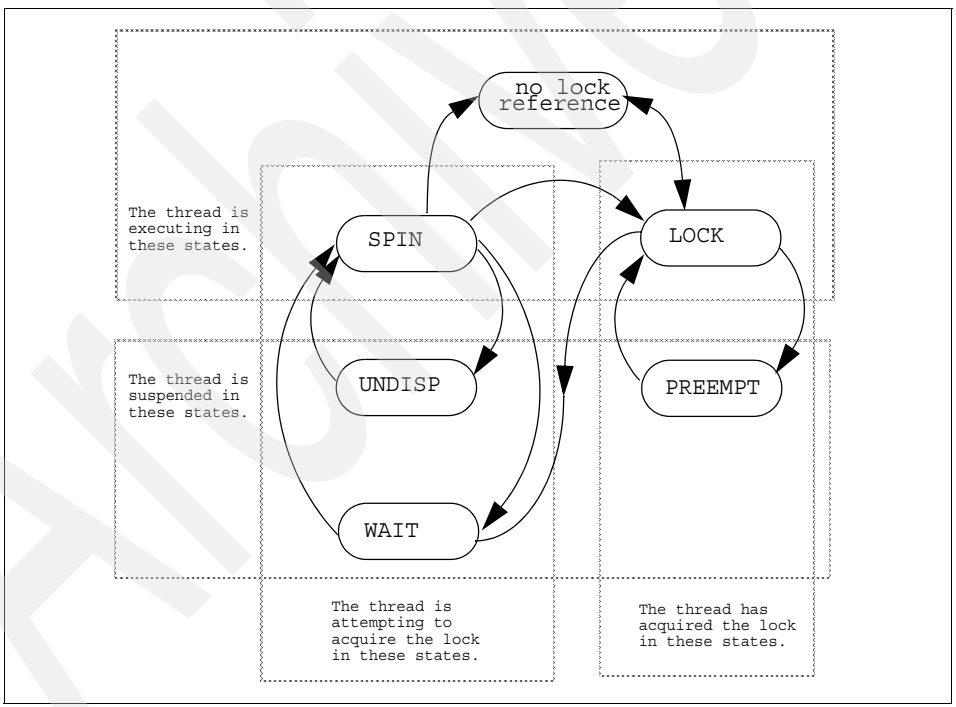

Figure 3-4 Lock states

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 undispatched 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 undispatched.

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 3-42 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 3-42 Function detail report for the simple lock report

```
                    Acqui-    Miss  Spin  Wait    Busy  Percent Held of Total Time
Function Name       sitions   Rate  Count Count   Count CPU     Elapse Spin  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
          ^^^^^^^^^^^^^^^   ^^^^^^^^^^^^^^^^^    ^^^^^^^^
          000000000001AA54  0000000000000000     0001AA54
          000000000001A494  0000000000000000     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 simple_lock, simple_lock_try, simple_unlock, disable_lock, or unlock_enable), 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 simple_lock_try() 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.

`Elapse(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 3-43 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 3-43 Thread detail report

ThreadID	Acqui-sitions	Miss Rate	Spin Count	Wait Count	Busy Count	Percent Held of Total Time CPU	Elapsed	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 *thread 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 *wait* 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 3-44.

Example 3-44 Complex lock report (top part)

```
[AIX COMPLEX Lock]                CLASS:      TOD_LOCK_CLASS.FFFF
ADDRESS: 0000000000856C88         KEX: unix
NAME:           tod_lock
================================================================================
        |                                |                 | Percent Held ( 15.710062s )
Acqui-  |  Miss  Spin   Wait   Busy   |    Secs Held    |  Real  Real    Comb  Real
sitions |  Rate  Count  Count  Count  |CPU      Elapsed |  CPU  Elapsed  Spin  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
```

```
          Min   Max   Avg  |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 3-45.

Example 3-45 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 3-46.

Example 3-46 Complex lock report (function and thread detail)

```
              Acquisitions Miss   Spin Count   Wait Count   Busy   Percent Held of Total Time
Function Name Write  Read   Rate   Write  Read  Write  Read  Count  CPU    ElapseSpin  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



          Acquisitions Miss   Spin Count   Wait Count   Busy   Percent Held of Total Time
ThreadID  Write  Read   Rate   Write  Read  Write  Read  Count  CPU    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
```

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 3-47 shows an example of the header.

Example 3-47 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 3-48.

Example 3-48 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
================================================================================
        |                              |                | Percent Held ( 15.710062s )
Acqui-  |  Miss  Spin   Wait   Busy    |    Secs Held   |  Real  Real    Comb  Real
sitions |  Rate  Count  Count  Count   |CPU      Elapsed|  CPU  Elapsed  Spin  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 *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 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:	
	`SpinQ`	The minimum, maximum, and average number of threads *spinning* on the lock, whether executing or suspended, across the analysis interval.
	`WaitQ`	The minimum, maximum, and average number of threads *waiting* 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 3-49.

Example 3-49 PThread mutex report (thread detail)

```
   Acqui-     Miss  Spin   Wait   Busy       Percent Held of Total Time
```

```
PThreadID  sitions    Rate  Count  Count  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 *PThread 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 *wait* 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` contains 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.

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 Example 3-50.

Example 3-50 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
================================================================================
         |                     |                  | Percent Held (383.290027s )
Acqui-   |  Miss  Spin   Wait  |    Secs Held     |  Real  Real    Comb  Real
sitions  |  Rate  Count  Count |CPU      Elapsed  |  CPU  Elapsed  Spin  Wait
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
```

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, **`splat`** reports the thread detail as shown in Example 3-51.

Example 3-51 PThread read/write lock (thread detail)

```
          Acquisitions    Miss     Spin Count     Wait Count    Busy   Percent Held of Total Time
ThreadID  Write  Read     Rate    Write  Read    Write  Read    CountCPU  Elapse  Spin    Wait
~~~~~~~~~ ~~~~~~ ~~~~~~~  ~~~~~~~ ~~~~~~ ~~~~~~~ ~~~~~~ ~~~~~~~ ~~~~~~~ ~~~~~~~ ~~~~~~~ ~~~~~~~
        1 0 36883860.000       0       0       0       00.00  99.99    0.00    0.00
```

The columns are defined as follows:

`PThreadID` The *PThread 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 *wait* until the lock came available, differentiated by write versus read.

`Busy Count` The number of times this thread used *try* to acquire the lock, without success (for example calls to simple_lock_try() that returned busy).

`Percent Held of Total Time` contains 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.

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 3-52.

Example 3-52 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
============================================================================
         |                     | Spin / Wait Time ( 18.330873s )
         |  Fail  Spin   Wait  |  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, **splat** reports the thread detail as shown in Example 3-53.

Example 3-53 PThread Condition-Variable Report (thread detail)

```
                          Fail    Spin    Wait     % Total Time
PThreadID   Passes         Rate    Count   Count     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 *passed*.

`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 *wait* until the condition came true.

`Percent Total Time` This field contains the following subfields:

- `Spin` The percentage of *elapsed real time* that this thread spun while testing the condition.
- `Wait` The percentage of *elapsed real time* that this thread spent waiting for the condition to hold.

3.7 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**, 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.

3.7.1 The trace command

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] [-r reglist] [-s] [-A process-id [,process-id]] [-t thread-id [,thread-id]] [-x program-specification | -X program-specification] [-l] [-P trace-propagation][-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 then 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 -Call 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 **smit** or the -J flag, these events are in the tidhk group. Additional event hooks can be read in Appendix B, “Trace hooks” on page 699.

-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 **trcevgrp -l**.

-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/trcmacros.h	Defines trchook and utrchook macros.
/var/adm/ras/trcfile	Contains the default trace log file.

3.7.2 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 3-67 on page 176.

You can use the System Management Interface Tool (**smit**) to run the **trace** daemon. See “Using SMIT to stop and start trace” on page 175 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.

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 want 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`**. 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 3-5 on page 154.

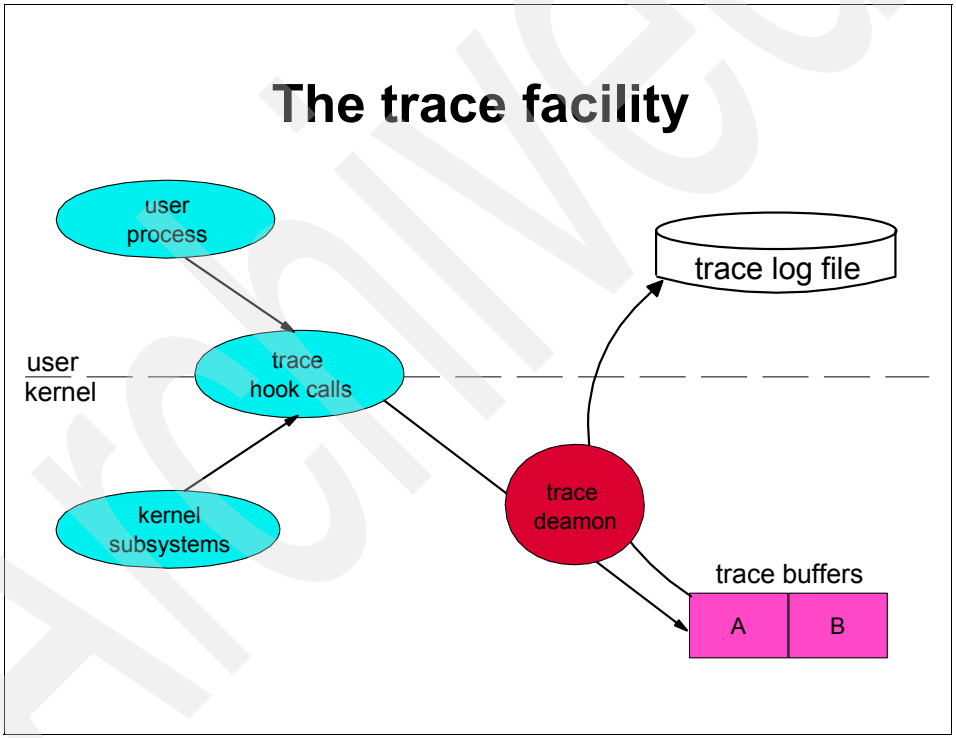

Figure 3-5 The trace facility

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 152), 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 second. 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 152.

3.7.3 How to start and stop trace

There are several ways to stop and start trace. Trace daemon can be started from SMIT, from command line, or by using other data collection programs, based on trace (filemon, netpmon etc.)

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 3-54 will appear.

Example 3-54 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 3-55.

Example 3-55 Using SMIT to start the trace

```
# smitty trace
                                START Trace
```

```
Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                [Entry Fields]
  EVENT GROUPS to trace                         []                      +
  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 3-54 on page 155 to stop the trace. The trace `trace.raw` will reside in the current directory.

3.7.4 Running trace interactively

Example 3-56 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 3-56 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
```

3.7.5 Running trace asynchronously

Example 3-57 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 3-57 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 3-56.

3.7.6 Running trace on an entire system for 10 seconds

Example 3-58 on page 157 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 3-58 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 3-59 shows how to run **trace** asynchronously, tracing the **ls** command and outputting the raw trace file to /tmp/my_trace_log.

Example 3-59 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
```

3.7.7 Tracing a command

The following section shows how to trace commands.

Tracing a command that is not already running on the system

Example 3-59 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 3-58, and use the **trcrpt** command with the -p flag to specify reporting of the specific process.

3.7.8 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 3-60. This increases the total buffered size capacity for collecting trace events.

Example 3-60 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 -aCall -o mylog** command would produce the files mylog, mylog-0, mylog-1, mylog-2, mylog-3, and so forth, one for each CPU.

3.7.9 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 178.

- 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 162.

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 3-61. 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 3-61 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 microsecond times for an indication of which processes are using the most time. This can also be achieved by using the shell script in Example 3-62. The script greps for the microsecond times, and displays trace file lines of the top 20 highest return times. It excludes the trace hook ID 102 (wait).

Example 3-62 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 3-63 shows the output from the script.

Example 3-63 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 3-64.

Example 3-64 Traces for process 29944 (java)

```
# trcrpt -O 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 3-63 on page 160 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 microseconds (refer to Example 3-63 on page 160) in trcrpt.29944. This will display the events for the process as shown in Example 3-65.

Example 3-65 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 3-65 has been split vertically, approximately halfway across the page, and shown separately in the next two examples.

Example 3-66 shows the left-hand side with the times.

Example 3-66 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 3-67. The trace hooks have been left in to enable you to associate the two examples.

Example 3-67 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 3-68).

Example 3-68 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 3-69 on page 163.

Example 3-69 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.

3.7.10 The trcnm command

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.

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.

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.

3.7.11 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 3-70.

Example 3-70 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.

3.7.12 The trcrpt command

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 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 -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:

- 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.

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.

3.7.13 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 3-71 shows how to run **trcrpt** using /var/adm/ras/trcfile as the raw trace file.

Example 3-71 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             []                         +
  IDs of events to INCLUDE in report            []                         +X
  Event Groups to EXCLUDE from report           []                         +
  ID's of events to EXCLUDE from report         []                         +X
  STARTING time                                 []
  ENDING time                                   []
  LOG FILE to create report from                [/var/adm/ras/trcfile]
  FILE NAME for trace report (default is stdout) []
```

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 3-72 shows this procedure.

Example 3-72 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

Fri Apr 16 12:36:57 2004
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)...
```

4

CPU analysis and tuning

This chapter provides detailed information about the following CPU monitoring or tuning tools.

- Monitoring tools
  - lparstat (new command in AIX 5L Version 5.3)
  - mpstat (new command in AIX 5L Version 5.3)
  - procmon (new tool in AIX 5L Version 5.3)
  - topas
  - sar
  - iostat
  - vmstat
  - ps
  - trace
  - curt
  - splat
  - truss
  - gprof,pprof,prof,tprof
  - time,timex
- Tuning tools
  - smtctl (new command in AIX 5L Version 5.3)
  - bindintcpu
  - bindprocessor
  - schedo
  - renice
  - nice

4.1 CPU overview

When investigating a performance problem, we usually start by monitoring the statistics of CPU utilization. It is important continuously observe system performance because, when performing performance problem determination, we need to compare the loaded system data with normal usage data.

Generally, CPU is one of the fastest components of the system and if CPU utilization keeps the CPU 100% busy, this also affects system-wide performance. If you discover that the system keeps the CPU 100% busy, you need to investigate the process which causes this. AIX provides many trace and profiling tools for system and/or processes.

4.1.1 Performance considerations with POWER4-based systems

POWER4™-based server supports Logical Partitioning (LPAR). Each of the partitions on a same system can run a different level of operating system and LPAR-ing has been designed to isolate software running in one partition from the other partitions. Generally, an application is not aware that it is running in a LPAR or not. LPAR is transparent to AIX applications and most AIX performance tools. From the processor point of view, each LPAR needs at least one processor, and it is necessary to assign CPUs in integer numbers.

DLPAR

Using the Dynamic LPAR function, you can change the number of online processors dynamically. Some performance monitoring tools such as **`topas, sar, vmstat, iostat, lparstat, mpstat`** support DLPAR operation. These commands can detect the change of system configuration and report the latest system configuration.

4.1.2 Performance considerations with POWER5-based systems

POWER5 is IBM's second generation of dual core microprocessor chips. POWER5 provides new and improved functions for more granular and flexible partitioning.

From the processor point of view, POWER5 processors contain new technologies, like Micro-Partitioning and simultaneous multi-threading (SMT). AIX 5L Version 5.3 also supports these new technologies.

Micro-Partitioning provides the ability to share a single processor between multiple partitions. These partitions are called shared processor partitions. Of course, POWER5-based systems continue to support partition with dedicated

dedicated processors. These partitions are called dedicated partitions. Dedicated partitions don't share a single physical processor with other partitions.

In a shared-partition environment, the POWER Hypervisor™ schedules and distributes processor entitlement to shared-partitions from a set of physical processors. These physical processor set is called shared processor pool. Processor entitlement is distributed with each turn of the hypervisor's dispatch wheel, and each partition consumes or cedes the given processor entitlement. Figure 4-1 shows a sample of a dedicated partition and a micro-partition on POWER5-based server.

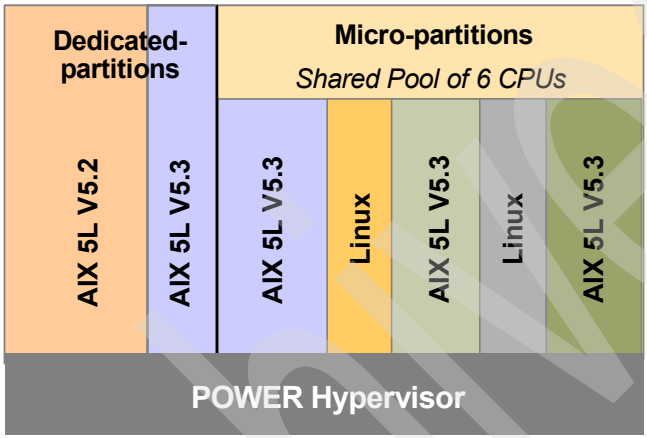

Figure 4-1 LPARs configuration on Power5-based server

In simultaneous multi-threading (SMT), the processor fetches instructions from more than one thread. The basic concept of SMT is that no single process use all processor execution units at the same time. The POWER5 design implements two-way SMT on each of the chip's two processor cores. Thus, each physical processor core is represented by two Virtual processors. Figure 4-2 on page 174 shows a comparison between single threaded and simultaneous multi-threading.

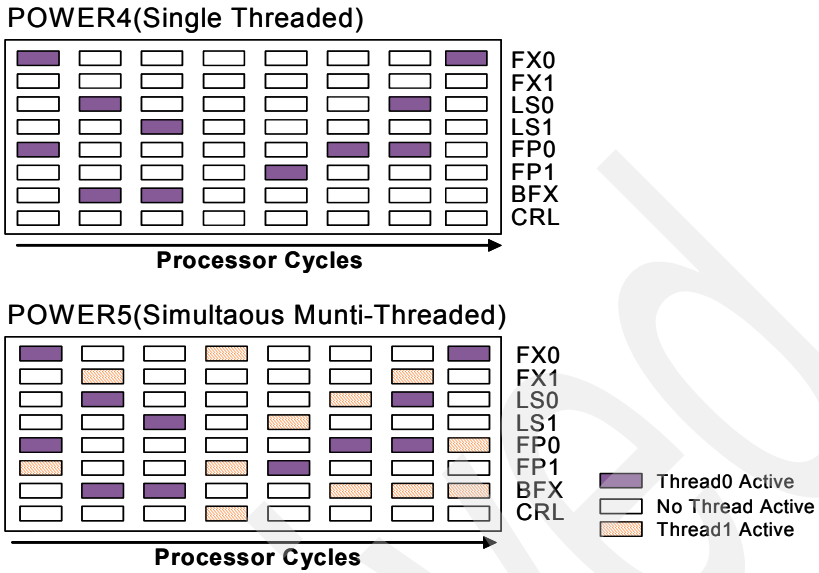

Figure 4-2 Simultaneous Multi-threading

For more information about Micro-Partitioning and SMT, refer to the whitepaper "IBM @server p5 AIX 5L Support for Micro-Partitioning and Simultaneous Multi-threading", at:

http://www-1.ibm.com/servers/aix/whitepapers/aix_support.pdf

4.2 CPU monitoring

This section introduces the most frequently used CPU monitoring commands. The command syntax, along with usage example is presented for clarity.

4.2.1 The lparstat command

The **lparstat** command has been introduced for showing logical partition (LPAR) related information and statistics. The **lparstat** command resides in /usr/bin and is part of the bos.acct package, which is installable from the AIX base installation media.

Syntax

lparstat { [**-i** | **-H** | [**-h**] [**Interval** [**Count**]] }

Flags

-i	Lists detailed information on LPAR configuration
-H	Provides detailed information about Hypervisor statistics
-h	Adds summarized Hypervisor statistics to the default output

Parameters

Interval	specifies the amount of time in seconds between each report
Count	specifies the number of reports generated

Examples

lparstat command has following three modes.

Monitoring mode

The lparstat command with no options will generate a single report containing utilization statistics related to the LPAR since boot time. Example 4-1 on page 176 shows a sample of the utilization statistics report.

The following information is displayed for the utilization statistics.

%user	Shows the percentage of the entitled processing capacity used while executing at the user (or application) level.
%sys	Shows the percentage of the entitled processing capacity used while executing at the system (or kernel) level.
%idle	Shows the percentage of the entitled processing capacity unused while the partition was idle and did not have any outstanding disk I/O request.
%wait	Shows the percentage of the entitled processing capacity unused while the partition was idle and had outstanding disk I/O request(s).

For the dedicated partitions, the entitled processing capacity is the number of physical processors.

The following statistics are displayed only on the shared partition.

physc	Shows the number of physical processors consumed.
%entc	Shows the percentage of the entitled capacity consumed.
lbusy	Shows the percentage of logical processors utilization while executing at the user and system level.

app Shows the available physical processors in the shared pool.

phint Shows the number of phantom (targeted to another shared partition in this pool) interruptions received.

Example 4-1 Displaying the utilization statistics with the lparstat command

```
r33n01:/ # lparstat 1 5

System configuration: type=Shared mode=Uncapped smt=On lcpu=4 mem=7168 ent=2.00

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint
-----  ----  -----  ----- ----- ----- ------   ---  ---- -----
  0.1   0.3    0.0   99.6  0.01   0.5    0.0     -   320     0
  0.0   0.3    0.0   99.7  0.01   0.5    0.0     -   376     0
  0.1   0.7    0.0   99.2  0.02   1.0    2.5     -   462     0
  0.0   0.3    0.0   99.7  0.01   0.4    0.0     -   434     0
  0.1   0.3    0.0   99.6  0.01   0.6    0.0     -   409     0
r33n01:/ #
```

If the **-h** flag is specified, the report will include the following Hypervisor related statistics.

%hypv Shows the percentage of time spent in hypervisor.

hcalls Shows the number of hypervisor calls executed.

Example 4-2 shows a sample of lparstat statistics report using the **-h** flag.

Example 4-2 The lparstat command with -h flag reports Hypervisor statistics

```
r33n01:/ # lparstat -h 1 5

System configuration: type=Shared mode=Uncapped smt=On lcpu=4 mem=7168 ent=2.00

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint  %hypv hcalls
-----  ----  -----  ----- ----- ----- ------   ---  ---- -----  ----- ------
 47.3   0.1    0.0   52.6  1.00  50.2   27.8     -   189     0    2.8    204
 47.3   0.0    0.0   52.7  1.00  50.0   25.0     -   160     0    2.7    166
 47.4   0.0    0.0   52.5  1.00  50.0   25.0     -   169     0    2.6    173
 47.3   0.0    0.0   52.7  1.00  50.0   25.0     -   160     0    2.7    164
 47.3   0.0    0.0   52.7  1.00  50.0   25.0     -   162     0    2.7    166
r33n01:/ #
```

Information mode

The lparstat command with -i flag displays static LPAR configuration.
Example 4-3 on page 177 shows a sample of static LPAR configuration report.

Example 4-3 Displaying the static LPAR configuration report

```
r33n01:/ # lparstat -i
Node Name                                  : r33n01
Partition Name                             : r33n01_aix
Partition Number                           : 1
Type                                       : Shared-SMT
Mode                                       : Uncapped
Entitled Capacity                          : 2.00
Partition Group-ID                         : 32769
Shared Pool ID                             : 0
Online Virtual CPUs                        : 2
Maximum Virtual CPUs                       : 40
Minimum Virtual CPUs                       : 1
Online Memory                              : 7168 MB
Maximum Memory                             : 12288 MB
Minimum Memory                             : 1024 MB
Variable Capacity Weight                   : 128
Minimum Capacity                           : 1.00
Maximum Capacity                           : 4.00
Capacity Increment                         : 0.01
Maximum Dispatch Latency                   : 0
Maximum Physical CPUs in system            : 4
Active Physical CPUs in system             : 4
Active CPUs in Pool                        : -
Unallocated Capacity                       : 0.00
Physical CPU Percentage                    : 100.00%
Unallocated Weight                         : 0
r33n01:/ #
```

Hypervisor mode

The **lparstat** command with the **-H** flag provides detailed Hypervisor information. This option basically displays the statistics for each of the Hypervisor calls. Example 4-4 on page 178 shows a sample of the statistics for each of the Hypervisor calls.

The following information is displayed for Hypervisor statistics:

Number of calls The number of Hypervisor calls made.

%Total Time Spent Percentage of total time spent in this type of call.

%Hypervisor Time Spent Percentage of Hypervisor time spent in this type of call.

Average Call Time(ns) Average call time for this type of call in nano-seconds.

Maximum Call Time(ns) Maximum call time for this type of call in nano-seconds.

Example 4-4 Displaying the detailed information of Hypervisor calls

```
r33n01:/ # lparstat -H 10 2

System configuration: type=Shared mode=Uncapped smt=On lcpu=4 mem=7168 ent=2.00

           Detailed information on Hypervisor Calls

Hypervisor         Number of    %Total Time   %Hypervisor   Avg Call    Max Call
  Call               Calls         Spent      Time Spent    Time(ns)    Time(ns)

remove                    0            0.0           0.0           1         638
read                      0            0.0           0.0           1         285
nclear_mod                0            0.0           0.0           1           0
page_init                 3            0.0           0.0         816        5142
clear_ref                 0            0.0           0.0           1           0
protect                   0            0.0           0.0           1           0
put_tce                   0            0.0           0.0           1         785
xirr                      4            0.0           0.0         841        1521
eoi                       3            0.0           0.0         519         785
ipi                       0            0.0           0.0           1         773
cppr                      0            0.0           0.0           1           0
asr                       0            0.0           0.0           1           0
others                    0            0.0           0.0           1           0
enter                     3            0.0           0.0         256         739
cede                   1798            8.1         100.0      898346    16378277
migrate_dma               0            0.0           0.0           1           0
put_rtce                  0            0.0           0.0           1           0
confer                    0            0.0           0.0           1        2336
prod                     92            0.0           0.0         376         928
get_ppp                   1            0.0           0.0        1310        1840
set_ppp                   0            0.0           0.0           1           0
purr                      0            0.0           0.0           1           0
pic                       1            0.0           0.0         281         407
bulk_remove               0            0.0           0.0           1           0
send_crq                  0            0.0           0.0           1           0
copy_rdma                 0            0.0           0.0           1           0
get_tce                   0            0.0           0.0           1           0
send_logical_lan          0            0.0           0.0           1           0
add_logicl_lan_buf        0            0.0           0.0           1           0
-------------------------------------------------------------------------------
-
remove                    6            0.0           0.0         217         638
read                    109            0.0           0.0          81         285
nclear_mod                0            0.0           0.0           1           0
page_init                 3            0.0           0.0        1641        5142
clear_ref                 0            0.0           0.0           1           0
protect                   0            0.0           0.0           1           0
put_tce                  28            0.0           0.0         292         785
```

```
xirr                      18          0.0          0.0          647         1605
eoi                       17          0.0          0.0          399          785
ipi                        0          0.0          0.0            1          773
cppr                       0          0.0          0.0            1            0
asr                        0          0.0          0.0            1            0
others                     0          0.0          0.0            1            0
enter                      9          0.0          0.0          336          739
cede                    1821          8.1        100.0       885673     16378277
migrate_dma                0          0.0          0.0            1            0
put_rtce                   0          0.0          0.0            1            0
confer                     0          0.0          0.0            1         2336
prod                     100          0.0          0.0          365          928
get_ppp                    1          0.0          0.0         1264         1840
set_ppp                    0          0.0          0.0            1            0
purr                       0          0.0          0.0            1            0
pic                        1          0.0          0.0          382          407
bulk_remove                0          0.0          0.0            1            0
send_crq                   0          0.0          0.0            1            0
copy_rdma                  0          0.0          0.0            1            0
get_tce                    0          0.0          0.0            1            0
send_logical_lan           0          0.0          0.0            1            0
add_logicl_lan_buf         0          0.0          0.0            1            0
--------------------------------------------------------------------------------
-
r33n01:/ #
```

4.2.2 The mpstat command

The **mpstat** command is the new command which collects and displays detailed output on performance statistics for all logical CPUs in the system. The **mpstat** command resides in /usr/bin and is part of the bos.acct fileset, which is installable from the AIX base installation media.

Syntax

mpstat [{ **-d** | **-i** | **-s** | **-a** }] [**-w**] [interval [count]]

flags

-a	Displays all statistics report in wide output mode
-d	Displays detailed affinity and migration statistics for AIX threads and dispatching statistics for logical processors in wide output mode
-i	Displays detailed interrupt statistics in wide output mode
-s	Displays SMT utilization report if SMT is enabled
-w	Turn on wide output mode

Parameters

Interval specifies the amount of time in seconds between each report

Count specifies the number of reports generated

Examples

When the **mpstat** command is invoked, it displays two sections of statistics. The first section displays the system configuration, which is displayed when the command starts, and whenever the system configuration is changed. User can specify the interval time between each report and the number of times of the statistics are reported.

The following system configuration information is displayed in the first section of the command output.

lcpu The number of logical processors.

ent Entitled processing capacity in processor units. This information will be displayed only if the partition type is shared.

The second section displays the utilization statistics for all logical CPUs. The mpstat command also displays a special CPU row with the cpuid “ALL”, which shows the partition-wide utilization. The mpstat command gives the various statistics. It depends on the flag.

Default utilization statistics

If you run the mpstat command without a flag, it only gives a basic statistics. If the partition type is shared, a special CPU row with the cpuid U can be displayed when the entitled processing capacity has not entirely been consumed. Example 4-5 shows a sample of the **mpstat** command without flags.

Example 4-5 Default output of the mpstat command

```
r33n01:/ # mpstat 1 3

System configuration: lcpu=4 ent=2.0

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0    0    0    0  164   83   40    0    1 100   17  0  0  0 100 0.17  8.3  113
  1    0    0    0  102    1    1    1    0 100 3830453 66 34  0  0 0.83 41.6    0
  2    0    0    0   15   34   17    1    0 100   54 68 19  0 13 0.00  0.1   46
  3    0    0    0   10   16    9    0    0 100    0  0 22  0 78 0.00  0.0   46
  U    -    -    -    -    -    -    -    - 100    -  -  -  0 50 1.00 49.9    -
ALL    0    0    0  291  134   67    2    1 100 3830524 28 14  0 58 1.00 50.1  102
------------------------------------------------------------------------------
  0    0    0    0  167  108   52    0    0 100   36  0  0  0 100 0.17  8.4  114
```

```
  1    0    0    0  100    0    0    1    0   - 3831531 66 34  0  0 0.83 41.6    0
  2    0    0    0   25   10    5    1    0 100    24 22 44  0 34 0.00  0.0   36
  3    0    0    0   10   15    8    0    0 100     0  0 50  0 50 0.00  0.0   36
  U    -    -    -    -    -    -    -    -   -     -  -  -  0 50 1.00 50.0    -
ALL    0    0    0  302  133   65    2    0 100 3831591 27 14  0 58 1.00 50.0   93
-------------------------------------------------------------------------------
  0    0    0    0  163   36   18    0    0 100     8  0  0  0 100 0.17  8.3  113
  1    0    0    0  102    1    1    1    0 100 3830744 66 34  0  0 0.83 41.7    0
  2    0    0    0   25   90   45    1    1 100    24 15 42  0 43 0.00  0.0   73
  3    0    0    0   11   17   11    0    0 100     2  1 40  0 59 0.00  0.0   73
  U    -    -    -    -    -    -    -    -   -     -  -  -  0 50 1.00 50.0    -
ALL    0    0    0  301  144   75    2    1 100 3830778 27 14  0 58 1.00 50.0  129
r33n01:/ #
```

The **mpstat** shows following statistics in default mode.

- Logical processor ID (cpu)
- Minor and major page faults (min, maj)
- Total number of inter-processor calls (mpc)
- Total number of interrupts (int)
- Total number of voluntary and involuntary context switches (cs, ics)
- Run rue size (rq)
- Total number of thread migrations (mig)
- Logical processor affinity (lpa)
- Total number of system calls (sysc)
- Processor usage statistics (us, sy, wa, id)
- Fraction of processor consumed (pc)
- The percentage of entitlement consumed (%ec)
- Total number of logical context switches (lcs)

Note: pc is displayed only in a shared partition, or when simultaneous multi-threading (SMT) is on. The **%ec** and **lcs** are displayed only in shared partition.

Dispatch and affinity statistics

If you want to see the detailed affinity, migration and dispatch metrics, you can use the mpstat command with the **-d** option as in Example 4-6.

Example 4-6 Displaying the affinity, migration and dispatch metrics

```
r33n01:/ # mpstat -d 1 3

System configuration: lcpu=4 ent=2.0

cpu     cs    ics  bound     rq   push S3pull  S3grd  S0rd  S1rd  S2rd  S3rd  S4rd  S5rd   ilcs
vlcs
```

```
  0    109     54      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
114
  1      1      1      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  2     10      5      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  3     14      7      0      0      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
17
ALL    134     67      3      3      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
65
------------------------------------------------------------------------------------------------
-------
  0    109     54      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
114
  1      1      1      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  2     12      6      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  3     14      7      0      0      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
18
ALL    136     68      3      3      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
66
------------------------------------------------------------------------------------------------
-------
  0    109     53      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
113
  1      1      1      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  2     10      5      1      1      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
0
  3     12      6      0      0      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
16
ALL    132     65      3      3      0      0      0 100.0   0.0   0.0   0.0   0.0   0.0      0
64
r33n01:/ #
```

Interrupt statistics

If you want to see the detailed interrupts statistics, you can use the mpstat command with **-i** option as in Example 4-7.

Example 4-7 Displaying the interrupt statistics

```
r33n01:/ # mpstat -i 1 3

System configuration: lcpu=4 ent=2.0

cpu   mpcs   mpcr    dev   soft    dec     ph
  0      0      0      1     56    115      0
```

```
  1      0      0      1      1    100      0
  2      0      0      2      1     14      0
  3      0      0      0      0     10      0
ALL      0      0      4     58    239      0
------------------------------------------------------------------------------
-
  0      0      0      1     52    114      0
  1      0      0      1      1    100      0
  2      0      0      0      6     13      0
  3      0      0      1      1     10      0
ALL      0      0      3     60    237      0
------------------------------------------------------------------------------
-
  0      0      0      0     51    113      0
  1      0      0      2      2    100      0
  2      0      0      0      6     16      0
  3      0      0      1      1     10      0
ALL      0      0      3     60    239      0
r33n01:/ #
```

SMT utilization statistics

To see the simultaneous multi-threading threads utilization, you can use the **mpstat** command with **-s** option. If **mpstat** is running in a dedicated partition and Simultaneous Multi-Threading is enabled, then only the thread (logical CPU) utilization is displayed. Example 4-8 shows a sample of the **mpstat** command with SMT enable mode on a shared processor partition.

Example 4-8 The mpstat command shows thread utilization with SMT enable

```
r33n01:/ # mpstat -s 1 3

System configuration: lcpu=4 ent=2.0

     Proc0           Proc2
      0.30%         100.00%
 cpu0    cpu1    cpu2    cpu3
  0.23%   0.08%  84.17%  15.83%
------------------------------------------------------------------------------
-
     Proc0           Proc2
      0.17%         100.00%
 cpu0    cpu1    cpu2    cpu3
  0.12%   0.05%  84.18%  15.82%
------------------------------------------------------------------------------
-
     Proc0           Proc2
      0.22%         100.00%
 cpu0    cpu1    cpu2    cpu3
```

```
  0.16%   0.05%  84.18%  15.82%
r33n01:/ #
```

4.2.3 The procmon tool

The procmon tool is the new command which shows performance statistics or the sorted list of processes table, and can also carry out actions on the processes. The procmon tool runs on the Performance Workbench platform. The Performance Workbench is an Eclipse-based tool and it has a graphical user interface to monitor the system activity.

The **perfwb** command is used to start the Performance Workbench. After perfwb is started, the procmon tool runs as a plug-in in the Performance Workbench. The perfwb command resides in /usr/bin and is part of the bos.perf.gtools.perfwb fileset, which is installable from the AIX base installation media. The Performance Workbench is included in bos.perf.gtools.perfwb fileset. The procmon tool plug-in is included in bos.perf.gtools.procmon fileset.

Procmon tool provides following functions.

- Displaying performance statistics
- Displaying sorted process lists
  - Columns and sorting key can be configured
  - Filtering rule can be defined
- Performing actions o processes
  - kill, renice, showing detailed information of processes
- Exporting procmon data to file

Syntax

perfwb

Example

Procmon perspective

Procmon provides two main tables, the performance statistic view and the processes table. These views are provided in the procmon perspective. To display the procmon perspective, you can select Window → Open Perspective → Procmon.

Displaying the performance statistics

If you click the Partition performance tab, it shows the performance statistics, as in Figure 4-3 on page 185.

CPU consumption displays the average of CPU utilization percentage. Memory consumption displays the information about the usage of memory and paging

space. This view also provides the partition state information. It includes the number of CPUs, active kernel, the number of processes, and the length of time the system has been started.

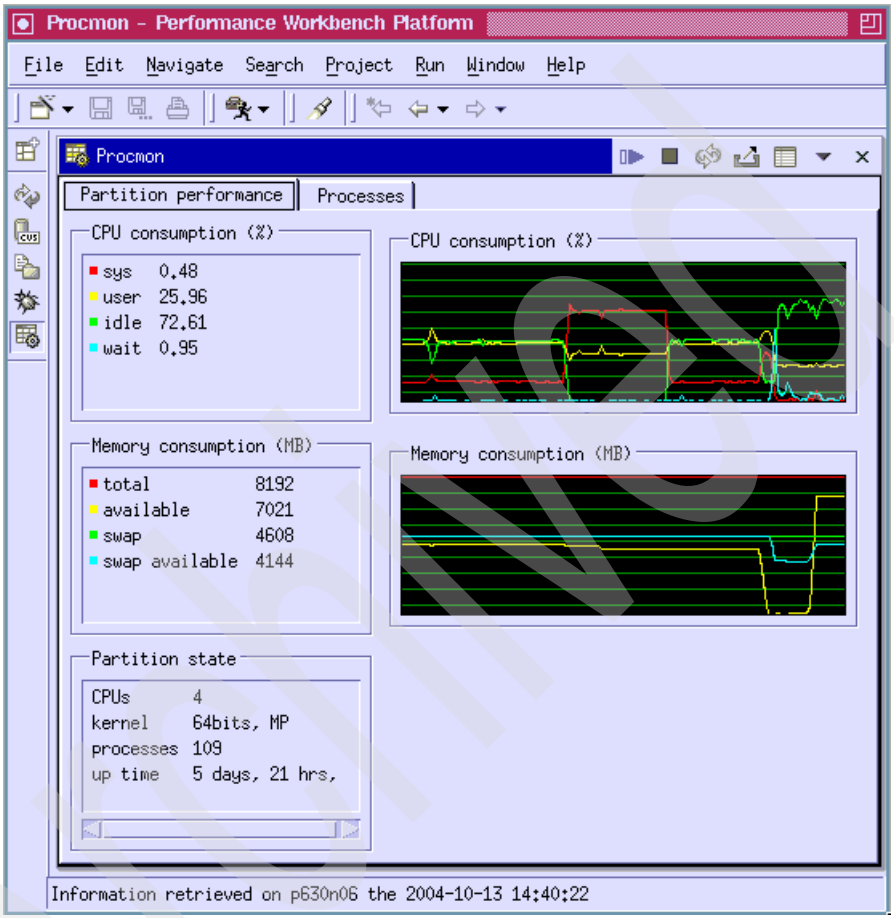

Figure 4-3 Displaying the performance statistics

Displaying the process table

If you want to see the current status of active processes, you can click Processes tab as in Figure 4-4 on page 186.

This will display a sorted list of processes running on the machine. By default, each line contains process ID (PID), CPU usage, memory usage, effective user name, and command name. You can customize these columns using procmon preference as in Figure 4-13 on page 195.

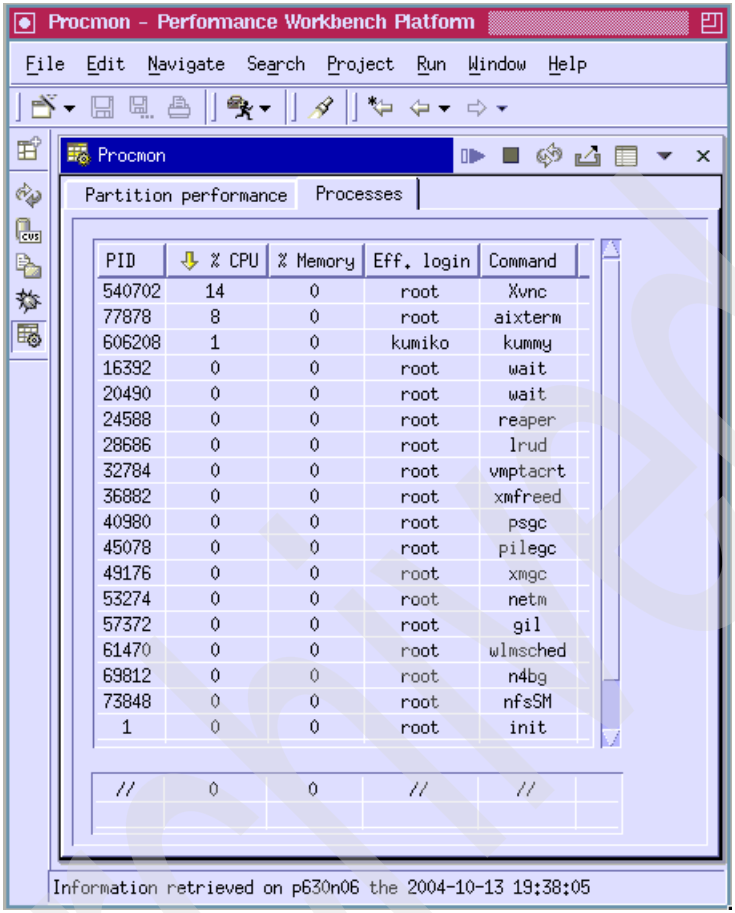

Figure 4-4 Displaying the process table

Performing an action

You can perform some commands to the processes from the processes tab. If you want to perform commands on processes, select the desired process and click the right mouse button to display the pop-up menu. This menu includes following two types of action.

- Detailed information
- Modify processes

Detailed information

This menu is used to display the thread or process information. To display this information, the **`svmon`** and **`proctools`** commands are used. Figure 4-5 on page 187 shows detailed information menu. You can customize the default option

of the **svmon** and the **proctools** using the preferences menu, as in Figure 4-11 on page 192 and Figure 4-12 on page 193.

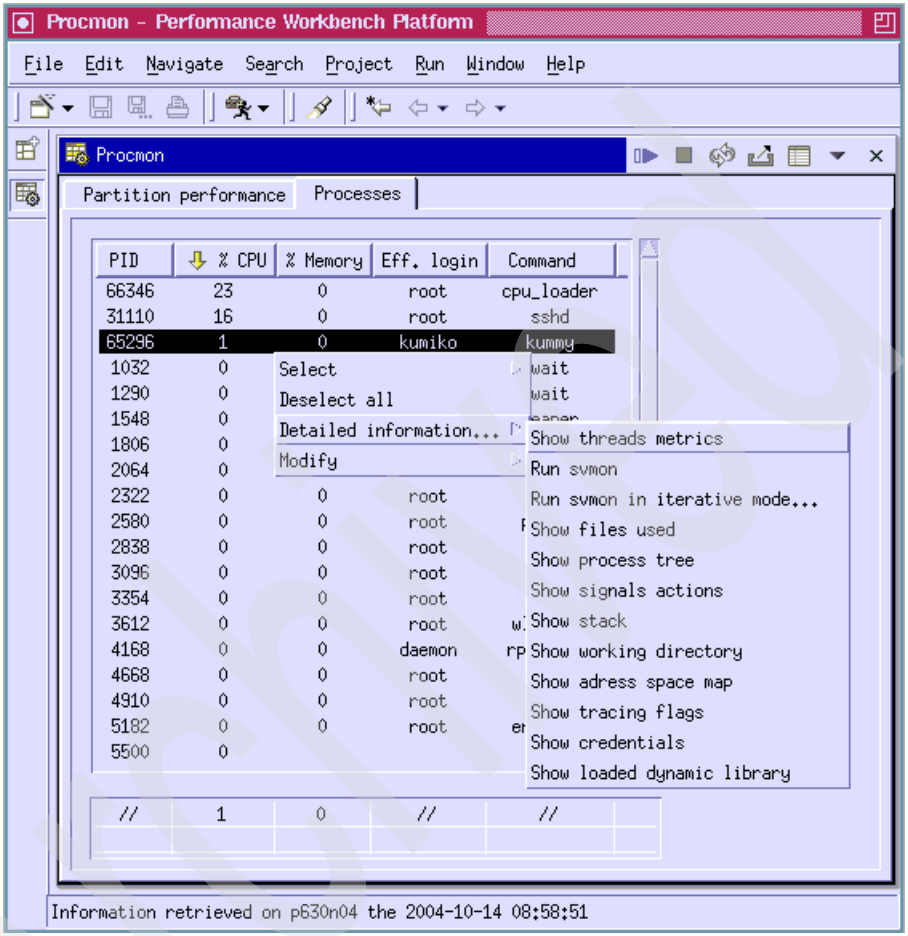

Figure 4-5 Detailed information menu

Show thread metrics	It shows detailed thread information.
Run svmon	It calls the svmon command.
Run svmon in iterative mode...	It calls the svmon -i command. a new panel opens to specify interval and the number of iterations.
Show files used	It calls the procfiles command
Show process tree	It calls the proctree command
Show signals actions	It calls the procsig command

Show stack	It calls the procstack command
Show working directory	It calls the procwdx command
Show address space map	It calls the procmap command
Show tracing flags	It calls the procflags command
Show credentials	It calls the proccred command
Show loaded dynamic library	It calls the procldd command

Figure 4-6 shows an example of “Show loaded dynamic library” on the process. If you want to save the result of this command, you can use save button. The information is saved in ASCII format.

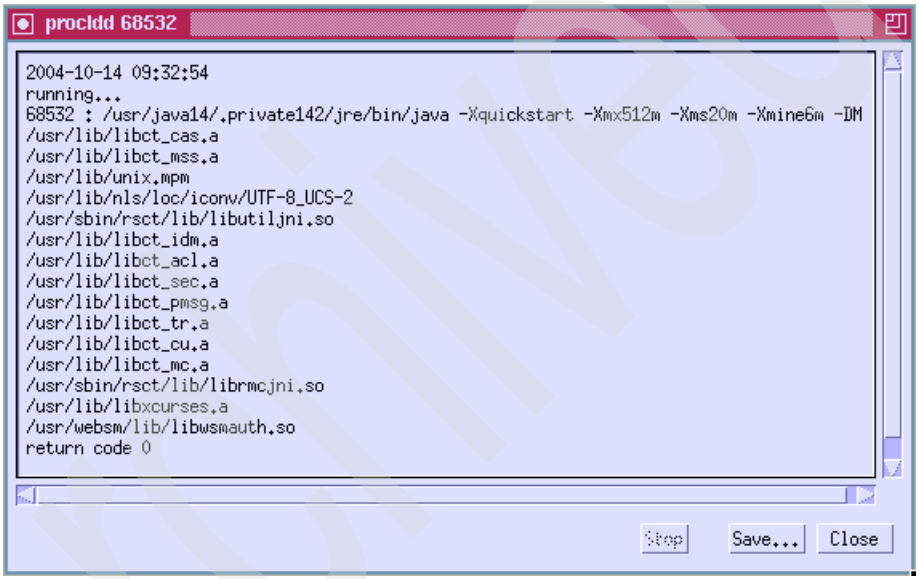

Figure 4-6 Perform “Show loaded dynamic library”

Modifying processes

This menu is used to perform operations on selected processes (kill, renice commands). Figure 4-7 on page 189 shows the process modification menu.

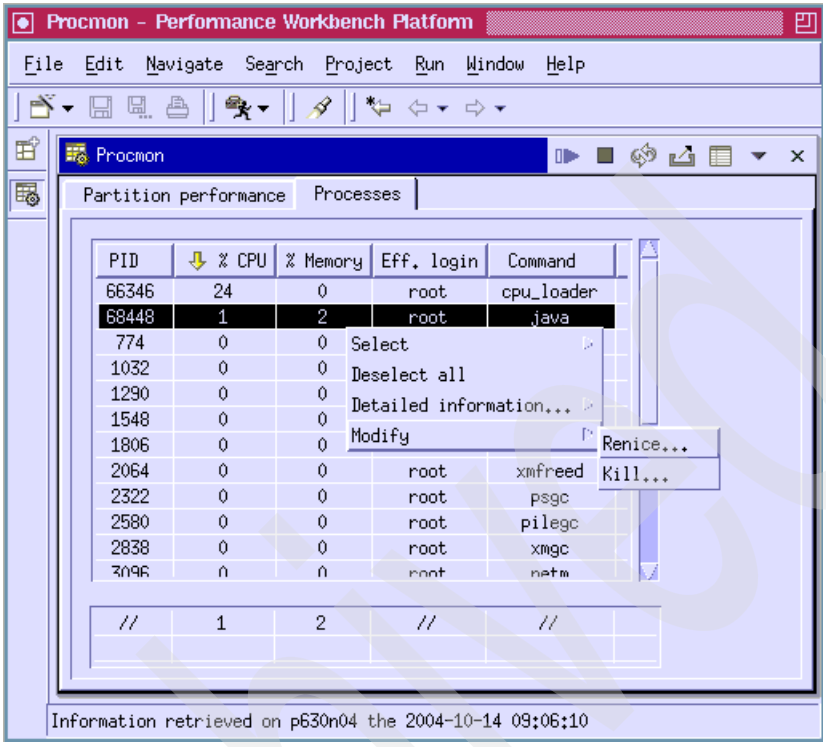

Figure 4-7 Displaying the modify menu

When you select kill menu, a new panel opens to specify the signal number for the kill command, as shown in Figure 4-8.

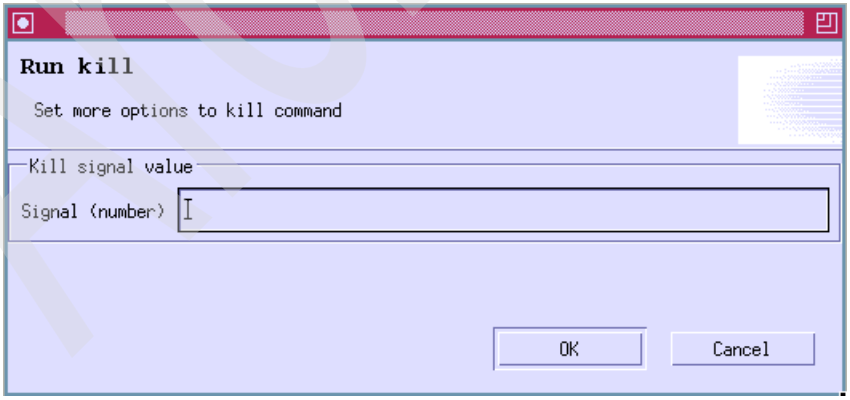

Figure 4-8 Specifies the signal number for the kill command

When you select renice menu, a new panel opens to specify the number to add to the nice value for renice command, as in Figure 4-9.

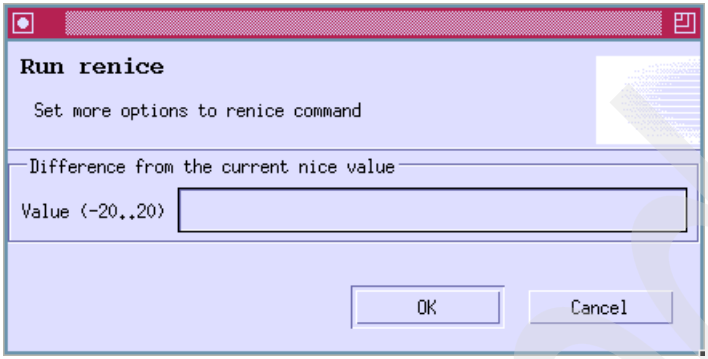

Figure 4-9 Specifies the number to add the nice value for the renice command

Configuring procmon

Procmon has configured with some default value to use. And you can change this configuration in the Window → Preference dialog. Procmon provides the following options:

- Configuring the working directory
- Configuring the Proctools
- Configuring svmon command
- Configuring the process table

Configuring the working directory

By default, Procmon uses the $HOME/workspace directory as procmon working directory. If you want to change the working directory, select Window → Preferences, and then select Procmon dialog. Figure 4-10 on page 191 shows the Preference dialog for setting the procmon tool working directory.

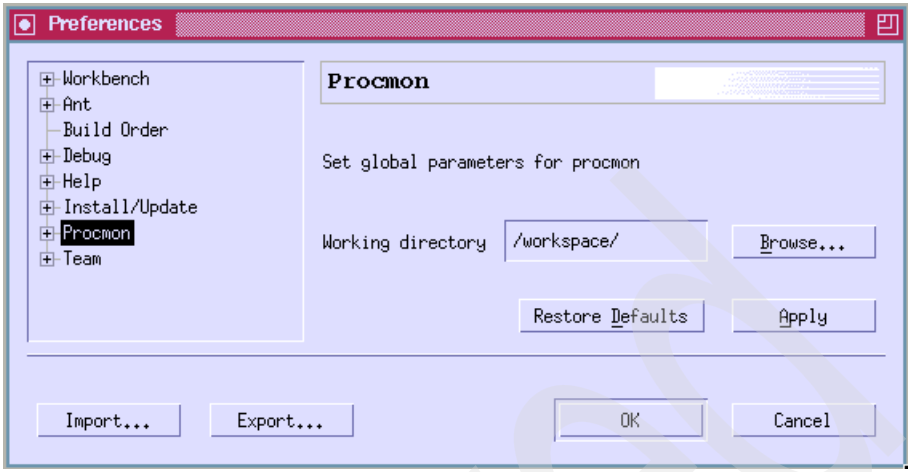

Figure 4-10 Configuring the working directory

Configuring the proctools

Some proctools commands can be executed on the processes selected from the process table. Proctools dialog is used to set the default option for proctools command. If you want to customize Proctools option, select Window → Preference, and then select Procmon → Commands → Proctools.

The following two options are available:

- Forces to take control of the target process even if another process has control. This option is supported by following proctools command as -F option.
  - procfiles
  - procstack
  - procwdx
  - procmap
  - pocldd
- Prints the name of the files referred to by file descriptors. This option is supported by the **`procfile`** command as -n option.

These options are used with proctools commands supporting these options. Figure 4-11 on page 192 shows preference panel for proctools.

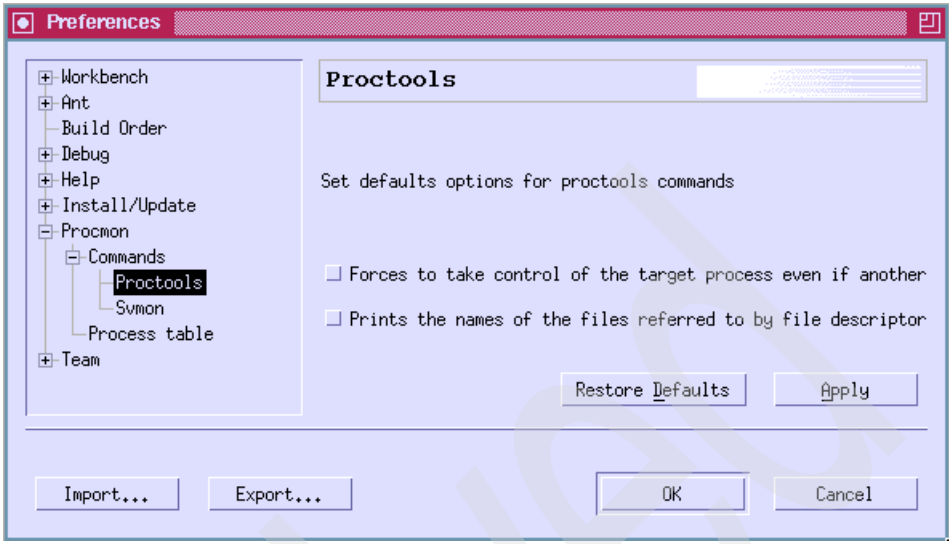

Figure 4-11 Configuring the proctools option

Configuring the svmon command

The **svmon** command can run on the processes selected from the process table. By default, some options are specified to this command, and these options can be customized. If you want to change the default option, select Windows® → Preference, and then select Procmon → Commands → Svmon. Figure 4-12 on page 193 shows Preference panel for the svmon command.

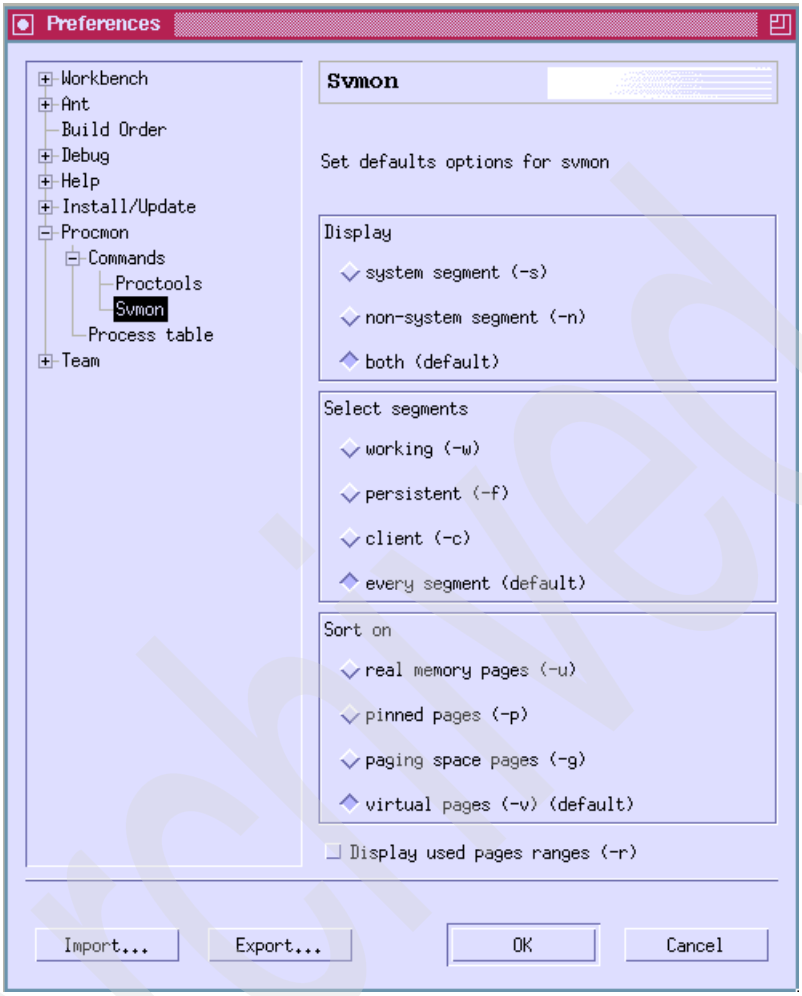

Figure 4-12 Configuring the svmon command

Following three groups of options are available.

- **Display**

system segments	Specifies only system segments are to be included in the statistics.
non-system segments	Specifies only non-system segments are to be included in the statistics.
both	Specifies all segments are to be included in the statistics.

- **Select segment**

working	Specifies only working segments are to be included in the statistics.
persistent	Specifies only persistent segments are to be included in the statistics.
client	Specifies only client segments are to be included in the statistics.
every segment	Specifies all segments are to be included in the statistics.

- **Sort on**

real memory pages	Specifies the information to be displayed is sorted in decreasing order by the total number of pages in real memory.
pinned pages	Specifies the information to be displayed is sorted in decreasing order by the total number of pages pinned.
paging space pages	Specifies the information to be displayed is sorted in decreasing order by the total number of pages reserved or used on paging space.
virtual pages	Specifies the information to be displayed is sorted in decreasing order by the total number of pages in virtual space.

You can also enable to display the ranges within the segment pages which have been allocated.

Configuring the process table

If you want to customize the process table, select Windows → Preference, and then select Procmon → Process table. This is used to specify the way process information are retrieved and displayed. Figure 4-13 on page 195 shows preference panel for process table.

Figure 4-13 Configuring the process table

The properties tab is used to customize following options:

- The number of processes displayed
- Refresh interval in second

- Enable automatic refresh of the process table
- Enable interrupt automatic refresh when starting a command on a PID

The displayed metrics is used to modify the columns displayed on process table. If you want to display an additional column, you can select the column name on the “Metrics available” field and add to the “Metrics displayed” field. If you remove the column name from the “Metrics displayed” field, it isn't displayed process table.

You can also define the default sort key of the process table. If you want to change the default sort key, select the column name from the Metrics displayed field and push “Set key” button. The sorting key can be changed using the process table panel too.

Other functions

Procmon provides additional handy functions. Filter processes and exporting procmon data are some of them.

Filter processes

Using the filter processes menu, you can define the filtering rule to the processes. This is used to display only processes you want to see in the processes table. If you want to create a new filter rule, select Filters... from the procmon menu. Figure 4-14 shows an example of defining the filter rule to display a process which has process ID 15748.

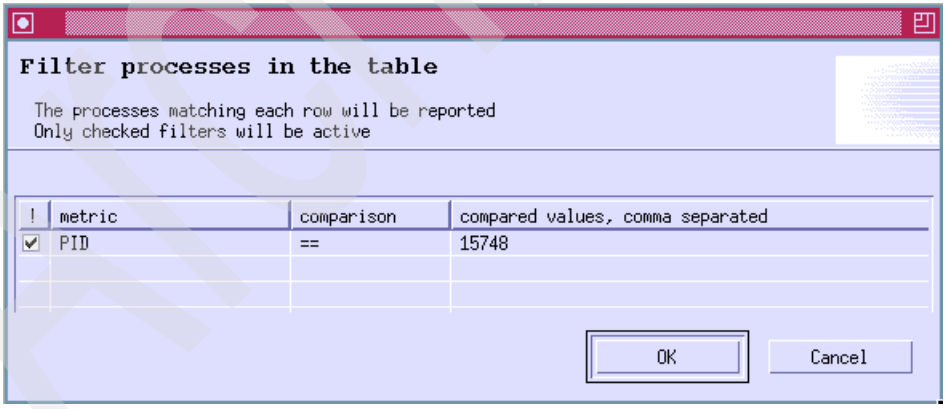

Figure 4-14 Defining the filter rule

Exporting procmon data

Procmon provides a way to export procmon data for external use. If you want to export the data, click the “Exports procmon reports to file” button in the procmon view. A new dialog opens to setup the export configuration, as shown in

Figure 4-15. Using this dialog, you can select the format of the exported data (xml, csv, html) and the data to export (statistic line, processes table, summation table).

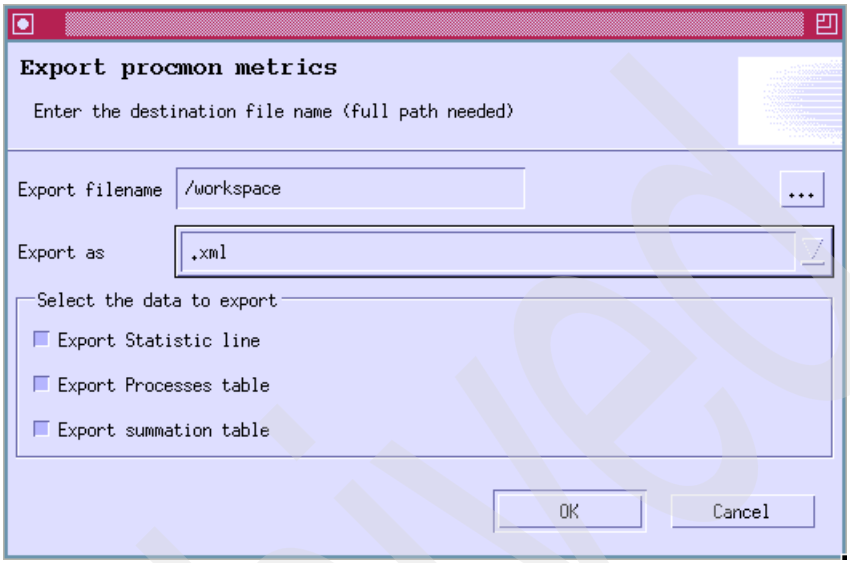

Figure 4-15 Export procmon data

4.2.4 The topas command

The **topas** command is used to display statistics about the activity on the local system. The **topas** command reports the various kinds of statistics, such as CPU utilization, CPU events and queues, process lists, memory and paging statistics, disk and network performance, and NFS statistics. The **topas** command resides in /usr/bin and is part of the bos.perf.tools fileset, which is installable from the AIX base installation media.

Syntax

topas [-d number_of_monitored_hot_disks] [-h] [-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][-U
username_owned_processes] [**-L l-P** | -W] [-m]

flags

-i Specifies the monitoring interval in seconds. The default is two seconds.

-L Displays the logical partition display.

Examples

Default output

Starting with AIX 5L Version 5.3, if the topas command runs on a shared partition, following two new values are reported for the CPU utilization. If the topas command runs on a dedicated partition, these values are not displayed.

Physc Number of physical processors granted to the partition

%Entc Percentage of Entitled Capacity granted to the partition

Example 4-9 shows the standard topas command and its output. It runs on a shared partition. If you run the topas command without flags, the output is refreshed every two seconds.

The topas command shows following information in default mode.

- System hostname
- Current date
- Refresh interval
- CPU utilization
- CPU events and queues
- Process lists
- Memory and paging statistics
- Disk and network performance
- WLM performance (displayed only when WLM is used)
- NFS statistics

Example 4-9 The topas command

```
Topas Monitor for host:    r33n05                EVENTS/QUEUES    FILE/TTY
Tue Oct 19 19:33:48 2004   Interval:  2          Cswitch     139  Readch    11185
                                                 Syscall   13777  Writech   53.6M
Kernel    0.5   |#                    |         Reads         3  Rawin         0
User     46.9   |##############       |         Writes    13710  Ttyout      115
Wait      0.0   |                     |         Forks         0  Igets         0
Idle     52.6   |###############      |         Execs         0  Namei         1
Physc =  1.00                  %Entc=  50.0     Runqueue    1.0  Dirblk        0
                                                 Waitqueue   0.0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out
en0       0.2     1.0     1.0     0.0     0.3   PAGING           MEMORY
lo0       0.1     2.0     2.0     0.1     0.1   Faults        0  Real,MB    6911
                                                 Steals        0  % Comp     11.8
```

```
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp   1.5
hdisk1    0.0      0.0     0.0     0.0     0.0  PgspOut       0  % Client    1.7
hdisk2    0.0      0.0     0.0     0.0     0.0  PageIn        0
hdisk0    0.0      0.0     0.0     0.0     0.0  PageOut       0  PAGING SPACE
cd0       0.0      0.0     0.0     0.0     0.0  Sios          0  Size,MB     512
                                                                 % Used      1.1
Name            PID  CPU%  PgSp Owner           NFS (calls/sec)  % Free     98.8
memtest      659600  25.0   1.3 root            ServerV2       0
topas        610492   0.0   1.5 root            ClientV2       0   Press:
pilegc       360624   0.0   0.1 root            ServerV3       0   "h" for help
IBM.CSMAg    532488   0.0   2.8 root            ClientV3       0   "q" to quit
getty        639088   0.0   0.4 root
gil          372918   0.0   0.1 root
```

The new **-L** flag has been added to the **topas** command to display logical partition. In this mode, the result of **topas** is similar to the **mpstat** command. Example 4-10 shows a sample of the **topas** command with **-L** flag.

Example 4-10 The topas command with -L flag

```
Interval:    2    Logical Partition: r33n05          Tue Oct 19 19:35:58 2004
Psize:       -                Shared SMT  ON          Online Memory:   6912.0
Ent: 2.00                      Mode: UnCapped         Online Logical CPUs:  4
Partition CPU Utilization                             Online Virtual CPUs:  2
%usr %sys %wait %idle physc %entc %lbusy   app   vcsw phint %hypv   hcalls
  47    0     0    53   1.0  50.00 25.00     -    356     0   0.0        0
--------------------------------------------------------------------------------
LCPU  minpf majpf  intr   csw icsw runq lpa scalls usr sys _wt idl   pc   lcsw
Cpu0      0     0    42    23   12    0 100     36  30  47   0  23 0.00     61
Cpu1      0     0    22     3    2    0 100      6   3  28   0  69 0.00     61
Cpu2      0     0   328   250  125    1 100     39   0   1   0  99 0.05    234
Cpu3      0     0   200     2    2    1 100  27428  99   1   0   0 0.95      0
```

Topas subcommands

While **topas** is running, it accepts one-character subcommands. Using these subcommands, you can change the displayed metrics. Following characters are some of useful subcommands.

a Always return to the default topas screen.

c Toggles the CPU statistics subsection between the cumulative report, off, and statistics of the per-processors.

p Toggles the active processes subsection on and off.

P Toggles the active processes with the full-screen mode on and off. This mode provides more detailed information about processes running on the system than the process subsection of the default display. This is the sam as the -P flag from the topas command line.

L Toggles the logical partition statistics on and off. This mod provides current LPAR configuration (CPU, memory), and statistics of the each logical CPUs. This display reports similar data to what is provided to mpstat and lparstat. This is the sam as the -L flag from the topas command line.

q Quit the topas command.

Arrow or Tab keys

Changes the sort key. Subsections from the default display such as the CPU, Network, Disk, WLM, and the full-screen WLM and process are sorted and displayed from highest to lowest order. The cursor over a column indicates the sort key. The cursor can be moved by using the Tab key or the arrow keys.

Example 4-11 shows the process lists with full screen mode. It's a sample output using the **P** subcommand.

Example 4-11 Displaying the process lists using the P subcommand

```
Topas Monitor for host:    r33n05      Interval:   2    Tue Oct 19 19:37:42 2004

                                   DATA  TEXT  PAGE              PGFAULTS
USER       PID    PPID PRI NI    RES   RES SPACE    TIME CPU%  I/O  OTH COMMAND
root    659600  655434 125 24    329     2   329   18:30 25.0    0    0 memtest
root    610492  487492  58 41    384    26   384    0:00  0.0    0    0 topas
root    532488  651328  60 20    703   162   716   57:51  0.0    0    0 IBM.CSMAg
root    639088       1  60 20    106    17   106    2:59  0.0    0    0 getty
root    372918       0  37 41     29     0    29    2:22  0.0    0    0 gil
root    389354       1  60 20    130     2   130    1:50  0.0    0    0 syncd
root    622642  651328  60 20    141    15   141    1:46  0.0    0    0 muxatmd
root    364722       0  60 41     12     0    12    0:24  0.0    0    0 xmgc
root    581668       1  60 20     50     0    50    0:24  0.0    0    0 rpc.lockd
root    593968  651328  60 20    540   141   541    0:08  0.0    0    0 rmcd
root    630858  651328  60 20    183    43   183    0:06  0.0    0    0 snmpmibd6
root    368820       0  36 41     12     0    12    0:02  0.0    0    0 netm
root         1       0  60 20    156    10   156    0:02  0.0    0    0 init
root    422132       1  60 20     12     0    12    0:01  0.0    0    0 rgsr
root    405724       0  60 20     12     0    12    0:00  0.0    0    0 lvmbb
root    409812       0  38 41     81     0    81    0:00  0.0    0    0 j2pg
root    413742       1  60 20    164    22   164    0:00  0.0    0    0 errdemon
root    340134       0  16 41     14     0    14    0:00  0.0    0    0 lrud
root    426200       0  17 20     24     0    24    0:00  0.0    0    0 dog
root    430318       1  39 41     12     0    12    0:00  0.0    0    0 aioserver
```

useful combinations

- `topas -i 1`
- `topas -L`
- `topas -P`

4.2.5 The sar command

The **sar** (System Activity Report) command is used to collect statistics report about CPU, I/O, and other system activities. The sar command shows statistics in two ways, show real time data or show previously data. The sar command resides in /usr/sbin and is part of the bos.acct fileset, which is installable from the AIX base installation media.

syntax

/usr/sbin/sar [{ -A | [-a] [-b] [-c] [-d][-k] [-m] [-q] [-r] [-u] [-v] [-w] [-y] }] [**-P ProcessorIdentifier, ... | ALL**] [-ehh [:mm [:ss]]] [-X File] [**-f File**] [-i Seconds] [**-o File**] [-s hh [:mm [:ss]]] [**Interval** [**Number**]]

flags

-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.

-o File Saves the statistics data in the file in binary form. Each statistics data are in a separate record and each record contains a tag identifying the time of the reading. You can extract records from this file using the sar command with -f flag.

-f File Extracts records from the specified File (created by -o File flag).

Parameters

Interval Specifies the amount of time in seconds between each report

Count Specifies the number of reports generated

Example

When the sar command is invoked, it displays several sections of information and statistics. The first section displays the node information, which include the OS version, machine ID, and invoked date.

The second section displays the system configuration, which is displayed when the command starts, and whenever there is a change in the system configuration. The following information is displayed in the second section of the command output.

lcpu Number of logical processors.

ent Entitled processing capacity in processor units. This information will be displayed only on shared partition.

The third section displays the utilization statistics.The sar command gives the various statistics. It depends on the flags.

Monitoring current CPU statistics

The sar command without a flag or with -u flag reports CPU utilization statistics. This statistics displays following values.

%usr Reports the percentage of time the CPU(s) spent in execution at the user (or application) level. It is equivalent to the us column reported by vmstat.

%sys Reports the percentage of time the CPU(s) spent in execution at the system (or kernel) level. It is equivalent to the sy column reported by vmstat.

%wio Reports the percentage of time the CPU(s) were idle during which the system had outstanding disk/NFS I/O request(s). It is equivalent to the wa column reported by vmstat.

%idle Reports the percentage of time the CPU(s) were idle with no outstanding disk I/O requests. It is equivalent to the id column reported by vmstat.

physc Reports the number of physical processors consumed. This will be reported only if the partition is running with shared processors or simultaneous multi-threading enabled. It is equivalent to the pc column reported by vmstat.

%entc Reports the percentage of entitled capacity consumed. This will be reported only if the partition is running with shared processors. It is equivalent to the ec column reported by vmstat.

Beginning with AIX 5L Version 5.3, the **sar** command reports utilization metrics physc and %entc for shared partitioning and simultaneous multi-threading (SMT) environments. The physc field indicates the number of physical processors consumed by the partition (in case of system wide utilization) or each logical CPU (if the -P flag is specified). The %entc field indicates the percentage of the allocated entitled capacity (in case of system wide utilization) or granted entitled capacity (if the -P flag is specified).

If you specify the interval and number, the sar command reports current CPU utilization statistics, as in Example 4-12.

Example 4-12 The sar command without option

```
r33n01:/ # sar 1 5

AIX r33n01 3 5 00C3E3CC4C00    10/15/04

System configuration: lcpu=4 ent=2.00

18:00:35    %usr    %sys    %wio   %idle   physc   %entc
18:00:36      28      14       0      58    1.00    50.2
18:00:37      28      14       0      58    1.00    50.1
18:00:38      28      14       0      58    1.00    50.1
18:00:39      28      14       0      58    1.00    50.2
18:00:40      28      14       0      58    1.00    50.1

Average       28      14       0      58    1.00    50.2
r33n01:/ #
```

You can monitor per-processor statistics using the **sar** command with -P flag. When using the -P flag, CPU number or “ALL” parameter is required. Example 4-13 shows a sample of the sar command with the -P flag. The last line of each time stamp shows the average CPU utilization for all of the displayed CPUs. It is denoted by a line with dash(-). The last stanza of the output shows the average utilization for each CPU for the duration of the monitoring.

When the partition runs in capped mode, the partition cannot get more capacity than it is allocated. In uncapped mode, the partition can get more capacity than it is actually allocated. This is called granted entitled capacity. If the -P flag is specified and there is unused capacity, sar prints the unused capacity as separate CPU with cpu id “U”.

Example 4-13 The sar command with -P flag

```
r33n01:/ # sar -P ALL 1 3

AIX r33n01 3 5 00C3E3CC4C00    10/15/04

System configuration: lcpu=4 ent=2.00

18:10:57 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:10:58  0       22      65       0      13    0.00     0.2
          1        0       4       0      96    0.00     0.1
          2       66      34       0       0    0.84    42.0
          3        0       0       0     100    0.16     8.0
          U        -       -       0      50    1.00    49.8
```

```
           -       28      14       0      58     1.00     50.2
18:10:59   0       21      65       0      15     0.00      0.2
           1        0       6       0      94     0.00      0.0
           2       66      34       0       0     0.84     42.0
           3        0       0       0     100     0.16      8.0
           U        -       -       0      50     1.00     49.8
           -       28      14       0      58     1.00     50.2
18:11:00   0        5      75       0      20     0.00      0.1
           1        0       7       0      93     0.00      0.0
           2       66      34       0       0     0.84     42.0
           3        0       0       0     100     0.16      8.0
           U        -       -       0      50     1.00     49.8
           -       28      14       0      58     1.00     50.2

Average    0       17      68       0      15     0.00      0.1
           1        0       5       0      95     0.00      0.0
           2       66      34       0       0     0.84     42.0
           3        0       0       0     100     0.16      8.0
           U        -       -       0      50     1.00     49.8
           -       28      14       0      58     1.00     50.2
r33n01:/ #
```

Extracts records from the File

If you specify the filename with **-f** flag, the **sar** command extracts this file and report to standard output, as show in Example 4-14. If you don't specify a file name, the default standard system activity daily data file is used. The default system activity daily data file name is /var/adm/sa/sadd. The "dd" parameter indicates the current day.

Note: The sar command calls a process named *sadc* to access system data. Two shell scripts (/usr/lib/sa/sa1 and /usr/lib/sa/sa2) are structured to be run by the **cron** command, and provide daily statistics and reports. Sample stanzas are included in the /var/spool/cron/crontabs/adm crontab file to collect the standard system activity. By default, this entries are commented out. If you want to collect the standard system activity data, you can customize or un-comment these sample stanzas.

Example 4-14 Extracting record from a file

```
r33n01:/home/kumiko # sar -o sar.out 1 10 > /dev/null
r33n01:/home/kumiko # sar -f sar.out

AIX r33n01 3 5 00C3E3CC4C00    10/15/04

System configuration: lcpu=4 ent=2.00
```

```
18:16:27    %usr    %sys    %wio   %idle   physc   %entc
18:16:28      28      14       0      58    1.01    50.3
18:16:29      28      14       0      58    1.00    50.1
18:16:30      28      14       0      58    1.00    50.1
18:16:31      28      14       0      58    1.00    50.2
18:16:32      28      14       0      58    1.00    50.1
18:16:33      28      14       0      58    1.00    50.2
18:16:34      28      14       0      58    1.00    50.1
18:16:35      28      14       0      58    1.00    50.1
18:16:36      28      14       0      58    1.00    50.1
18:16:37      28      14       0      58    1.00    50.1

Average       28      14       0      58    1.00    50.2
r33n01:/home/kumiko # sar -f sar.out -P 3 -s 18:16:30 -e 18:16:35

AIX r33n01 3 5 00C3E3CC4C00    10/15/04

System configuration: lcpu=4 ent=2.00

18:16:30 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:16:31  3        0       0       0     100    0.16     7.9
18:16:32  3        0       0       0     100    0.16     7.9
18:16:33  3        0       0       0     100    0.16     7.9
18:16:34  3        0       0       0     100    0.16     7.9
18:16:35  3        0       0       0     100    0.16     7.9

Average   3        0       0       0     100    0.16     7.9
r33n01:/home/kumiko #
```

useful combinations

- **sar -P ALL interval number**
- **sar -o ouput.filename interval count > /dev/null &**

4.2.6 The iostat command

The **iostat** command is used to report CPU statistics, input/output statistics, adapters, tty devices, disks and CD-ROMs statistics.

Syntax

iostat [-a] [-s] [**-t**] [-T] [-d [-m] [-A] [-P] [-q | -Q] [-l] [Drives ...] [**Interval**] [**Count**]

flags

-t Specifies tty/cpu report only

parameters

Interval Specifies the amount of time in seconds between each report

Count Specifies the number of reports generated

Example

Starting with AIX 5.3, the **iostat** command reports the percentage of physical processors consumed (%physc), and the percentage of entitlement consumed (%entc). These metrics will only be displayed on shared processor partition or simultaneous multi-threading (SMT) environments. Example 4-15 shows a sample of the **iostat** command with -t flag. The first report of statistics section provides the statistics concerning the time since the system was booted. Each subsequent report covers the time since the previous report. For multiprocessor systems, the CPU values are global averages among all processors.

The first section of the iostat command displays the current system configuration. And the next section reports the following statistics information.

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.

%user Shows the percentage of CPU utilization that occurred while executing at the user level (application).

%sys Shows the percentage of CPU utilization that occurred while executing at the system level (kernel).

%idle Shows the percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.

%iowait Shows the percentage of time that the CPU or CPUs were idle during which the system had an outstanding disk I/O request.

%physc The percentage of physical processors consumed, displayed only if the partition is running with shared processor.

%entc The percentage of entitled capacity consumed, displayed only if the partition is running with shared processor.

Example 4-15 The iostat command with -t flag

```
r33n01: # iostat -t 5

System configuration: lcpu=4 ent=2.00

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait    % physc   %
entc
          0.0          8.2               0.0       0.0       100.0       0.0           0.00
0.1
```

```
          0.6          41.2               0.0       0.0       100.0        0.0           0.00
0.1
          1.6          33.6               2.7       1.4        95.8        0.0           0.00
4.9
          0.8          21.0              27.7      14.3        58.0        0.0           0.00
50.1
          1.4          21.6              27.7      14.3        58.0        0.0           0.00
50.1
          1.6          27.3              11.0       5.7        83.3        0.0           0.00
20.1
          1.8          33.7              23.5       0.1        76.4        0.0           0.00
24.8
          0.0          20.2              47.5       0.1        52.5        0.0           0.00
50.1
          0.0          20.2              47.4       0.1        52.5        0.0           0.00
50.1
          0.0          20.2              47.5       0.1        52.5        0.0           0.00
50.1
r33n01:/usr/lib/dr/scripts #
```

The **iostat** command has been enhanced to supports dynamic configuration changes. If configuration change is detected, the iostat report issues a warning and refreshes the latest system configuration. Example 4-16 shows the output when the iostat command detects dynamic configuration changes.

Example 4-16 The iostat command detects dynamic configuration change

```
r33n05:/ # iostat -t 5 10

System configuration: lcpu=2 ent=2.00

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait    % physc   %
entc
          0.0          8.2              49.4       0.7       49.9        0.0           0.00
50.1
          0.0         40.5              49.3       0.7       50.0        0.0           0.00
50.0
          0.0         20.2              49.4       0.7       49.9        0.0           0.00
50.1
System configuration changed. The current iteration values may be inaccurate.
          0.2         35.7              33.0      23.5       43.3        0.2           0.00
65.5

System configuration: lcpu=4 ent=2.00

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait    % physc   %
entc
```

```
          0.0          44.2          46.9       0.5       52.5       0.0       0.00
50.1
          0.0          40.6          46.9       0.5       52.6       0.0       0.00
50.1
          0.0          20.4          48.9       0.5       50.6       0.0       0.00
52.3
          0.0          20.2          46.9       0.5       52.6       0.0       0.00
50.0
          0.0          20.0          47.0       0.5       52.5       0.0       0.00
50.1
          0.0          20.4          46.9       0.5       52.6       0.0       0.00
50.0
r33n05:/ #
```

useful combinations

- **iostat -t interval count**
- **iostat -t -T interval count**

4.2.7 The vmstat command

The **vmstat** command reports statistics about kernel threads, virtual memory, disks, traps and CPU activity.

Command Syntax

vmstat [-f] [-i] [-s] [-I] [-t] [-v] [PhysicalVolume ...] [**Interval** [**Count**]]

Parameters

Interval	Specifies the amount of time in seconds between each report
Count	Specifies the number of reports generated

Examples

Beginning with AIX 5L Version 5.3, the **vmstat** command reports the number of physical processors consumed (pc), and the percentage of entitlement consumed (ec). These new metrics will be displayed only when the partition is running as a shared processor partition or with simultaneous multi-threading (SMT) enabled. If the partition is running as a dedicated processor partition and with simultaneous multi-threading (SMT) disabled, these new metrics will not be displayed. Example 4-17 on page 209 shows a sample of the **vmstat** command without flag on shared-partition. The first report contains statistics for the time since system startup. Subsequent reports contain statistics collected during the interval since the previous report.

Following statistics information is he columns which related to CPU within the **vmstat** command output.

kthr Kernel thread state changes per second over the sampling interval

- **r** The number of kernel threads placed in run queue
- **b** The number of kernel threads placed in wait queue (awaiting resource or input/output)

faults Trap and interrupt rate averages per second over the sampling interval

- **in** The number of device interrupts
- **sy** The number of system calls
- **cs** The number of kernel thread context switches

cpu Breakdown of percentage usage of CPU time

- **us** The percentage of user time
- **sy** The percentage of system time
- **id** The percentage of CPU idle time
- **wa** The percentage of CPU idle time during which the system had outstanding disk or NFS I/O requests
- **pc** The number of physical processors consumed. Displayed only if the partition is running with shared processor
- **ec** The percentage of entitled capacity consumed. Displayed only if the partition is running with shared processor

Example 4-17 The vmstat command without flag

```
r33n01:/ # vmstat 1 5

System configuration: lcpu=4 mem=7168MB ent=0

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 1  0 149598 1647726   0   0   0    0    0   0   2 13860 133 47  1 52  0  1.01  50.3
 1  0 149600 1647724   0   0   0    0    0   0   1 13700 130 47  1 53  0  1.00  50.1
 2  0 149616 1647708   0   0   0    0    0   0   4 2493708 141 65 10 25  0  1.65  82.3
 2  0 149616 1647708   0   0   0    0    0   0   1 3832368 129 75 15 11  0  2.00 100.0
 2  0 149616 1647708   0   0   0    0    0   0   1 3832602 132 75 15 11  0  2.00 100.0
r33n01:/ #
```

The **vmstat** command has been enhanced to supports dynamic configuration changes. If configuration change is detected, vmstat report issues a warning, and then changes to the latest system configuration. Example 4-18 shows the output when **vmstat** detects dynamic configuration changes.

Example 4-18 The vmstat command detects dynamic configuration change

```
r33n05:/ # vmstat 5

System configuration: lcpu=2 mem=6912MB ent=0

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 0  0 194344 1553879   0   0   0    0    0   0   3   42 146  0  0 99  0  0.00   0.2
 1  0 194346 1553877   0   0   0    0    0   0   0   13 138  0  0 99  0  0.00   0.1
System configuration changed. The current iteration values may be inaccurate.
 4  0 194891 1553331   0   0   0    0    0   0   3  499 191  0 17 82  0  0.47  23.7

System configuration: lcpu=4 mem=6912MB ent=0

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 0  0 194891 1553331   0   0   0    0    0   0   0   17 134  0  0 99  0  0.00   0.1
 0  0 194891 1553331   0   0   0    0    0   0   0   14 136  0  0 99  0  0.00   0.1
^Cr33n05:/ #
```

4.2.8 The ps command

The **ps** command shows current status of processes. With regard to CPU, this command shows how much CPU resource a process is using, and whether processes are being penalized by the system. The **ps** command resides in /usr/bin and is part of the bos.rte.control fileset, which is installed by default from the AIX base installation media.

Syntax

X/Open Standards

ps [-A] [-M] [-N] [-a] [-d] [**-e**] [**-f**] [-k] [-l] [-F format] [-o Format] [-c Clist] [-G Glist] [-g Glist] [-m] [-n NameList] [-p Plist] [-t Tlist] [-U Ulist] [-u Ulist] [**-T pid**] [**-L pidlist**] [-X]

Berkeley Standards

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

Flags

-e Writes information to standard output about all processes, except kernel processes.

-f Generates a full listing.

-k Lists kernel processes.

-o Format Displays information in the format specified by the Format variable. Multiple field specifiers can be specified for the Format variable. For more information of field name, refer to **ps** command reference.

-L pidlist Generates a list of descendants of each and every process ID that has been passed to it in the pidlist variable. The pidlist variable is a list of comma-separated process IDs. The list of descendants from all of the given pid is printed in the order in which they appear in the process table.

-M Lists all 64 bit processes.

-T pid Displays the process hierarchy rooted at a given pid in a tree format using ASCII art. This flag can be used in combination with the -f, -F, -o, and -l flags.

-U Ulist Displays only information about processes with the user ID numbers or login names specified for the Ulist variable. This flag is equivalent to the -u Ulist flag.

a Displays information about all processes with terminals (ordinarily only the user's own processes are displayed).

u Displays user-oriented output. This includes the USER, PID, %CPU, %MEM, SZ, RSS, TTY, STAT, STIME, TIME, and COMMAND columns.

Example

Displaying all non-kernel processes

To display all non-kernel processes, the **ps** command with the combination of the -e and -f flags are used frequently. Example 4-19 on page 212 shows a sample of the **ps** command. Generally, this command reports a long list. You had better to use pipe or redirect output to file.

This command includes the field relevant to CPU in the output.

C Recent used CPU time for process. CPU utilization of process or thread, incremented each time the system clock ticks and the process or thread is found to be running. The value is decayed by the scheduler by dividing it by 2 once per second. For the sched_other

policy, CPU utilization is used in determining process scheduling priority. Large values indicate a CPU intensive process and result in lower process priority whereas small values indicate an I/O intensive process and result in a more favorable priority.

TIME The total CPU time for the process since it started.

Example 4-19 Displaying all non-kernel processes

```
r33n05:/ # ps -ef | pg
     UID    PID   PPID   C    STIME    TTY  TIME CMD
    root      1      0   0   Oct 06      -  0:02 /etc/init
    root 385070 626748   0 15:00:27      -  0:00 telnetd -a
    root 389354      1   0   Oct 06      -  1:50 /usr/sbin/syncd 60
    root 413742      1   0   Oct 06      -  0:00 /usr/lib/errdemon
    root 442370 651328   0   Oct 06      -  0:00 /usr/sbin/syslogd
    root 450780      1   0   Oct 06      -  0:00 /usr/ccs/bin/shlap64
    root 454836 651328   0   Oct 06      -  0:00 /usr/sbin/rsct/bin/IBM.ServiceRMd
    root 487492 598230   0 15:16:15  pts/1  0:00 -ksh
    root 491692 651328   0   Oct 06      -  0:00 /usr/sbin/rsct/bin/IBM.AuditRMd
  kumiko 495722 610458   0 18:20:07  pts/1  0:00 ./cputest
    root 507930 655434   0 18:26:18  pts/0  0:00 ps -ef
    root 532488 651328   0   Oct 06      - 57:37 /usr/sbin/rsct/bin/IBM.CSMAgentRMd
    root 548890 651328   0   Oct 06      -  0:00 /usr/sbin/rsct/bin/IBM.HostRMd
    root 552996 651328   0   Oct 06      -  0:00 /usr/sbin/rsct/bin/IBM.ERrmd
    root 557090 655434   0 18:26:18  pts/0  0:00 pg
    root 573470 651328   0   Oct 06      -  0:00 /usr/sbin/rpc.lockd -d 0
    root 577578      1   0   Oct 06      -  0:00 /usr/sbin/cron
    root 585776      1   0   Oct 06      -  0:00 /usr/sbin/uprintfd
    root 589880 651328   0   Oct 06      -  0:00 /usr/sbin/rsct/bin/IBM.DRMd
    root 593968 651328   0   Oct 06      -  0:08 /usr/sbin/rsct/bin/rmcd -r
    root 598230 626748   0 15:16:15      -  0:00 telnetd -a
    root 602224 651328   0   Oct 06      -  0:00 /usr/sbin/writesrv
    root 606306 651328   0   Oct 06      -  0:00 /usr/sbin/biod 6
  kumiko 610458 487492   0 18:14:23  pts/1  0:00 -ksh
    root 614494 651328   0   Oct 06      -  0:00 /usr/sbin/qdaemon
... lines omitted ...
```

Displaying the percentage of CPU execution time of process

To displaying the percentage of time the process has used the CPU since the process started, use the **ps** command with **u** or **v** flag. Example 4-20 on page 213 shows a sample of the **ps** command with the **u** flag, and %CPU field shows the percentage of CPU execution time.

%CPU The percentage of time the process has used the CPU since the process started. The value is computed by dividing the time the process uses the CPU by the elapsed time of the process. In a multi-processor environment, the value is further divided by the

number of available CPUs because several threads in the same process can run on different CPUs at the same time.

Example 4-20 Displaying the percentage of CPU execution time of process

```
r33n05:/ # ps u
USER        PID %CPU %MEM   SZ  RSS    TTY STAT    STIME  TIME COMMAND
root     659600 43.8  0.0 1316 1324  pts/0 A    19:18:27  0:07 ./memtest
root     655434  0.0  0.0  744  792  pts/0 A    15:00:28  0:00 -ksh
root     557126  0.0  0.0  212  220  pts/0 A    19:18:07  0:00 ./cputest
root     503982  0.0  0.0  804  828  pts/0 A    19:18:35  0:00 ps u
root     487492  0.0  0.0  744  792  pts/1 A    15:16:15  0:00 -ksh
r33n05:/ #
```

Displaying processes related with specified user

To list processes owned by specific users, use the **ps** command with the **-u** flag as shown in Example 4-21.

Example 4-21 Displaying processes related with specified user

```
r33n05:/ # ps -fu kumiko
     UID    PID   PPID   C    STIME    TTY  TIME CMD
  kumiko 495722 610458   0 18:20:07  pts/1  0:00 ./cputest
  kumiko 610458 487492   0 18:14:23  pts/1  0:00 -ksh
  kumiko 659464 610458  90 18:20:20  pts/1  0:14 ./memtest
r33n05:/ #
```

Displaying the specified column

To display a specified format with field specifiers, use the **ps** command with the -o flag. Example 4-22 shows a sample of displaying only specified field using the -o flag.

Example 4-22 Displaying the specified column

```
r33n05:/ # ps -o pid,ppid,ruser,cpu,nice,time,comm
   PID   PPID    RUSER  CP NI        TIME COMMAND
557122 655434     root   0 24    00:00:11 cputest
655434 385070     root   0 20    00:00:11 ksh
659498 655434     root   0 20    00:00:11 ps
r33n05:/ #
```

Displaying the 64-bit processes

To list all the 64-bit processes, use the **ps** command with the **-M** flag, as shown in Example 4-23 on page 214.

Example 4-23 Displaying the 64-bit processes

```
r33n05:/ # ps -efM
     UID    PID   PPID   C    STIME    TTY  TIME CMD
    root 450780      1   0   Oct 06      -  0:00 /usr/ccs/bin/shlap64
  kumiko 495722 610458   0 18:20:07  pts/1  0:00 ./cputest
    root 630858 651328   0   Oct 06      -  0:06 /usr/sbin/snmpmibd
r33n05:/ #
```

Displaying the process hierarchy

To display the process hierarchy in a tree format using ASCII art, use the **ps** command with the **-T** flag as shown in Example 4-24.

Example 4-24 Displaying the process hierarchy

```
r33n05:/ # ps -T 651328
    PID    TTY  TIME CMD
 651328      -  0:00 srcmstr
 442370      -  0:00    |\--syslogd
 454836      -  0:00    |\--IBM.ServiceRMd
 491692      -  0:00    |\--IBM.AuditRMd
 532488      - 57:42    |\--IBM.CSMAgentRMd
 548890      -  0:00    |\--IBM.HostRMd
 552996      -  0:00    |\--IBM.ERrmd
 573470      -  0:00    |\--rpc.lockd
 589880      -  0:00    |\--IBM.DRMd
 593968      -  0:08    |\--rmcd
 602224      -  0:00    |\--writesrv
 606306      -  0:00    |\--biod
 614494      -  0:00    |\--qdaemon
 618556      -  0:00    |\--rpc.statd
 622642      -  1:45    |\--muxatmd
 626748      -  0:00    |\--inetd
 385070      -  0:00    |   |\--telnetd
 655434  pts/0  0:00    |   |    \--ksh
 557122  pts/0  0:00    |   |       |\--cputest
 659528  pts/0  0:00    |   |        \--ps
 598230      -  0:00    |    \--telnetd
 487492  pts/1  0:00    |        \--ksh
 610458  pts/1  0:00    |            \--ksh
 495722  pts/1  0:00    |                \--cputest
 630858      -  0:06    |\--snmpmibd64
 634942      -  0:00    |\--portmap
 643140      -  0:00     \--sendmail
r33n05:/ #
```

Useful combination

- **ps -ef | grep [condition]**
- **ps -el**
- **ps -ef | sort +3 -r | head -n 10**
- **ps -fu [user_id]**
- **ps eww | grep [PID]**

4.2.9 The trace tool

The **trace** command is a daemon that records selected system events. The trace daemon configures a trace session and starts the collection of system events. The data collected by the trace daemon is recorded in the trace log. This trace log has binary format data. The **trcrpt** command is used to format report from the trace log.

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 /usr/bin/trace is a symbolic link to /usr/sbin/trace. The **trcnm** and the 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.

Syntax

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] [**-r reglist**] [-s] [**-A process-id [,process-id]**] [-t thread-id [,thread-id]] [-x program-specification | -X program-specification] [-l] [-P trace-propagation][-L Size] [**-T Size**]

Flags

-a Runs the trace daemon asynchronously. 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 have symbolic link to /usr/bin/trace.

-A process-id[,process-id]
Traces only the listed processes and, optionally, their children. A process-id is a decimal number. Multiple process IDs can be separated by commas or enclosed in quotes and

separated by spaces. The -A flag is only valid for trace channel 0. The -A and -g flags are incompatible.

-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. Since 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.

-j Event[,Event] Specifies the trace events for which you want to collect.The Event list items can be separated by commas, or enclosed in double quotation marks and separated by commas or blanks.

-J Event-group [, Event-group]
Specifies the event groups for which you want to collect.

-o Name Overrides the /var/adm/ras/trcfile default trace log file and writes trace data to a user specified file.

-r reglist Optional, and only valid for a trace run on a 64-bit kernel. The reglist options are separated by commas, or enclosed in quotation marks, and separated by blanks. Up to 8 registers may be specified. Following reglist values are supported.

PURR	The PURR. Register for this cpu
MCR0, MCR1, MCRA	The MCR. Registers, 0, 1, and A
PMC1, PMC2, ... PMC8	PMC. Registers 1 through 8.

-T Size Overrides the default trace buffer size of 128 KB with the **Size** byte.

The trcstop command

trcstop

The trcnm command

trcnm [-a [FileName]] | [FileName] | -KSymbol1 ...

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 [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.

-j Displays the list of hook IDs.

-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.

-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.

Examples

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. If you specify many or all hooks, the trace log file become very large.

Terminology used for trace

In order to understand the trace tool, you need 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 want to monitor the open() system call, this

event has hook15B. Trace hooks can be displayed with **trcrpt -j**. Example 4-25 shows the trace hook lists.

Hook ID A unique number is assigned to a trace hook 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/include/sys/trchkid.h. Example 4-26 on page 219 shows a part of /usr/include/sys/trchkid.h.

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. Using the -C flag 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. Depending on the amount of data that is being collected, the trace log file can become huge size quickly.

Example 4-25 Displaying the trace hook

```
r33n05:/ # trcrpt -j | more
...line is omitted...
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
```

```
179 LAPI
180 SIGACTION SYSTEM CALL
181 SIGCLEANUP
...line is omitted...
```

Example 4-26 Displaying the trace hook ID

```
r33n05:/ # more /usr/include/sys/trchkid.h
...skipping...
#define HKWD_SYSC_LOADTBL         0x15100000
#define HKWD_SYSC_LOCKF           0x15200000
#define HKWD_SYSC_LOCKX           0x15300000
#define HKWD_SYSC_LSEEK           0x15400000
#define HKWD_SYSC_MACCTL          0x15500000
#define HKWD_SYSC_MKDIR           0x15600000
#define HKWD_SYSC_MKNOD           0x15700000
#define HKWD_SYSC_MNTCTL          0x15800000
#define HKWD_SYSC_MOUNT           0x15900000
#define HKWD_SYSC_NICE            0x15a00000
#define HKWD_SYSC_OPEN            0x15b00000
#define HKWD_SYSC_OPENX           0x15c00000
#define HKWD_SYSC_OUNAME          0x15d00000
#define HKWD_SYSC_PAUSE           0x15e00000
#define HKWD_SYSC_PIPE            0x15f00000
#define HKWD_SYSC_PLOCK           0x16000000
#define HKWD_SYSC_PROFIL          0x16100000
#define HKWD_SYSC_PTRACE          0x16200000
#define HKWD_SYSC_READ            0x16300000
...line is omitted...
```

Running trace interactively

When you use the trace command without -a flag, the trace daemon runs interactive mode. In interactive mode, the trace daemon 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.

Example 4-27 on page 220 shows how to run trace daemon interactively. In this example, trace daemon record system event of the **ls** command as well as other processes running on the system, and /var/adm/ras/trcfile is used for trace log file.

Example 4-27 Running trace interactively

```
r33n05:/ # trace
-> !ls
.SPOT             audit             lpp               tftpboot
.Xdefaults        bin               mnt               tmp
.kshrc            dev               opt               u
.mwmrc            etc               proc              unix
.profile          export            sbin              usr
.rhosts           home              smit.log          var
.rhosts.prev      lib               smit.script
.sh_history       lost+found        smit.transaction
-> q
r33n05:/ # ls -l /var/adm/ras/trcfile
-rw-rw-rw-   1 root     system      1216448 Oct 21 11:20 /var/adm/ras/trcfile
r33n05:/ #
```

Running trace asynchronously

When you use the trace command with -a flag, the trace daemon runs asynchronous mode. Example 4-28 shows how to run trace daemon asynchronously. In this example, trace daemon record system event of the **ls** command as well as other processes running on the system, and /var/adm/ras/trcfile is used for trace log file. This method is used to avoid delays when the command finishes. And by using this method, the trace file is considerably smaller than the interactive mode shown in Example 4-27.

Example 4-28 Running trace asynchronously

```
r33n05:/ # trace -a; ls; trcstop
.SPOT             audit             lpp               tftpboot
.Xdefaults        bin               mnt               tmp
.kshrc            dev               opt               u
.mwmrc            etc               proc              unix
.profile          export            sbin              usr
.rhosts           home              smit.log          var
.rhosts.prev      lib               smit.script
.sh_history       lost+found        smit.transaction
r33n05:/ # ls -l /var/adm/ras/trcfile
-rw-rw-rw-   1 root     system       523560 Oct 21 11:23 /var/adm/ras/trcfile
r33n05:/ #
```

Running trace all system event for 10 seconds

Example 4-29 on page 221 shows how to run trace on the entire system for 10 seconds. This command traces all system activity and includes all trace hooks. The file /var/adm/ras/trcfile is used for trace log file.

Example 4-29 Running trace all system event for 10 seconds

```
r33n05:/ # trace -a; sleep 10; trcstop
r33n05:/ # ls -l /var/adm/ras/trcfile
-rw-rw-rw-   1 root     system      2054688 Oct 21 11:43 /var/adm/ras/trcfile
r33n05:/ #
```

Tracing to a specific log file

If you want to specify the trace log file, use -o flag. Example 4-30 shows a sample how to run trace asynchronously and output the trace file to /tmp/my_trace_log.

Example 4-30 Tracing to a specific log file

```
r33n05:/ # trace -a -o /tmp/my_trace_log; ls; trcstop
.SPOT             audit             lpp               tftpboot
.Xdefaults        bin               mnt               tmp
.kshrc            dev               opt               u
.mwmrc            etc               proc              unix
.profile          export            sbin              usr
.rhosts           home              smit.log          var
.rhosts.prev      lib               smit.script
.sh_history       lost+found        smit.transaction
r33n05:/ # ls -l /tmp/my_trace_log
-rw-rw-rw-   1 root     system       536928 Oct 21 11:51 /tmp/my_trace_log
r33n05:/ #
```

Tracing using one set of buffers per CPU

Only one trace files is used to record all CPU system event by default. By using the -C option, trace daemon used one set of buffers per CPU, and produces one file per CPU as show in Example 4-31. This consume large amounts of memory and file space for collecting system events. In this example, four individual files (one for each CPU) and the master file /var/adm/ras/trcfile are created.

Example 4-31 Tracing using one set of buffers per CPU

```
r33n05:/ # trace -a -C all; sleep 10; trcstop
Warning:  The available space, 127672320 bytes, may be insufficient.
r33n05:/ # ls -l /var/adm/ras/trcfile*
-rw-rw-rw-   1 root     system        50528 Oct 21 11:56 /var/adm/ras/trcfile
-rw-rw-rw-   1 root     system       105728 Oct 21 11:56 /var/adm/ras/trcfile-0
-rw-rw-rw-   1 root     system       421664 Oct 21 11:56 /var/adm/ras/trcfile-1
-rw-rw-rw-   1 root     system      1074336 Oct 21 11:56 /var/adm/ras/trcfile-2
-rw-rw-rw-   1 root     system       371016 Oct 21 11:56 /var/adm/ras/trcfile-3
r33n05:/ #
```

Formatting the trace log file

Using the **trcrpt** command, you can format the trace log file. Generally, this command displays many lines, you had better use pipe or -o flag. Example 4-32 shows a sample of formatting a trace log using the **trcrpt** command.

Example 4-32 Formats the trace log file

```
r33n05:/ # trace -a -o /tmp/trace.log; sleep 10; trcstop
r33n05:/ #
r33n05:/ # trcrpt /tmp/trace.log | more
Thu Oct 21 14:11:00 2004
System: AIX 5.3 Node: r33n05
Machine: 00C3E3CC4C00
Internet Address: 81280D45 129.40.13.69
The system contains 80 cpus, of which 80 were traced.
Buffering: Kernel Heap
This is from a 64-bit kernel.
Tracing all hooks.


trace -a -o /tmp/trace.log


ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001    0.000000000       0.000000                   TRACE ON channel 0
                                                    Thu Oct 21 14:11:00 2004
104    0.000022621       0.022621           return from system call
101    0.000023710       0.001089           _getppid LR = D023335C
104    0.000024113       0.000403           return from _getppid [0 usec]
101    0.000024621       0.000508           kill LR = 10009BC0
14E    0.000025142       0.000521                   kill: signal SIGUSR1 to
process 610342 trace
3B7    0.000025621       0.000479   SECURITY: privcheck entry: p=4
3B7    0.000025907       0.000286   SECURITY: privcheck exit: rc=0
119    0.000026865       0.000958                   pidsig:   pid=610342
signal=SIGUSR1 lr=33E78
11F    0.000027798       0.000933                   setrq:   cmd=trace
pid=610342 tid=1130525 prio
rity=60 policy=0 rq=0002
492    0.000028235       0.000437                   h_call: start H_PROD
iar=2922C p1=0002 p2=00FF
 p3=0000
492    0.000029588       0.001353                   h_call: end H_PROD
iar=2922C rc=0000
104    0.000030176       0.000588           return from kill [6 usec]
101    0.000030752       0.000576           close LR = 10009BD8
12E    0.000031184       0.000432           close fd=0
104    0.000031768       0.000584           return from close [1 usec]
```

```
101    0.000032218       0.000450          close LR = 10009BE4
... lines omitted ...
```

Formatting the trace log file with specified columns

Using the -O flag, the **trcrpt** command can format the trace log file with specified column. Following options are supported for -O flag.

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 within the range of 0 through 500. The column headings are included on each page. No page breaks are present when the default value of 0 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, trace -C. For example, if you're reading a report from a system having 4 CPUs, and the reported CPUs value goes from 4 to 3, then you know that there are no more hooks to be reported for that CPU.

PURR=[on|off] Tells **trcrpt** to show the PURR along with any timestamps. The PURR is displayed following any timestamps. If the PURR is not valid for the processor traced, the elapsed time is shown instead of the PURR. If

the PURR is valid, or the cpuid is unknown, but wasn't traced for a hook, the PURR field contains asterisks (*).

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 reporting of the time stamp associated with an event in the trace report. The possible values are:

0 Time elapsed since the trace was started and delta time from the previous event. The elapsed time is in seconds and the delta time is in milliseconds. Both values are reported to the nearest nanosecond. This is the default.

1 Short elapsed time. Reports only the elapsed time (in seconds) from the start of the trace. Elapsed time is reported to the nearest microsecond.

2 Microsecond delta time. This is like 0, except the delta time is in microseconds, reported to the nearest microsecond.

3 No time stamp.

Example 4-33 shows a sample of the **trcrpt** command with -O flag. In this example, CPU ID and process ID are specified to report.

Example 4-33 Formatting the trace log file with specified columns

```
r33n05:/ # trace -a -C all -o /tmp/trace2.log; sleep 10; trcstop
r33n05:/ #
r33n05:/ # trcrpt -O pid=on,cpuid=on /tmp/trace2.log | more
... skip ...
492 2   16392         0.000038214       0.000630                  h_call: end H_CEDE iar
=1BF92D03D9ED9 rc=0000
492 3   20490         0.000038218       0.000004                  h_call: end H_CEDE iar
=1BF92D03D9ED7 rc=0001
100 3   20490         0.000039600       0.001382                          DATA ACCESS PA
GE FAULT iar=27744 cpuid=03
116 1   557308        0.000039911       0.000311                  xmalloc fastpath:   si
```

```
ze=02C0 align=0007 heap=F100060000000000
106 2   487678        0.000040046       0.000135                  dispatch:   cmd=trace
pid=487678 tid=1134655 priority=60 old_tid=16393 old_priority=255 CPUID=2
116 1   557308        0.000040348       0.000302                  xmalloc return:     re
sult=F1000600179D3800
200 3   20490         0.000041042       0.000694                  resume  wait iar=27744
 cpuid=03
116 1   557308        0.000041067       0.000025                  xmalloc fastpath:   si
ze=0200 align=0007 heap=F100060000000000
200 2   487678        0.000041289       0.000222                  resume  trace iar=A8A8
C cpuid=02
116 1   557308        0.000041294       0.000005                  xmalloc return:     re
sult=F100060015AF3E00
492 3   20490         0.000041630       0.000336                  h_call: start H_CEDE i
ar=500B p1=1BF92D03DA3DA p2=0000 p3=0000
104 2   487678        0.000042004       0.000374           return from system call. error
 EINTR
200 2   487678        0.000044466       0.002462                  resume  trace iar=2ED9
8 cpuid=02
... lines omittes ...
```

Reporting only specified process related event(s)

There are two methods for reporting only specified process related event:

– First, using the **trace** command with the **-A** flag. Using the -A flag, the trace daemon records only the specified processes. In versions prior to AIX 5L Version 5.3, the trace daemon traced the entire system event. Beginning with AIX 5L Version 5.3, the trace command enhanced to enable recording only for specified processes, threads or programs. This enhancement can save space in the trace file and also helps to focus on just the process or thread you want to see. Example 4-34 shows a sample of tracing only specified process.

Example 4-34 Tracing only specified process

```
r33n05:/ # ps -ef | grep inetd
    root 598252 495628   0 16:46:58  pts/1  0:00 grep inetd
    root 626748 651328   0   Oct 06      -  0:00 /usr/sbin/inetd
r33n05:/ # trace -a -A 626748; sleep 10; trcstop
r33n05:/ # trcrpt -O pid=on | more
... skip ...

trace -a -A 626748


ID  PID      I    ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001 598260        0.000000000       0.000000                   TRACE ON channel 0
```

```
                                                        Thu Oct 21 16:47:41 2004
200 626748        1.738538012    1738.538012                   resume  inetd iar=540BC cp
uid=FFFFFFFF
104 626748        1.738547256       0.009244           return from system call
101 626748        1.738550004       0.002748           naccept LR = 10003120
252 626748        1.738550777       0.000773           SOCK accept sofd=12 name=000000000
0000000 namelen=0
254 626748        1.738552806       0.002029                   MBUF m_get canwait=M_WAIT
type=MT_SONAME callfrom=000000000018DC4C
254 626748        1.738553651       0.000845                   MBUF return from m_get mbu
f=F100061001F78400 dataptr=F100061001F78468
252 626748        1.738554071       0.000420           SOCK soaccept so=F100061006B1A000
nam=F100061001F78400
52F 626748        1.738555231       0.001160   SEC CRED: crref  callfrom=00000000002D75AC
 callfrom2=000000000018E02C pid=626748 (inetd)
52F 626748        1.738555500       0.000269   SEC CRED: crfree  callfrom=00000000002D75E
0 callfrom2=000000000018E02C pid=626748 (inetd)
535 626748        1.738556105       0.000605                   TCP tcp_usrreq so=F1000610
06B1A000 req=0000000000000005 m=0000000000000000 nam=F100061001F78400
539 626748        1.738556617       0.000512                   PCB in_setpeeraddr inp=F10
0061006B1A278 nam=F100061001F78400 af=0000000000000018
535 626748        1.738556978       0.000361                   TCP tcp_usrreq_err so=F100
061006B1A000 error=0000000000000000
252 626748        1.738557521       0.000543           SOCK return from soaccept so=F1000
61006B1A000 error=0
... lines omitted ...
```

– Second, using the **trcrpt** command with **-p** flag. If you have trace log file includes all system activity, you can extract the process related event using the trcrpt command with -p flag. Example 4-35 shows a sample of extracting the process related event. In this example, the trace log file includes entire system event, and the trcrpt extracts only the event related to process which has PID 610382.

Example 4-35 Extracting the process related event

```
r33n05:/ # ps -ef | grep memtest
    root 508054 655474  72 16:40:46  pts/0  0:01 ./memtest
r33n05:/ # trace -a; sleep 10; trcstop
r33n05:/ # trcrpt -p 508054 | more
... skip ...

trace -a


ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

101    0.000000000       0.000000           kwrite LR = D0235FD0
```

```
19C    0.000000453       0.000453          write(1,FFFFFFFFF09148D0,19)
104    0.000001525       0.001072          return from kwrite [2 usec]
101    0.000002852       0.001327          kwrite LR = D0235FD0
19C    0.000003281       0.000429          write(1,FFFFFFFFF09148D0,19)
104    0.000004306       0.001025          return from kwrite [1 usec]
101    0.000005638       0.001332          kwrite LR = D0235FD0
19C    0.000006029       0.000391          write(1,FFFFFFFFF09148D0,19)
104    0.000007012       0.000983          return from kwrite [1 usec]
101    0.000008319       0.001307          kwrite LR = D0235FD0
19C    0.000008735       0.000416          write(1,FFFFFFFFF09148D0,19)
104    0.000009882       0.001147          return from kwrite [2 usec]
101    0.000011378       0.001496          kwrite LR = D0235FD0
19C    0.000011773       0.000395          write(1,FFFFFFFFF09148D0,19)
104    0.000012857       0.001084          return from kwrite [1 usec]
101    0.000014189       0.001332          kwrite LR = D0235FD0
19C    0.000014609       0.000420          write(1,FFFFFFFFF09148D0,19)
104    0.000015743       0.001134          return from kwrite [2 usec]
101    0.000017109       0.001366          kwrite LR = D0235FD0
19C    0.000017504       0.000395          write(1,FFFFFFFFF09148D0,19)
... lines omitted ...
```

Reporting only specified hook event(s)

There are two methods for reporting only specified trace hook event.

– First, using the **trace** command with the **-j** or **-J** flag. Using the -j flag, you can specify the trace event which you want to collect. Using the -J flag, you can specify the event-group which you want to collect.

  Example 4-36 shows a sample of tracing only specified hook event. In this example, only the system event which has trace hook 0x15B(it means open() system call) is recorded.

Example 4-36 Tracing only specified event

```
r33n05:/ # trace -j 15B -a ; sleep 10; trcstop
r33n05:/ # trcrpt | more

... skip ...

trace -j 15B -a


ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001    0.000000000       0.000000                  TRACE ON channel 0
                                                   Thu Oct 21 17:37:54 2004
15B    0.001030802       1.030802          open fd=3
15B    0.404840773     403.809971          open fd=3
15B    0.408591970       3.751197          open fd=3
```

```
15B    0.410061609       1.469639           open fd=3 _FLARGEFILE
15B    0.412220483       2.158874           open fd=3 _FLARGEFILE
15B    0.412356823       0.136340           open fd=3 _FLARGEFILE
15B    0.412592924       0.236101           open fd=3 _FLARGEFILE
15B    0.413577794       0.984870           open fd=3 _FLARGEFILE
15B    0.421370865       7.793071           open fd=3 _FLARGEFILE
15B    0.421716424       0.345559           open fd=3 _FLARGEFILE
15B    0.422057873       0.341449           open fd=3 _FLARGEFILE
15B    0.422317827       0.259954           open fd=3 _FLARGEFILE
15B    0.425110983       2.793156           open fd=3 _FLARGEFILE
15B    0.425318420       0.207437           open fd=3 _FLARGEFILE
15B    0.426369042       1.050622           open fd=3 _FLARGEFILE
15B    0.426418831       0.049789           open fd=3 _FLARGEFILE
15B    0.426633155       0.214324           open fd=3 _FLARGEFILE
15B    0.437612941      10.979786           open fd=3 _FLARGEFILE
... lines omitted ...
```

– Second, using the **trcrpt** command with **-d** or **-D** flag. If you have trace log file includes all system activity, you can extract only specified event using the trcrpt command with -d or -D flag. Using the -d flag, you can extract the trace event only you want to see. Using the -J flag, you can extract the event-group only you want to see. Example 4-37 shows a sample of extracting only specified hook event. In this example, only the system event which has trace hook 0x12E(it means close() system call) is extracted.

Example 4-37 Extracting only specified event

```
r33n05:/ # trace -a -T 1310720; sleep 10; trcstop
r33n05:/ # trcrpt -d 12E| more
... skip ...

trace -a -T 1310720


ID     ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

001    0.000000000       0.000000                  TRACE ON channel 0
                                                   Thu Oct 21 17:57:44 2004
12E    0.000030025       0.030025           close fd=0
12E    0.000031537       0.001512           close fd=1
12E    0.000032823       0.001286           close fd=2
12E    0.000051021       0.018198           close fd=4
12E    0.000059033       0.008012           close fd=5
12E    0.001025827       0.966794           close fd=10
12E    0.001171386       0.145559           close fd=3
12E    3.181724995    3180.553609           close fd=3
12E    3.182076970       0.351975           close fd=3
12E    3.182180945       0.103975           close fd=3
```

```
12E    3.182238554       0.057609          close fd=3
006    6.261873882    3079.635328                  TRACEBUFFER WRAPAROUND 0000
12E    7.786841453    1524.967571          close fd=12
12E    8.182368760     395.527307          close fd=3
12E    8.182719710       0.350950          close fd=3
... lines omitted ...
```

Useful combinations

- **trace -a; command; trcstop**
- **trace -a -C all**
- **trace -a -j [trace_event]**
- **trace -a -J [event_group]**
- **trcrpt -o [file_name]**
- **trcrpt -d [trace_event]**
- **trcrpt -p [PID]**

4.2.10 The curt command

The CPU Usage Reporting Tool (**curt**) is used to generate statistics report related to CPU utilization and process/thread activity from a trace log file. For information about trace, refer to 4.2.9, “The trace tool” on page 215. The curt command resides in /usr/bin and is part of the bos.perf.tools fileset, which is installable from the AIX base installation media.

Syntax

curt **-i inputfile** [**-o outputfile**] [**-n gennamesfile**] [**-m trcnmfile**] [-a pidnamefile] [-f timestamp] [-l timestamp] [**-r PURR**] [-ehpstP]

Flags

-i inputfile	Specifies the input AIX trace file to be analyzed.
-o outputfile	Specifies the output file (default is stdout).
-n gennamesfile	Specifies a names file produced by gennames.
-m trcnmfile	Specifies a names file produced by trcnm.
-r PURR	Uses the PURR register to calculate CPU times.

Parameters

inputfile	The AIX trace file that should be processed by curt.

Examples

The curt command reads a raw format trace file and generate a report which contains summaries on CPU utilization and either process or thread activity. This

report 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. The trace file which is gathered using the trace command should contain at least following trace events.

- HKWD_KERN_SVC (101)
- HKWD_KERN_SYSCRET (104)
- HKWD_KERN_FLIH (100)
- HKWD_KERN_SLIH (102)
- HKWD_KERN_SLIHRET (103)
- HKWD_KERN_DISPATCH (106)
- HKWD_KERN_RESUME (200)
- HKWD_KERN_IDLE (10C)
- HKWD_SYSC_FORK (139)
- HKWD_SYSC_EXECVE (134)
- HKWD_KERN_PIDSIG (119)
- HKWD_SYSC__EXIT (135)
- HKWD_SYSC_CRTHREAD (465)
- HKWD_KERN_INITP (210)
- HKWD_NFS_DISPATCH (215)
- HKWD_CPU_PREEMPT (419)
- HKWD_DR (38F)
- HKWD_KERN_PHANTOM_EXTINT (47F)
- HKWD_KERN_HCALL (492)
- HKWD_PTHREAD_VPSLEEP (605)
- HKWD_PTHREAD_GENERAL (609)

Trace event-group “curt” also contains these event. Example 4-38 shows event list of curt event group.

Example 4-38 event list of curt event group

```
r33n05:/ # trcevgrp -l curt
curt - Hooks for CURT performance tool (reserved)

100,101,102,103,104,106,10C,119,134,135,139,200,210,215,38F,465,419,47F,492,605
,609
r33n05:/ #
```

Preparing for curt report

To generate the curt report, you need to prepare the raw format trace file. Example 4-39 shows a sample of creating a trace file. In this example we run the trace command with -C all option, and to merge the trace files we run the **trcrpt** command. 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 **curt** to output more names, especially interrupt handlers.

Example 4-39 Preparing curt report

```
r33n05:/ # trace -a -C all ; sleep 10; trcstop
r33n05:/ # ls /var/adm/ras/trcfile*
/var/adm/ras/trcfile    /var/adm/ras/trcfile-1  /var/adm/ras/trcfile-3
/var/adm/ras/trcfile-0  /var/adm/ras/trcfile-2
r33n05:/ # trcrpt -r -C all > /tmp/trace.r
r33n05:/ # trcnm > /tmp/trcnm.out
r33n05:/ # gennames > /tmp/gennames.out
```

Creating curt report

Example 4-40 shows a sample of creating the curt report. Default curt report includes the following information.

- 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
- Hypervisor Calls Summary
- System NFS Calls Summary
- FLIH Summary
- SLIH Summary

Example 4-40 Creating a curt report

```
r33n05:/ # curt -i /tmp/trace.r -m /tmp/trcnm.out -n /tmp/gennames.out >
/tmp/curt.out
r33n05:/ #
```

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 4-41.

Example 4-41 General information

```
r33n05:/ # more /tmp/curt.out
Run on Fri Oct 22 11:14:23 2004
Command line was:
curt -i /tmp/trace.r -m /tmp/trcnm.out -n /tmp/gennames.out
----
AIX trace file name = /tmp/trace.r
AIX trace file size = 10178908
AIX trace file created = Fri Oct 22 09:43:29 2004

Command used to gather AIX trace was:
  trace -a -C all

... lines omitted ...
```

System summary

The next part of the default output is the system summary. This section describes the time spent by the system as a whole (all processors) in various execution modes (see Example 4-42 on page 234). This section has the following fields.

processing total time 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 time 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.

processing category This column gives execution modes. These mode are as follows.

APPLICATION The sum of times spent by all processors in User (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.

HCALL The sum of times spent by all processors doing Hypervisor Calls. This is the portion of time that a processor spends executing in the hypervisor code providing services directly requested by the kernel.

KPROC The sum of times spent by all processors executing kernel processes other than the IDLE process and NFS processes. This is the portion of time that a processor spends executing specially created dispatchable processes which only execute kernel code.

NFS The sum of times spent by all processors executing NFS operations. NFS operations begin with RFS_DISPATCH_ENTRY and end with RFS_DISPATCH_EXIT subhooks.

FLIH The sum of times spent by all processors in FLIHs (first level interrupt handlers).

SLIH The sum of times spent by all processors in SLIHs (second level interrupt handlers).

DISPATCH The sum of times spent by all processors in the AIX dispatch code. This sum includes the time spent in dispatching all threads (i.e. it includes the dispatches 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 the DISPATCH category includes the IDLE DISPATCH category's time, the IDLE DISPATCH category's time is not separately added to calculate either CPU(s) busy time or TOTAL (see below).

CPU(s) busy time The sum of times spent by all processors executing in application, syscall, kproc, 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 IDLE. This number is referred to as “total processing time”.

Total Physical CPU time (msec)

The real time the CPU(s) were running (not preempted).

Physical CPU percentage

The Physical CPU(s) Time as a percentage of total time.

Example 4-42 System summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    System Summary
                    --------------
 processing       percent       percent
 total time    total time     busy time
     (msec)  (incl. idle)  (excl. idle)  processing category
===========   ===========   ===========  ==================
      27.49         27.05         27.05  APPLICATION
      45.22         44.49         44.49  SYSCALL
      13.99         13.76         13.76  HCALL
       3.40          3.34          3.34  KPROC (excluding IDLE and NFS)
       0.00          0.00          0.00  NFS
       7.15          7.03          7.03  FLIH
       3.02          2.97          2.97  SLIH
       1.37          1.35          1.35  DISPATCH (all procs. incl. IDLE)
       0.58          0.57          0.57  IDLE DISPATCH (only IDLE proc.)
-----------   ----------       -------
     101.64         99.99        100.00  CPU(s) busy time
       0.01          0.01                IDLE
-----------    ----------
     101.65                             TOTAL

Avg. Thread Affinity = 1.00

Total Physical CPU time (msec) = 103.43
Physical CPU percentage        = 0.86
... lines omitted ...
```

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". Beginning with AIX 5L Version 5.3, some fields related to Hypervisor Call are added.

Total number of H_CEDE
: The number of H_CEDE hypervisor call done by this processor; with preemption indicates the number of H_CEDE calls resulting in preemption.

Total number of H_CONFER
: The number of H_CONFER hypervisor call done by this processor; with preemption indicates the number of H_CONFER calls resulting in preemption.

A sample of processor summary output is shown in Example 4-43.

Example 4-43 Processor summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
              Processor Summary  processor number 1
              ------------------------------------

 processing       percent       percent
 total time    total time     busy time
     (msec)  (incl. idle)  (excl. idle)  processing category
===========  ===========  ===========  ==================
      20.66        59.91        59.91  APPLICATION
       7.94        23.03        23.03  SYSCALL
       8.62        25.00        25.00  HCALL
       1.55         4.50         4.50  KPROC (excluding IDLE and NFS)
       0.00         0.00         0.00  NFS
       2.11         6.13         6.13  FLIH
       1.84         5.33         5.33  SLIH
       0.38         1.10         1.10  DISPATCH (all procs. incl. IDLE)
       0.16         0.46         0.46  IDLE DISPATCH (only IDLE proc.)
-----------    ----------    -------
      34.49        99.99       100.00  CPU(s) busy time
       0.00         0.01              IDLE
-----------    ----------
      34.49                          TOTAL

Avg. Thread Affinity = 1.00

Total number of process dispatches = 175
Total number of idle dispatches     = 156

Total Physical CPU time (msec) = 44.23
Physical CPU percentage        = 0.72
Physical processor affinity    =  0.960751
Dispatch Histogram for processor (PHYSICAL CPUid : times_dispatched).
    PHYSICAL   CPU 1 : 586
```

```
Total number of preemptions = 586
Total number of H_CEDE      = 574      with preeemption = 573
Total number of H_CONFER    = 0      with preeemption = 0

              Processor Summary  processor number 2
              ------------------------------------
 processing       percent       percent
 total time    total time     busy time
     (msec)  (incl. idle)  (excl. idle)  processing category
===========   ===========   ===========  ==================
       6.83         12.84         12.85  APPLICATION
      37.28         70.12         70.12  SYSCALL
       5.37         10.09         10.10  HCALL
       1.85          3.47          3.47  KPROC (excluding IDLE and NFS)
... lines omitted ...
```

Application summary by 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 output has two main sections, of which one shows the total processing time of the thread in milliseconds (processing total in miliseconds), 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). PID (process ID) and TID (thread ID) are always given in decimal. A sample of application summary by TID is shown in Example 4-44.

Example 4-44 Application summary by TID

```
r33n05:/ # more /tmp/curt.out
... skip ...

                Application Summary (by Tid)
                ----------------------------
    -- processing total (msec) --          -- percent of total processing time --
   combined  application      syscall     combined  application      syscall  name (Pid  Tid)
   ========  ===========      =======     ========  ===========      =======
==================
    21.8643       3.5268      18.3375      21.5099       3.4697      18.0403  (610532  1118323)
    16.5744      16.5744       0.0000      16.3058      16.3058       0.0000  (487600  1167509)
     3.1864       1.1024       2.0840       3.1348       1.0846       2.0502  trcstop(598254
1179805)
     2.5426       2.5426       0.0000       2.5014       2.5014       0.0000  (487600  1142923)
     1.2743       0.9171       0.3572       1.2537       0.9023       0.3514  muxatmd(622642
798857)
     1.1672       1.1672       0.0000       1.1483       1.1483       0.0000  getty(639088
774293)
```

```
     1.0533       1.0533       0.0000       1.0362       1.0362       0.0000  (487600  1044593)
     0.4702       0.4702       0.0000       0.4626       0.4626       0.0000  (487600  1163459)
     0.3353       0.0791       0.2562       0.3299       0.0778       0.2520  ksh(655440
1073245)
     0.3098       0.0958       0.2139       0.3048       0.0943       0.2105  ksh(659476
1056821)
     0.1192       0.0084       0.1108       0.1172       0.0083       0.1090  (598252  1179803)
     0.0852       0.0333       0.0520       0.0839       0.0327       0.0511  rmcd(593968
684231)
     0.0295       0.0113       0.0182       0.0291       0.0111       0.0179  snmpmibd64(630858
807073)
     0.0094       0.0067       0.0028       0.0093       0.0065       0.0027  syncd(389354
409619)
... lines omitted ...
```

Application summary by 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. A sample of application summary by PID is shown in Example 4-45.

Example 4-45 Application summary by PID

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Application Summary (by Pid)
                    ----------------------------
    -- processing total (msec) --          -- percent of total processing time --
   combined  application      syscall     combined  application      syscall  name (Pid)(Thread
Count)
   ========  ===========      =======     ========  ===========      =======
===================
    21.8643       3.5268      18.3375      21.5099       3.4697      18.0403  (610532)(1)
    20.6405      20.6405       0.0000      20.3060      20.3060       0.0000  (487600)(4)
     3.1864       1.1024       2.0840       3.1348       1.0846       2.0502
trcstop(598254)(1)
     1.2743       0.9171       0.3572       1.2537       0.9023       0.3514
muxatmd(622642)(1)
     1.1672       1.1672       0.0000       1.1483       1.1483       0.0000  getty(639088)(1)
     0.3353       0.0791       0.2562       0.3299       0.0778       0.2520  ksh(655440)(1)
     0.3098       0.0958       0.2139       0.3048       0.0943       0.2105  ksh(659476)(1)
     0.1192       0.0084       0.1108       0.1172       0.0083       0.1090  (598252)(1)
     0.0852       0.0333       0.0520       0.0839       0.0327       0.0511  rmcd(593968)(1)
     0.0295       0.0113       0.0182       0.0291       0.0111       0.0179
snmpmibd64(630858)(1)
     0.0094       0.0067       0.0028       0.0093       0.0065       0.0027  syncd(389354)(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. A sample of application summary by process type is shown in Example 4-46.

Example 4-46 Application summary by process type

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Application Summary (by process type)
                    -------------------------------------
    -- processing total (msec) --          -- percent of total processing time --
   combined  application      syscall     combined  application      syscall  name (thread
count)
   ========  ===========      =======     ========  ===========      =======  ==============
    42.6240      24.1758      18.4482      41.9332      23.7840      18.1492  (6)
     3.1864       1.1024       2.0840       3.1348       1.0846       2.0502  trcstop(1)
     1.2743       0.9171       0.3572       1.2537       0.9023       0.3514  muxatmd(1)
     1.1672       1.1672       0.0000       1.1483       1.1483       0.0000  getty(1)
     0.6451       0.1749       0.4701       0.6346       0.1721       0.4625  ksh(2)
     0.0852       0.0333       0.0520       0.0839       0.0327       0.0511  rmcd(1)
     0.0295       0.0113       0.0182       0.0291       0.0111       0.0179  snmpmibd64(1)
     0.0094       0.0067       0.0028       0.0093       0.0065       0.0027  syncd(1)

... lines omitted ...
```

Kproc summary

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 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

kernel The amount of CPU time, expressed in milliseconds, that the thread spent in kernel mode

operation The amount of CPU time, expressed in milliseconds, that the thread spent in operation 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

kernel The amount of CPU time that the thread spent in kernel mode, 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

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

A sample of kproc summary is shown in Example 4-47.

Example 4-47 Kproc summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Kproc Summary  (by Tid)
                    ----------------------
    -- processing total (msec) --          --   percent of total  time   --   name (Pid Tid
Type)
   combined       kernel    operation     combined       kernel    operation
   ========       ======  ===========     ========       ======  ===========
===================
     1.4096       1.4096       0.0000       1.3868       1.3868       0.0000  rtcmd(393466
1130587 -)
     1.2630       1.2630       0.0000       1.2425       1.2425       0.0000  wait(12294  12295
W)
     0.4387       0.4387       0.0000       0.4316       0.4316       0.0000  swapper(0  3 -)
     0.3051       0.3051       0.0000       0.3002       0.3002       0.0000  xmgc(364722
372919 -)
     0.2956       0.2956       0.0000       0.2908       0.2908       0.0000  gil(372918
401605 -)
     0.2712       0.2712       0.0000       0.2668       0.2668       0.0000  wait(16392  16393
W)
     0.2627       0.2627       0.0000       0.2585       0.2585       0.0000  gil(372918
393409 -)
     0.2358       0.2358       0.0000       0.2320       0.2320       0.0000  gil(372918
405703 -)
```

```
     0.2337       0.2337       0.0000       0.2299       0.2299       0.0000  gil(372918
389311 -)
     0.1018       0.1018       0.0000       0.1001       0.1001       0.0000  rpc.lockd(581668
761981 -)
     0.0517       0.0517       0.0000       0.0508       0.0508       0.0000  rpc.lockd(581668
770175 -)
     0.0163       0.0163       0.0000       0.0160       0.0160       0.0000  netm(368820
381115 -)
     0.0065       0.0065       0.0000       0.0064       0.0064       0.0000  rpc.lockd(581668
778441 -)
... lins omitted ...

                    Kproc Types
                    -----------
 Type Function                       Operation
 ==== ============================   ========================
  W   idle thread                    -
  N   NFS daemon                     NFS Remote Procedure Calls

... lines omitted ...
```

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 4-48 on page 241. The list is sorted by the total time in milliseconds consumed by each type of system call. 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

Example 4-48 System Calls Summary

```
[node6][/]> curt -i /tmp/trcrpt.r -m /tmp/trcnm.out | more
... skip ...
                    System Calls Summary
                    --------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  SVC (Address)
               (msec)    time    (msec)    (msec)    (msec)
========  ===========  ======  ========  ========  ========  ================
    1014       4.7623   0.05%    0.0047    0.0044    0.0106  _nsleep(1456048)
       2       1.9362   0.02%    0.9681    0.8623    1.0740  execve(1452280)
     103       0.9178   0.01%    0.0089    0.0021    0.0551  _select(145f4d8)
      44       0.6808   0.01%    0.0155    0.0088    0.0371  access(1461d40)
     149       0.6127   0.01%    0.0041    0.0014    0.0188  kread(145b578)
      38       0.6057   0.01%    0.0159    0.0009    0.1054  __loadx(1462190)
       2       0.5078   0.01%    0.2539    0.1355    0.3723  _exit(14552e0)
       2       0.5030   0.01%    0.2515    0.2384    0.2646  kfork(1452260)
     198       0.2872   0.00%    0.0015    0.0007    0.0568  kioctl(145e218)
       2       0.2353   0.00%    0.1177    0.0032    0.2322  _setpgid(145b818)
      56       0.2320   0.00%    0.0041    0.0037    0.0068  thread_waitact(1455c40)
      19       0.2203   0.00%    0.0116    0.0077    0.0287  statx(145e590)
      19       0.2168   0.00%    0.0114    0.0080    0.0294  open(145e4a0)
... lines omitted ...
```

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 4-49 displays the pending system calls summary. 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.

Example 4-49 Pending System Calls Summary

```
[node6][/]> curt -i /tmp/trcrpt.r -m /tmp/trcnm.out | more
... skip ...
                    Pending System Calls Summary
                    ----------------------------
Accumulated   SVC (Address)               Procname (Pid  Tid)
Time (msec)
============  ========================  ========================
```

```
      0.0169  (unknown)(3e96790)          X(94296 106609)
      0.0037  _nsleep(1456048)           nimesis(503816 180277)
      0.0030  thread_waitact(1455c40)     syncd(61588 213097)
      0.0089  _select(145f4d8)            prngd(147566 282801)
      0.0043  kwaitpid(1455310)           ksh(323638 356359)
      0.0051  _select(145f4d8)            aixmibd(221292 377017)
      0.0034  _nsleep(1456048)           nimesis(503816 589859)
      0.0034  _nsleep(1456048)           nimesis(503816 598231)
      0.0035  _nsleep(1456048)           nimesis(503816 602177)
      0.0031  _nsleep(1456048)           java(352468 614515)
      0.0035  _nsleep(1456048)           nimesis(503816 663625)
      0.0037  _nsleep(1456048)           java(409634 667895)
... lines omitted ...
```

Hypervisor calls summary

Hypervisor calls summary sections is a new section beginning with AIX 5L Version 5.3. If there is hypervisor activity in the trace, an additional section is inserted at this point of the report. This major section of the report is called Hypervisor Calls Summary. This section summarizes the processing time spent in hypervisor calls. A sample of Hypervisor calls summary is shown in Example 4-50.

Example 4-50 Hypervisor Calls Summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Hypervisor Calls Summary
                    ------------------------
   Count   Total Time   % sys  Avg Time  Min Time  Max Time  HCALL (Address)
               (msec)    time    (msec)    (msec)    (msec)
========  ===========  ======  ========  ========  ========  =================
     665       1.2927   1.27%    0.0019    0.0016    0.0033  H_XIRR(3a86be0)
     418       0.8637   0.85%    0.0021    0.0012    0.0035  H_CEDE(9032)
    1021       0.8120   0.80%    0.0008    0.0004    0.0097  H_PUT_TCE(3a9a090)
    1283       0.6963   0.69%    0.0005    0.0005    0.0014  H_PUT_TCE(3aa4060)
     176       0.1420   0.14%    0.0008    0.0007    0.0013  H_EOI(3a83d38)
     130       0.1352   0.13%    0.0010    0.0006    0.0016  H_PROD(2922c)
      35       0.0642   0.06%    0.0018    0.0011    0.0022  H_CEDE(3a86be0)
       3       0.0057   0.01%    0.0019    0.0019    0.0020  H_CEDE(1032)
       2       0.0044   0.00%    0.0022    0.0021    0.0023  H_CEDE(3aa4060)
... lines omitted ...
```

System NFS calls summary

NFS calls summary section is a new section beginning with AIX 5L Version 5.3. This section summarizes the processing time spent in NFS operations. For each NFS operation, identified by operation name and NFS version, the summary

gives the number of times the operation was called and the total processor time for all calls in milliseconds and as a percentage of total NFS operation time for all operations with the same NFS version. In addition, the summary gives the average, minimum and maximum times for one call to the operation.The System NFS Calls Summary is followed by the Pending NFS Calls Summary. This part lists the NFS calls which have started but not completed. A sample of system NFS calls summary is shown in Example 4-51.

Example 4-51 System NFS calls summary

```
[node6][/]> curt -i /tmp/trcrpt.r -m /tmp/trcnm.out | more
... skip ...
                    System NFS Calls Summary
                    ------------------------
   Count   Total Time  Avg Time  Min Time  Max Time  % Tot  % Tot  Opcode
               (msec)    (msec)    (msec)    (msec)   Time  Count
========  ===========  ========  ========  ========  =====  =====  =============
     449       9.1109    0.0203    0.0181    0.0256  100.00  100.00  RFS3_GETATTR
--------  -----------  --------  --------  --------  -----  -----  -------------
     449       9.1109    0.0203                                     NFS V3 TOTAL


                    Pending NFS Calls Summary
                    -------------------------
Accumulated   Sequence Number  Procname (Pid  Tid)
Time (msec)
============  ===============  =========================

... lines omitted ...
```

FLIH summary

This section lists all first level interrupt handlers that were called during the monitoring period. 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. A sample of FLIH Summary is shown in Example 4-52 on page 244. 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.

Example 4-52 FLIH Summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Global Flih Summary
                    -------------------
       Count  Total Time    Avg Time    Min Time    Max Time  Flih Type
                  (msec)      (msec)      (msec)      (msec)
      ====== =========== =========== =========== ===========  =========
         489      0.3754      0.0008      0.0007      0.0011    9(PHANTOM)
         317      0.6528      0.0021      0.0010      0.1666   32(QUEUED_INTR)
         876      4.1670      0.0048      0.0004      0.0115   31(DECR_INTR)
        1126      2.0283      0.0018      0.0006      0.0152    3(DATA_ACC_PG_FLT)
         176      3.2764      0.0186      0.0011      0.1947    5(IO_INTR)

                    Per CPU Flih Summary
                    --------------------

CPU Number 1:
       Count  Total Time    Avg Time    Min Time    Max Time  Flih Type
                  (msec)      (msec)      (msec)      (msec)
      ====== =========== =========== =========== ===========  =========
         139      0.2229      0.0016      0.0010      0.0420   32(QUEUED_INTR)
         174      1.0495      0.0060      0.0004      0.0115   31(DECR_INTR)
         520      0.7246      0.0014      0.0006      0.0145    3(DATA_ACC_PG_FLT)
         136      1.9101      0.0140      0.0017      0.1947    5(IO_INTR)

... lines omitted ...
```

SLIH summary

This section lists all second level interrupt handlers that were called during the monitoring period. The Global Slih Summary lists the total of second level interrupts on the system, while the Per CPU Slih Summary lists the second level interrupts per CPU. A sample of SLIH Summary is shown in Example 4-53 on page 245. 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.

Example 4-53 SLIH summary

```
r33n05:/ # more /tmp/curt.out
... skip ...
                    Global Slih Summary
                    -------------------
       Count  Total Time    Avg Time    Min Time    Max Time Slih Name(Address)
                  (msec)      (msec)      (msec)      (msec)
      ====== =========== =========== =========== =========== ================
          22      0.6328      0.0288      0.0053      0.2561 sisscsi_dd[sisscsi_dd64](3ae8128)
         154      2.9742      0.0193      0.0084      0.4301 goentdd[goentdd64](3bfba20)

                    Per CPU Slih Summary
                    --------------------

CPU Number 1:
       Count  Total Time    Avg Time    Min Time    Max Time Slih Name(Address)
                  (msec)      (msec)      (msec)      (msec)
      ====== =========== =========== =========== =========== ================
          14      0.2305      0.0165      0.0053      0.0234 sisscsi_dd[sisscsi_dd64](3ae8128)
         122      2.0994      0.0172      0.0084      0.4301 goentdd[goentdd64](3bfba20)


... lines omitted ...
```

Useful combinations

- **curt -i [input_file] -o [output_file]**
- **curt -i [input_file] -n [namefile] -m [namefile] -o [output_file]**

4.2.11 The splat command

The Simple Performance Lock Analysis Tool (splat) is a software tool that provides kernel and pthread lock usage reports. 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.

Syntax

splat **-i file** [**-n file**] [**-o file**] [**-d [bfta]**] [-l address] [-c class] [**-s [acelmsS]**] [-C cpus] [**-S count**] [-t start] [-T stop] [-p]

splat -j

Flags

-i inputfile	AIX trace file (REQUIRED).
-n namefile	File containing output of **gensyms** command.
-o outputfile	File to write reports to (DEFAULT: stdout).
-d detail	Detail information. Following parameter is supported.
[b]asic	summary and lock detail (DEFAULT)
[f]unction	basic + function detail
[t]hread	basic + thread detail
[a]ll	basic + function + thread detail
-s criteria	Sort the lock, function, and thread reports criteria. Following parameter is supported.
a	acquisitions
c	percent CPU hold time
e	percent elapsed hold time
l	lock address, function address, or thread ID
m	miss rate
s	spin count
S	percent CPU spin hold time (DEFAULT)
w	percent real wait time
W	average waitq depth
-S	count The maximum number of entries in each report (DEFAULT: 10).

Examples

Splat takes as primary input an AIX trace file which has been collected with the AIX trace command. Before analyzing a trace with splat, you will need to make sure that the trace is collected with an adequate set of hooks. The trace file which

is gathered using the trace command should contain at least followings trace events.

- HKWD_KERN_DISPATCH (106)
- HKWD_KERN_IDLE (10C)
- HKWD_KERN_RELOCK (10E)
- HKWD_KERN_LOCK (112)
- HKWD_KERN_UNLOCK (113)
- HKWD_SYSC_EXECVE (134)
- HKWD_SYSC_FORK (139)
- HKWD_CPU_PREEMPT (419)
- HKWD_SYSC_CRTHREAD (465)
- HKWD_KERN_WAITLOCK (46D)
- HKWD_KERN_WAKEUPLOCK (46E)
- HKWD_PTHREAD_CONDS (606)
- HKWD_PTHREAD_MUTEX (607)
- HKWD_PTHREAD_RWLOCK (608)
- HKWD_PTHREAD_GENERAL (609)

Trace event-group “splat“ contains these events. Example 4-54 shows event list of splat event group.

Example 4-54 Event list of splat event-group

```
r33n05:/ # trcevgrp -l splat
splat - Hooks for SPLAT performance tool (reserved)
        106,10C,10E,112,113,134,139,465,46D,46E,606,607,608,609,419
r33n05:/ #
```

Creating a splat report

To generate the **splat** report from a trace file, use **-i** flag to specify the trace file. Example 4-55 shows a sample of generating a splat report.

Example 4-55 Creating splat report

```
r33n05:/ # trace -J splat -a -o /tmp/trace.out
r33n05:/ # gennames > /tmp/gennames.out
r33n05:/ # splat -sa -da -S100 -i /tmp/trace.out -n /tmp/gennames.out -o /tmp/splat.out
```

Execution summary

The execution summary section contains following information.

- The command used to run splat.
- The command used to collect the system 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.

Example 4-56 Execution summary

```
r33n05:/tmp # more /tmp/splat.out

splat Cmd:      splat -sa -da -S100 -i /tmp/trace.out -n /tmp/gennames.out -o /tmp/splat.out

Trace Cmd:   trace -J splat -a -o /tmp/trace.out
Trace Host:  r33n05 (00C3E3CC4C00) AIX 5.3
Trace Date:  Fri Oct 22 18:09:01 2004


Elapsed Real Time:       4.421330
Number of CPUs Traced:   80           (Observed):3
Cumulative CPU Time:     353.706403


                                             start                  stop
                                      --------------------   --------------------
trace interval     (absolute tics)        515696762799603        515697815076151
                   (relative tics)                      0             1052276548
                   (absolute secs)          2166793.121007         2166797.542337
                   (relative secs)               0.000000               4.421330
analysis interval  (absolute tics)        515696762799603        515697815076151
                   (trace-relative tics)                0             1052276548
                   (self-relative tics)                 0             1052276548
                   (absolute secs)          2166793.121007         2166797.542337
                   (trace-relative secs)         0.000000               4.421330
```

```
                (self-relative secs)          0.000000          4.421330

... lines omitted ...
```

Gross lock summary

Example 4-57 on page 250 shows a sample of the gross lock summary report. The gross lock summary report section contains following information.

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 acquired during the analysis interval; for PThread condition-variables, the total number of times the condition passed 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 cumulative CPU time, 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 (*).

Example 4-57 Gross lock summary

```
r33n05:/tmp # more /tmp/splat.out
... skip...
                                  Unique       Acquisitions      Acq. or Passes      % Total
System
                       Total      Addresses     (or Passes)         per Second          'spin'
Time
                     ---------  -------------   ------------      --------------
---------------
AIX (all) Locks:        1139           1139           17444          3945.4191          0.000007
           RunQ:           0              0               0             0.0000          0.000000
         Simple:        1088           1088           16916          3825.9980          0.000007
    Transformed:           0              0               0             0.0000
         Krlock:           0              0               0             0.0000          0.000000
        Complex:          51             51             528           119.4211          0.000000
PThread CondVar:           0              0               0             0.0000          0.000000
          Mutex:           0              0               0             0.0000          0.000000
         RWLock:           0              0               0             0.0000          0.000000

... lines omitted ...
```

Per-lock summary

Example 4-58 on page 251 shows a sample of the per-lock summary report. The per-lock summary section contains following information.

Lock The name, lock class or address of the lock.

Type The type of the lock, identified by one of the following letters:

- **Q** A RunQ lock
- **S** A simple kernel lock
- **D** A disabled simple kernel lock
- **C** A complex kernel lock
- **M** A PThread mutex
- **V** A PThread condition-variable
- **L** A PThread read/write lock

Acquisitions The number of successful lock attempts for this lock, minus the number of times a thread was preempted while holding this lock.

Spins The number of unsuccessful lock attempts for this lock, minus the number of times a thread was undispatched while spinning.

Wait or Transform The number of unsuccessful lock attempts that resulted in the attempting thread going to sleep to wait for the lock to become available, or allocating a krlock.

%Miss Spins divided by Acquisitions plus Spins, multiplied by 100.

%Total Acquisitions divided by the total number of all lock acquisitions, multiplied by 100.

Locks/CSec Acquisitions divided by the combined elapsed duration in seconds.

Real CPU The percent of combined elapsed trace time that threads held the lock in question while dispatched. DISPATCHED_HOLDTIME_IN_SECONDS divided by combined trace duration, multiplied by 100.

Real Elaps(ed) The percent of combined elapsed trace time that threads held the lock while dispatched or sleeping. UNDISPATCHED_AND_DISPATCHED_HOLDTIME_IN_SECONDS divided by combined trace duration, multiplied by 100.

Comb Spin The percent of combined elapsed trace time that threads spun while waiting to acquire this lock. SPIN_HOLDTIME_IN_SECONDS divided by combined trace duration, multiplied by 100.

Example 4-58 Per-lock summary report

```
r33n05:/ # more /tmp/splat.out
...skip ...

100 max entries, Summary sorted by Acquisitions:

                                T  Acqui-          Wait
                                y  sitions          or                      Locks or
Percent Holdtime
                               p    or           Trans-                    Passes    Real
Real     Comb
 Lock Name, Class, or Address     e  Passes  Spins   form   %Miss   %Total    / CSec     CPU
Elapse    Spin
********************************** *  ******* ****** ****** ******* ********  *********
******** ******** ********
              000000000101CDD8    D    1548      0      0   0.0000   8.8741      4.377
0.0002   0.0182   0.0000
              F100060004289A58    D    1088      0      0   0.0000   6.2371      3.076
0.0001   0.0083   0.0000
              F1000600041BB0B0    D     710      0      0   0.0000   4.0702      2.007
0.0002   0.0130   0.0000
```

```
              F1000588D000BD30    D     665      0      0   0.0000   3.8122      1.880
0.0002   0.0127   0.0000
              F1000610003FF0B8    D     660      0      0   0.0000   3.7835      1.866
0.0001   0.0045   0.0000
              F1000108C045FCB8    D     658      0      0   0.0000   3.7721      1.860
0.0015   0.1201   0.0000
              000000000101BDD8    D     463      0      0   0.0000   2.6542      1.309
0.0001   0.0057   0.0000
              F1000600151962C8    D     440      0      0   0.0000   2.5224      1.244
0.0002   0.0123   0.0000
              0000000003BA67A0    D     440      0      0   0.0000   2.5224      1.244
0.0000   0.0031   0.0000
              0000000003B70BB0    D     440      0      0   0.0000   2.5224      1.244
0.0004   0.0304   0.0000
              00000000028C5AB0    D     440      0      0   0.0000   2.5224      1.244
0.0001   0.0107   0.0000
              F1000108C045EF38    D     351      0      0   0.0000   2.0122      0.992
0.0002   0.0147   0.0000
              F1000588D0000430    D     279      0      0   0.0000   1.5994      0.789
0.0001   0.0051   0.0000

... lines omitted ...
```

AIX kernel lock details

By default, splat prints out a lock detail report for each entry in the summary report. Example 4-59 shows a sample of kernel lock detail in lock detail report.

Example 4-59 kernel lock detail report

```
r33n05:/ # more /tmp/splat.out
...skip ...


[AIX SIMPLE Lock]                ADDRESS: 000000000101CDD8          KEX: unix
================================================================================
         |                Trans-      |                 | Percent Held (  4.421330s )
Type:    |  Miss  Spin    form   Busy  |    Secs Held    |  Real  Real    Comb   Real
Disabled |  Rate  Count   Count  Count |CPU      Elapsed |  CPU  Elapsed  Spin   Wait
         |  0.000 0       0      0     |0.000803 0.000803|  0.00   0.02   0.00   0.00
--------------------------------------------------------------------------------
Total Acquisitions:  1548     |SpinQ   Min   Max   Avg  |Krlocks SpinQ Min   Max   Avg
Acq. holding krlock: 0        |Depth   0     0     0    |Depth         0     0     0
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
PROD     |                          CONFER                              | HANDOFF
0        | SELF: 0         TARGET: 0                    ALL: 0          | 0
         |          w/ preemption: 0           w/ preemption: 0         |
```

```
--------------------------------------------------------------------------------

                    Lock Activity (mSecs) - Interrupts Disabled

  SIMPLE      Count         Minimum        Maximum        Average          Total
  +++++++     ++++++  +++++++++++++++ +++++++++++++++ +++++++++++++++ +++++++++++++++
  LOCK          1548        0.000328       0.001853       0.000518       0.802521
   w/ KRLOCK       0        0.000000       0.000000       0.000000       0.000000
  SPIN             0        0.000000       0.000000       0.000000       0.000000
  KRLOCK LOCK      0        0.000000       0.000000       0.000000       0.000000
  KRLOCK SPIN      0        0.000000       0.000000       0.000000       0.000000
  TRANSFORM        0        0.000000       0.000000       0.000000       0.000000
... lines omitted ...
```

Function detail report

Example 4-60 is an example of the function detail report. This report is obtained by using **splat** with the **-df** or **-da** options.

Example 4-60 Function detail report

```
r33n05:/ # more /tmp/splat.out
...skip ...

                      Acqui-    Miss  Spin  Transf. Busy    Percent Held of Total Time
  Function Name       sitions    Rate  Count  Count  Count    CPU   Elapse  Spin  Transf.
Return Address   Start Address    Offset
 ^^^^^^^^^^^^^^^^^^^ ^^^^^^^^  ^^^^^^ ^^^^^^ ^^^^^^ ^^^^^^  ^^^^^^ ^^^^^^ ^^^^^^ ^^^^^^
^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^ ^^^^^^^^
             .tstart      499    0.00      0      0      0    0.00   0.01   0.00   0.00
0000000000060D48 0000000000000000 00060D48
              .clock      497    0.00      0      0      0    0.00   0.00   0.00   0.00
0000000000060228 0000000000000000 00060228
              .clock      496    0.00      0      0      0    0.00   0.01   0.00   0.00
0000000000060110 0000000000000000 00060110
           .sys_timer       44    0.00      0      0      0    0.00   0.00   0.00   0.00
000000000005FD10 0000000000000000 0005FD10
              .tstop        7    0.00      0      0      0    0.00   0.00   0.00   0.00
0000000000060AB8 0000000000000000 00060AB8
          .tstop_fast        4    0.00      0      0      0    0.00   0.00   0.00   0.00
000000000005EB28 0000000000000000 0005EB28
         .incinterval        1    0.00      0      0      0    0.00   0.00   0.00   0.00
00000000000FCDE0 0000000000000000 000FCDE0

... lines omitted ...
```

Thread detail report

Example 4-61 shows an example of the thread detail report. This report is obtained by using the -dt or -da options of splat command.

Example 4-61 Thread detail report

```
r33n05:/ # more /tmp/splat.out
...skip ...
           Acqui-    Miss   Spin  Transf. Busy    Percent Held of Total Time
Process
 ThreadID  sitions   Rate   Count  Count  Count    CPU   Elapse  Spin Transf. ProcessID  Name
~~~~~~~~  ~~~~~~~~  ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~  ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~ ~~~~~~~~~
~~~~~~~~~~~~~~~

    16393      1523    0.00      0      0      0    0.02   0.02   0.00   0.00     16392  wait
   409619         8    0.00      0      0      0    0.00   0.00   0.00   0.00    389354  syncd
   381115         6    0.00      0      0      0    0.00   0.00   0.00   0.00    368820  netm
   622683         3    0.00      0      0      0    0.00   0.00   0.00   0.00    442370
syslogd
        3         3    0.00      0      0      0    0.00   0.00   0.00   0.00    639088
swapper
   561177         2    0.00      0      0      0    0.00   0.00   0.00   0.00    389354  syncd
   372919         2    0.00      0      0      0    0.00   0.00   0.00   0.00    364722  xmgc
  1146957         1    0.00      0      0      0    0.00   0.00   0.00   0.00    507954

... lines omitted ...
```

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. This section has three part. Example 4-62 shows a sample of the top part of this section.

Example 4-62 Complex lock report

```
r33n05:/ # more /tmp/splat.out
...skip ...
[AIX COMPLEX Lock]                ADDRESS: F100060016230160          KEX: liblvm
================================================================================
         |                              |                  | Percent Held (  4.421330s )
Acqui-   |  Miss  Spin   Wait   Busy    |    Secs Held     |  Real  Real    Comb   Real
sitions  |  Rate  Count  Count  Count   |CPU      Elapsed  |  CPU  Elapsed  Spin   Wait
72       |  0.000 0      0      0       |0.000252 0.000252 |  0.00   0.01   0.00   0.00
--------------------------------------------------------------------------------
%Enabled  100.00 (      72)|SpinQ   Min   Max   Avg   | WaitQ    Min   Max   Avg
%Disabled   0.00 (       0)|Depth   0     0     0     | Depth    0     0     0
--------------------------|Readers 0     0     0     |Readers   0     0     0
          Min   Max   Avg  |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    |
-------------------------------------------------------------------------------

... lines omitted ...
```

The lock activity report

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 4-63.

Example 4-63 Complex lock report - Lock activity

```
r33n05:/ # more /tmp/splat.out
...skip ...
                     Lock Activity w/Interrupts Enabled (mSecs)

  READ     Count         Minimum        Maximum        Average          Total
  +++++++  ++++++  +++++++++++++++ +++++++++++++++ +++++++++++++++ +++++++++++++++
  LOCK         70        0.002466       0.004618       0.003575       0.250218
  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          2        0.000429       0.000941       0.000685       0.001370
  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

... lines omitted ...
```

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 4-64.

Example 4-64 Complex lock report - function and thread detail

```
r33n05:/ # more /tmp/splat.out
...skip ...
                      Acquisitions    Miss    Spin Count    Wait Count   Busy    Percent Held
of Total Time
 Function Name        Write   Read    Rate   Write   Read  Write   Read  Count   CPU   Elapse
Spin   Wait    Return Address   Start
Address    Offset
 ^^^^^^^^^^^^^^^^^^^^ ^^^^^^ ^^^^^^  ^^^^^^ ^^^^^^ ^^^^^^ ^^^^^^ ^^^^^^ ^^^^^^  ^^^^^^ ^^^^^^
^^^^^^ ^^^^^^  ^^^^^^^^^^^^^^^^ ^^^^^^^
^^^^^^^^ ^^^^^^^^
             .j2_rdwr      0     70    0.00      0      0      0      0      0    0.00   0.01
0.00   0.00  00000000002FC384 0000000
000000000 002FC384
            .j2_close      1      0    0.00      0      0      0      0      0    0.00   0.00
0.00   0.00  0000000000367BD8 0000000
000000000 00367BD8
             .j2_open      1      0    0.00      0      0      0      0      0    0.00   0.00
0.00   0.00  00000000003055B4 0000000
000000000 003055B4

        Acquisitions  Miss   Spin Count   Wait Count  Busy   Percent Held of Total Time
Process
 ThreadID  Write  Read    Rate   Write  Read   Write  Read   Count     CPU  Elapse  Spin   Wait
ProcessID  Name
 ^^^^^^^^^  ^^^^^^^ ^^^^^^^  ^^^^^^^ ^^^^^^^ ^^^^^^^ ^^^^^^^ ^^^^^^^  ^^^^^^^ ^^^^^^^ ^^^^^^^
^^^^^^^ ^^^^^^^^^^  ^^^^^^^^^^^^^^^
  1146959       2     70    0.00      0      0      0      0      0    0.01   0.01   0.00
0.00    507956  trcstop

... lines omitted ...
```

Useful combinations

- **splat -sa -da -i [input_file] -n [name_file] -o [ouput_file]**

4.2.12 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 truss command resides in /usr/bin and is part of the bos.sysmgt.serv_aid fileset, which is installable from the AIX base installation media.

Syntax

truss [-f] [**-c**] [-a] [-l] [-d] [-D] [-e] [-i] [{ **-t** | -x} [!] Syscall [...]] [-s [!] Signal [...]] [{ -m }[!] Fault [...]] [{ -r | -w} [!] FileDescriptor [...]] [{ -u } [!]LibraryName [...]:: [!]FunctionName [...]] [**-o Outfile**] {Command| -p pid [. . .]}

Flags

-c Counts traced system calls, faults, and signals rather than displaying trace results line by line. A summary report is produced after the traced command terminates or when truss is interrupted. If the -f flag is also used, the counts include all traced Syscalls, Faults, and Signals for child processes.

-o Outfile Designates the file to be used for the trace output. By default, the output goes to standard error.

-t [!] Syscall Includes or excludes system calls from the trace process. System calls to be traced 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 trace output. The default is -tall.

Examples

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. Example 4-65 shows the flow of using the date command. You can see that after the program has been loaded and the initial setup has been performed, kioctl and kwrite system calls are used in this program.

Example 4-65 Truss command without any flag

```
r33n05:/ # truss date
execve("/usr/bin/date", 0x2FF2294C, 0x2FF22954)  argc: 1
sbrk(0x00000000)                                 = 0x20000F94
sbrk(0x0000000C)                                 = 0x20000F94
__libc_sbrk(0x00000000)                          = 0x20000FA0
getuidx(4)                                       = 0
getuidx(2)                                       = 0
getuidx(1)                                       = 0
getgidx(4)                                       = 0
getgidx(2)                                       = 0
getgidx(1)                                       = 0
```

```
__loadx(0x01000080, 0x2FF1E6E0, 0x00003E80, 0x2FF22680, 0x00000000) =
0xD0083130
__loadx(0x01000180, 0x2FF1E6E0, 0x00003E80, 0xF08B3A94, 0xF08B39C4) =
0xF09E8B78
__loadx(0x07080000, 0xF08B3A64, 0xFFFFFFFF, 0xF09E8B78, 0x00000000) =
0xF09E9A98
... slip...

access("/usr/lib/nls/msg/en_US/date.cat", 0)    = 0
_getpid()                                       = 561398
kioctl(1, 22528, 0x00000000, 0x00000000)        = 0
Sun Oct 24 15:26:29 EDT 2004
kwrite(1, " S u n   O c t   2 4   1"..., 29)     = 29
kfcntl(1, F_GETFL, 0x2FF22FFC)                  = 2
kfcntl(2, F_GETFL, 0xF09148D0)                  = 2
_exit(0)
r33n05:/ #
```

Truss summary report

To get a truss summary report, use the -c flag with the truss command. Example 4-66 shows a samples of truss summary report. In this example, we collect summary of date command.

Example 4-66 Truss summary report

```
r33n05:/ # truss -c -o /tmp/truss_c.out date
Sun Oct 24 15:03:00 EDT 2004
r33n05:/ # more /tmp/truss_c.out
syscall               seconds   calls  errors
execve                    .00       1
_exit                     .00       1
kwrite                    .00       1
_getpid                   .00       1
getuidx                   .00      18
kioctl                    .00       1
getgidx                   .00      18
__libc_sbrk               .00       1
sbrk                      .00       2
access                    .00       1
kfcntl                    .00       2
__loadx                   .00      17
                         ----     ---    ---
sys totals:               .00      64      0
usr time:                 .00
elapsed:                  .00
r33n05:/ #
```

Monitoring specified system call

To monitor specified system call, use the -t flag with the truss command. Example 4-67 shows a sample of monitoring specified system call. In this example, the truss command monitor only open() system call.

Example 4-67 Monitoring specified system call

```
r33n05:/ # truss -t open -o /tmp/truss.test perfwb > /dev/null
r33n05:/ # more  /tmp/truss.test
open("/usr/lib/nls/msg/en_US/ksh.cat", O_RDONLY) = 3
open(".kshrc", O_RDONLY)                         = 3
open("/usr/bin/perfwb", O_RDONLY)                = 3
r33n05:/ #
```

Useful combinations

- **truss command -o [ouput_file]**
- **truss -c command -o [ouput_file]**
- **truss -t Systemcall -o [ouput_file] command**

4.2.13 The gprof command

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. The **gprof** command is in fact a subset of the **prof** command.

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.

Syntax

gprof [-b] [-e Name] [-E Name] [-f Name] [-g filename] [-i filename] [-p filename] [-F Name] [-L PathName] [-s] [-z] [a.out [gmon.out ...]]

Examples

To use **gprof**, we need to make binary code and gmon.out file. Example 4-68 on page 260 shows a sample of preparing **gprof** and using **gprof** command. The first step is to compile the C source code into a binary code using **-pg** flag. Next step is to run binary code. Then gmon.out file is created. The **gprof** command makes execution profile of this program.

Example 4-68 Using gprof command

```
r33n05:/home/kumiko/src # cc -pg -o memtest memtest.c
r33n05:/home/kumiko/src # ./memtest > /dev/null
r33n05:/home/kumiko/src # ls -l gmon.out
-rw-r--r--   1 root     system       933968 Oct 24 15:36 gmon.out
r33n05:/home/kumiko/src # gprof memtest > gprof.out
```

Detailed function report

Example 4-69 shows a sample of detail function report. This reports the functions sorted according to the time they represent, including the time of their call-graph descendents.

Example 4-69 Detailed function report

```
r33n05:/home/kumiko/src # more gprof.out
... skip ...
                                     called/total       parents
index  %time    self descendents  called+self    name           index
                                     called/total       children

                0.00        0.00    2004/2004        .printf [2]
[1]      0.0    0.00        0.00    2004          ._doprnt [1]
                0.00        0.00       1/1           .fwrite [22]

-----------------------------------------------

                0.00        0.00    2004/2004        .main [26]
[2]      0.0    0.00        0.00    2004          .printf [2]
                0.00        0.00    2004/2004        ._doprnt [1]

-----------------------------------------------

                0.00        0.00    2001/2001        .fflush_unlocked [6]
[3]      0.0    0.00        0.00    2001          ._xflsbuf [3]
                0.00        0.00    2001/2001        ._xwrite [4]

-----------------------------------------------

                0.00        0.00    2001/2001        ._xflsbuf [3]

... lines omitted ...
```

Flat profile

Example 4-70 on page 261 shows a sample of the flat profile of **gprof** command.

Example 4-70 Flat profile

```
r33n05:/home/kumiko/src # gprof memtest > gprof.out
... skip ...

  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
  0.0       0.00     0.00     2004     0.00     0.00  ._doprnt [1]
  0.0       0.00     0.00     2004     0.00     0.00  .printf [2]
  0.0       0.00     0.00     2001     0.00     0.00  ._xflsbuf [3]
  0.0       0.00     0.00     2001     0.00     0.00  ._xwrite [4]
  0.0       0.00     0.00     2001     0.00     0.00  .fflush [5]
  0.0       0.00     0.00     2001     0.00     0.00  .fflush_unlocked [6]
  0.0       0.00     0.00     2001     0.00     0.00  .write [7]
  0.0       0.00     0.00        3     0.00     0.00  .splay [8]
  0.0       0.00     0.00        2     0.00     0.00  .free [9]
  0.0       0.00     0.00        2     0.00     0.00  .free_48_44 [10]
  0.0       0.00     0.00        2     0.00     0.00  .free_common [11]
  0.0       0.00     0.00        2     0.00     0.00  .free_y [12]
  0.0       0.00     0.00        2     0.00     0.00  .leftmost [13]
  0.0       0.00     0.00        2     0.00     0.00  .malloc_common [14]
  0.0       0.00     0.00        2     0.00     0.00  .malloc_common_50_34 [15]
  0.0       0.00     0.00        2     0.00     0.00  .malloc_y [16]
  0.0       0.00     0.00        1     0.00     0.00  .__ioctl [17]
  0.0       0.00     0.00        1     0.00     0.00  ._findbuf [18]
  0.0       0.00     0.00        1     0.00     0.00  ._wrtchk [19]

... lines omitted ...
```

Listing of cross references

A cross-reference index, as shown in Example 4-71 is the last item produced summarizing the cross references found during profiling.This report is an alphabetical listing of the cross references found during profiling.

Example 4-71 Listing of cross references

```
r33n05:/home/kumiko/src # gprof memtest > gprof.out
... skip ...

Index by function name

  [17] .__ioctl             [10] .free_48_44          [16] .malloc_y
   [1] ._doprnt             [11] .free_common         [27] .moncontrol
  [18] ._findbuf            [12] .free_y              [28] .monitor
  [19] ._wrtchk             [22] .fwrite              [29] .nsleep
   [3] ._xflsbuf            [23] .fwrite_unlocked     [30] .pre_ioctl
   [4] ._xwrite             [24] .ioctl                [2] .printf
  [20] .exit                [25] .isatty              [31] .rightmost
  [21] .extend_brk          [13] .leftmost            [32] .sleep
```

```
   [5] .fflush              [26] .main                [8] .splay
   [6] .fflush_unlocked     [14] .malloc_common       [7] .write
   [9] .free                [15] .malloc_common_50_3

... lines omitted ...
```

Useful combination

- **gprof [binary_code] > [ouput_file]**

4.2.14 The pprof command

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.

Syntax

pprof { **time** | -I pprof.flow | -i tracefile | -d } [-T bytes] [-v] [-s] [-n] [-f] [-p] [-w]

Paramters

time Specifies the number of seconds to trace the system.

Examples

Creating a pprof report

The **pprof** command reports on all kernel threads running within an interval using the trace utility. This information is saved in the file files. Example 4-72 shows a sample of creating **pprof** output. In this example, **pprof** reports CPU usage of all kernel threads for 60 seconds.

Example 4-72 Creating pprof report

```
r33n05:/home/kumiko/pprof # pprof 60
Sun Oct 24 16:00:15 2004
System: AIX r33n05 Node: 5 Machine: 00C3E3CC4C00

r33n05:/home/kumiko/pprof #
*** PPROF COMPLETED ***

r33n05:/home/kumiko/pprof # ls -l
total 72
-rw-r--r--   1 root     system         7004 Oct 24 16:01 pprof.cpu
-rw-r--r--   1 root     system         2061 Oct 24 16:01 pprof.famcpu
```

```
-rw-r--r--   1 root     system         5465 Oct 24 16:01 pprof.famind
-rw-r--r--   1 root     system         3072 Oct 24 16:01 pprof.flow
-rw-r--r--   1 root     system         1599 Oct 24 16:01 pprof.namecpu
-rw-r--r--   1 root     system         7005 Oct 24 16:01 pprof.start
r33n05:/home/kumiko/pprof #
```

The pprof.cpu report

Example 4-75 on page 265 shows a sample of pprof.cpu file. This file contains all kernel-level threads sorted by actual CPU time.

- Process Name (Pname)
- Process ID (PID)
- Parent Process ID (PPID)
- Process State at Beginning and End (BE)
- Thread ID (TID)
- Parent Thread ID (PTID)
- Actual CPU Time (ACC_time)
- Start Time (STT_time)
- Stop Time (STP_time)
- The difference between the Stop time and the Start time (STP_STT)

Example 4-73 The pprof.cpu report

```
r33n05:/home/kumiko/pprof # more pprof.cpu

                    Pprof CPU Report

                    Sorted  by  Actual CPU  Time

                    From: Sun Oct 24 16:00:15 2004
                    To:   Sun Oct 24 16:01:16 2004


        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
               =====    =====    ===== ===    =====    =====  ========  ========  ========
========
                wait    16392        0  AA    16393        0     0.066     0.000    60.968
60.968
               syncd   389354        1  AA   565273        0     0.022     1.884     2.166
0.282
               getty   639088        1  AA   774293        0     0.020     0.001    60.968
60.967
```

```
                wait    12294        0  AA    12295        0     0.017     0.379    60.972
60.593
               pprof   598092   401466  AA  1056959        0     0.013     1.822    60.088
58.266
            muxatmd   622642   651328  AA   798857        0     0.007     1.125    56.132
55.007
            swapper        0        0  AA        3        0     0.006     0.504    60.505
60.001

... lines omitted ...
```

The pprof.start report

Example 4-74 shows the pprof.start file. This file lists all kernel threads sorted by start time.

- Process Name (Pname)
- Process ID (PID)
- Parent Process ID (PPID)
- Process State Beginning and End (BE)
- Thread ID (TID)
- Parent Thread ID (PTID)
- Actual CPU Time (ACC_time)
- Start Time (STT_time)
- Stop Time (STP_time)
- The difference between the Stop time and the Start time (STP_STT)

Example 4-74 The pprof.start report

```
r33n05:/home/kumiko/pprof # more pprof.start


                  Pprof START TIME Report

                  Sorted  by  Start Time

                  From: Sun Oct 24 16:00:15 2004
                  To:   Sun Oct 24 16:01:16 2004


       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
               =====    =====    ===== ===    =====    =====  ========  ========  ========
========
```

```
               wait    16392        0  AA    16393        0     0.066     0.000    60.968
60.968
            UNKNOWN   569472       -1  EA  1196165        0     0.000     0.000     0.000
0.000
              trace   536820   401466  AX  1167547        0     0.001     0.000     0.000
0.000
            UNKNOWN   569472       -1  CA  1204321  1196165     0.001     0.000    60.088
60.087
            UNKNOWN   561244       -1  EA  1200233        0     0.000     0.000     0.000
0.000
              getty   639088        1  AA   774293        0     0.020     0.001    60.968
60.967
               netm   368820        0  AA   381115        0     0.000     0.078    60.079
60.000
... lines omitted ...
```

The pprof.namecpu report

Example 4-75 shows the pprof.namecpu file. This file lists information about each type of kernel thread.

- Process Name (Pname)
- Number of Threads (#ofThreads)
- CPU Time (CPU_Time)
- % of Total CPU Time (%)

Example 4-75 The pprof.namecpu report

```
r33n05:/home/kumiko/pprof # more pprof.namecpu


                Pprof  PROCESS  NAME  Report

                Sorted  by  CPU  Time

                From: Sun Oct 24 16:00:15 2004
                To:   Sun Oct 24 16:01:16 2004


           Pname #ofThreads  CPU_Time        %
        ======== ==========  ========  ========
            wait          2     0.083    45.917
           syncd         14     0.028    15.490
           getty          1     0.020    11.064
           pprof          4     0.013     7.192
             gil          4     0.012     6.639
         muxatmd          1     0.007     3.873
         swapper          1     0.006     3.319
       rpc.lockd         12     0.003     1.660
```

```
                xmgc          1     0.002     1.106
               trace          1     0.001     0.553
           snmpmibd64         1     0.001     0.553

... lines omitted ...
```

The pprof.famind report

Example 4-76 shows the pprof.famind file. This file lists all processes grouped by families.

- Start Time (STT)
- Stop Time (STP)
- Actual CPU Time (ACC)
- Process ID (PID)
- Parent Process ID (PPID)
- Thread ID (TID)
- Parent Thread ID (PTID)
- Process State at Beginning and End (BE)
- Level (LV)
- Process Name (PNAME)

Example 4-76 The pprof.famind report

```
r33n05:/home/kumiko/pprof # more pprof.famind

         Pprof PROCESS FAMILY Report - Indented

         Sorted by Family and Start Time

                   From: Sun Oct 24 16:00:15 2004
                   To:   Sun Oct 24 16:01:16 2004


        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
      =======  =======  =======  =====  =====  =====  =====  == == ==============

        0.504   60.505    0.006      0      0      3      0  AA  0      swapper
       47.327   47.327    0.000      1      0   4099      0  AA  0      init

        0.000   60.968    0.066  16392      0  16393      0  AA  2     .. wait

        0.000    0.000    0.000 569472     -1 1196165      0  EA  2     .. UNKNOWN
        0.000   60.088    0.001 569472     -1 1204321 1196165  CA  2     ..- UNKNOWN
```

```
      0.000    0.000    0.001 536820 401466 1167547      0  AX  2     .. trace

... lines omitted ...
```

The pprof.famcpu report

Example 4-77 shows the pprof.famcpu file. This file 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.

- Start Time (Stt-Time)
- Process Name (Pname)
- Process ID (PID)
- Number of Threads (#Threads)
- Total CPU Time (Tot-Time)

Example 4-77 The pprof.famcpu report

```
r33n05:/home/kumiko/pprof # more pprof.famcpu


                Pprof PROCESS FAMILY SUMMARY Report

                Sorted by CPU Time

                  From: Sun Oct 24 16:00:15 2004
                  To:   Sun Oct 24 16:01:16 2004



     Stt-Time                  Pname    PID #Threads Tot-Time
     ========    ====================  ===== ======== ========
       0.0000                   wait  16392        1    0.066
       1.8844                  syncd 389354       14    0.028
       0.0009                  getty 639088        1    0.020
       0.3790                   wait  12294        1    0.017
       0.0002                  trace 536820        5    0.015
       0.1790                    gil 372918        4    0.013
       1.1250                muxatmd 622642        1    0.007
       0.5039                swapper      0        1    0.006
       0.1110               rpc.lockd 581668       12    0.002
       1.6998                    xmgc 364722        1    0.002
       0.0002                 UNKNOWN 569472        2    0.001
       0.7567                    rmcd 593968        1    0.001
       2.8284              snmpmibd64 630858        1    0.001
      47.4474                    rgsr 422132        1    0.000
       0.0783                    netm 368820        1    0.000
      47.3268                    init      1        1    0.000
```

```
  1.8852                  j2pg 409812        5    0.000
 47.3267               UNKNOWN 536822        1    0.000
  1.6304                pilegc 360624        2    0.000
  0.5040                reaper 336036        1    0.000
 45.5198                  cron 577578        1    0.000
 52.1472              sendmail 643140        1    0.000
 48.6320               syslogd 442370        1    0.000
  1.6737                 rdpgc 467172        1    0.000
 18.1069                 nfsSM 463074        1    0.000
  0.0002               UNKNOWN 561244        1    0.000
                                         ======== ========
                                               63    0.181
r33n05:/home/kumiko/pprof #
```

Useful combination

- **pprof sleep [second]**

4.2.15 The prof command

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.

Syntax

prof [-t | -c | -a | -n] [-o | **-x**] [**-g**] [-z] [-h] [**-s**] [-S] [-v] [-L PathName] [Program] [-m MonitorData ...]

Flags

-x Displays each address in hexadecimal, along with the symbol name.

-g Includes non-global symbols (static functions).

-s Produces a summary file in mon.sum. This is useful when more than one profile file is specified.

Examples

To use prof, we need to make binary code and mon.out file. Example 4-78 on page 269 shows a sample of preparing prof and using **prof** command. The first step is to compile the source code into a binary using **-p** flag. Next step is to run binary code. Then mon.out file is created. The prof command makes execution profile of this program.

Example 4-78 Creating prof report

```
r33n05:/home/kumiko/src # cc -p -o memtest memtest.c
r33n05:/home/kumiko/src # ./memtest > /dev/null
r33n05:/home/kumiko/src # ls -l mon.out
-rw-r--r--   1 root     system       933750 Oct 24 16:19 mon.out
r33n05:/home/kumiko/src # prof -xg -s > prof.out
r33n05:/home/kumiko/src # ls -l prof.out
-rw-r--r--   1 root     system         2404 Oct 24 16:19 prof.out
r33n05:/home/kumiko/src #
```

The prof report

Example 4-79 shows a sample of prof report. 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.

Example 4-79 The prof report

```
r33n05:/home/kumiko/src # more prof.out

Address  Name                 %Time     Seconds     Cumsecs  #Calls   msec/call
10000558 .main                  0.0        0.00        0.00       1      0.0
d1d95e64 .monitor               0.0        0.00        0.00       1      0.0
d1d97188 .moncontrol            0.0        0.00        0.00       1      0.0
d1d9dc64 .free_common           0.0        0.00        0.00       2      0.0
d1d9debc .malloc_common         0.0        0.00        0.00       2      0.0
d1d9e2e4 .free                  0.0        0.00        0.00       2      0.0
d1d9e3ac .free_48_44            0.0        0.00        0.00       2      0.0
d1d9e4c8 .malloc_common_50_34   0.0        0.00        0.00       2      0.0
d1d9ecd0 .splay                 0.0        0.00        0.00       3      0.0
d1d9f25c .leftmost              0.0        0.00        0.00       2      0.0
d1d9f8b4 .free_y                0.0        0.00        0.00       2      0.0
```

```
d1da0c70 .malloc_y              0.0         0.00         0.00       2       0.0
d1da32ec .printf                0.0         0.00         0.00    2004       0.000
d1da366c ._doprnt               0.0         0.00         0.00    2004       0.000
d1dbc4f4 .__ioctl               0.0         0.00         0.00       1       0.0
d1dbc6c0 .pre_ioctl             0.0         0.00         0.00       1       0.0
d1dbc9e8 .ioctl                 0.0         0.00         0.00       1       0.0
d1dbe164 .fflush_unlocked       0.0         0.00         0.00    2001       0.000
d1dbe860 ._findbuf              0.0         0.00         0.00       1       0.0
d1dbeab0 ._xflsbuf              0.0         0.00         0.00    2001       0.000
d1dbebf0 ._xwrite               0.0         0.00         0.00    2001       0.000

... lines omitted ...
```

4.2.16 The tprof command

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

Syntax

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** } }

Flags

-e	Turns on kernel extension profiling.
-k	Enables kernel profiling.
-s	Enables shared library profiling.
-x Program	Specifies the program to be executed by **tprof**. Data collection stops when Program completes or trace is manually stopped with either **trcoff** or **trcstop**

Note: The -x flag must be the last flag in the list of flags specified in tprof.

Examples

For subroutine-level profiling, the **tprof** command can be run without modifying executable programs (that is, no re-compilation with special compiler flags is

necessary). This is still true if the executables have been striped, unless the back tables have also been removed. Example 4-80 shows a sample of creating tprof report for 60 seconds, and Example 4-81 shows a sample of displaying **tprof** report.

Example 4-80 Creating tprof report

```
r33n05:/home/kumiko/tprof # tprof -kes -x sleep 60
Sun Oct 24 16:34:10 2004
System: AIX 5.3 Node: r33n05 Machine: 00C3E3CC4C00
Starting Command sleep 60
stopping trace collection.
shmat: A file descriptor does not refer to an open file.
Generating sleep.prof
r33n05:/home/kumiko/tprof # ls -l sleep.prof
-rw-r--r--   1 root     system        14231 Oct 24 16:35 sleep.prof
```

Example 4-81 The tprof report

```
r33n05:/home/kumiko/tprof # more sleep.prof

Process                              Freq  Total Kernel   User Shared  Other
=======                              ====  ===== ======   ==== ======  =====
wait                                    4  92.47  92.47   0.00   0.00   0.00
/usr/java14/jre/bin/java                2   7.49   0.39   0.00   7.03   0.07
./memtest                               1   0.04   0.04   0.00   0.00   0.00
=======                              ====  ===== ======   ==== ======  =====
Total                                   7 100.00  92.90   0.00   7.03   0.07

Process                  PID      TID  Total Kernel   User Shared  Other
=======                  ===      ===  ===== ======   ==== ======  =====
wait                   16392    16393  24.05  24.05   0.00   0.00   0.00
wait                   12294    12295  24.01  24.01   0.00   0.00   0.00
wait                    8196     8197  22.22  22.22   0.00   0.00   0.00
wait                   20490    20491  22.19  22.19   0.00   0.00   0.00
/java14/jre/bin/java  503996  1212499   7.24   0.29   0.00   6.88   0.07
/java14/jre/bin/java  503994  1212497   0.25   0.11   0.00   0.14   0.00
./memtest             655552  1192035   0.04   0.04   0.00   0.00   0.00
=======                  ===      ===  ===== ======   ==== ======  =====
Total                                  100.00  92.90   0.00   7.03   0.07

        Total Samples = 2790    Total Elapsed Time = 61.97s

  Total % For All Processes (KERNEL) = 92.90

Subroutine                                          %   Source
==========                                       ====== ======
h_cede_end_point                                  92.47 hcalls.s
pcs_glue                                           0.07 vmvcs.s
```

```
.memset_overlay                                          0.07 64/low.s
.trchook64                                               0.07 trchka64.s
.pagerRdwrReadAhead                                      0.04 rnel/j2/j2_vcpager.c
.iMark                                                   0.04 kernel/j2/j2_inode.c
.vnop_seek                                               0.04 s/kernel/lfs/vnops.c
.memmove_overlay                                         0.04 64/low.s
.simple_lock                                             0.04 64/low.s
.v_inspft                                                0.04 kernel/vmm/v_lists.c

  Total % For All Processes (SH-LIBs) = 7.03

Shared Object                                                          %
=============                                                        ======
/usr/java14/jre/bin/libjitc.a                                         4.16
/usr/java14/jre/bin/classic/libjvm.a                                  2.11
/usr/java14/jre/bin/libzip.a                                          0.43
/usr/lib/libc.a[shr.o]                                                0.18
/usr/lib/libpthreads.a[shr_xpg5.o]                                    0.14

  Profile: /usr/java14/jre/bin/libjitc.a

  Total % For All Processes (/usr/java14/jre/bin/libjitc.a) = 4.16

Subroutine                                               %   Source
==========                                             ====== ======
.union_set                                               0.32 /Qopt/dfQ_fsescape.c
.Commoning_Init_Dataflow_B                               0.22 /dfQ_commoning_sub.c
.Commoning_Final_Dataflow_B                              0.14 /dfQ_commoning_sub.c
.Copypropa_Init_Dataflow                                 0.14 Qopt/dfQ_copypropa.c
.dopt_generate_dag                                       0.11 Dopt/dopt_quad2dag.c
.MERGE_VARREF                                            0.11 /util/jit_dataflow.c
.Deadstore_Final_Dataflow_V                              0.11 Qopt/dfQ_deadstore.c
._fill                                                   0.07 noname
.alloc_sets                                              0.07 /Qopt/dfQ_fsescape.c
.CostBenefitAnalysis                                     0.07 /dfQ_commoning_sub.c

... lines omitted ...
```

Useful combinations

- **tprof -x sleep [second]**
- **tprof -skeuj -x sleep [second]**
- **tprof -kes -x sleep [seconds]**
- **tprof -A -x sleep**

4.2.17 The time command

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.

The **time** command resides in /usr/bin and is part of the bos.rte.misc_cmds fileset, which is installable from the AIX base installation media.

Syntax

time [-p] **Command** [Argument ...]

Parameters

Command The command that will be timed by the time command.

Examples

The time command simply counts the CPU ticks from when the command that was entered as an argument is started until that command completes. Example 4-82 shows a sample of using time command to determine the length of time to calculate. This command reports following information.

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.

Example 4-82 Counting CPU ticks using the time command

```
r33n05:/ # /usr/bin/time bc <<! /dev/null
> 999^9999
> !

... skip ...

Real   5.91
User   4.11
System 0.16
r33n05:/ #
```

4.2.18 The timex command

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

Syntax

timex [-o] [-p] [**-s**] **Command**

Flags

-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 will be timed by the time command.

Examples

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. Example 4-83 shows a sample of **timex** command with **-s** flag.

Example 4-83 Displaying statistics information using the times command

```
r33n05:/ # timex -s bc <<! /dev/null
> 9999^9999
> !

... skip ...

real 10.45
user 7.30
sys  0.22


AIX r33n05 3 5 00C3E3CC4C00    10/25/04

System configuration: lcpu=4 ent=2.00

12:17:51    %usr    %sys    %wio   %idle   physc   %entc
12:18:02      34       1       0      65    0.74    37.1

System configuration: lcpu=4 ent=2.00

12:17:51 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
12:18:02       0       0       0       0       0       0       0       0

System configuration: lcpu=4 mem=6912MB ent=2.00
```

```
12:17:51   slots cycle/s fault/s  odio/s
12:18:02  130178    0.00   32.16    0.00

System configuration: lcpu=4 ent=2.00

12:17:51 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
12:18:02       0       0    3893       0       0       0

System configuration: lcpu=4 ent=2.00

12:17:51 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
12:18:02    3787       4    3725    0.28    0.47    7558    4108

System configuration: lcpu=4 ent=2.00

12:17:51 cswch/s
12:18:02     139

System configuration: lcpu=4 ent=2.00

12:17:51  iget/s lookuppn/s dirblk/s
12:18:02       0          6       0

System configuration: lcpu=4 ent=2.00

12:17:51 runq-sz %runocc swpq-sz %swpocc
12:18:02     1.0      73

System configuration:

12:17:51  proc-sz     inod-sz     file-sz     thrd-sz
12:18:02  135/262144  0/176       1297/2455   284/524288

System configuration: lcpu=4 ent=2.00

12:17:51   msg/s  sema/s
12:18:02    0.00    0.00
r33n05:/ #
```

Useful combinations

- **timex [command]**
- **timex -s command**

4.3 CPU related tuning tools and techniques

This section describes some additional CPU related performance tools and tuning techniques.

4.3.1 The smtctl command

The **smtctl** command controls the enabling and disabling of processor simultaneous multi-threading mode.

This command is provided for privileged users and applications to control utilization of processors with simultaneous multi-threading support. The simultaneous multi-threading mode allows processors to have thread level parallelism at the instruction level. This mode can be enabled or disabled for all processors either immediately or on subsequent boots of the system. This command controls the simultaneous multi-threading options.

Syntax

smtctl [**-m off** | **on** [**-w boot** | **now**]]

Flags

-m off	Sets the simultaneous multi-threading mode to disabled.
-m on	Sets the simultaneous multi-threading mode to enabled.
-w boot	Makes the simultaneous multi-threading mode change effective on next and subsequent reboots.
-w now	Makes the simultaneous multi-threading mode change immediately but will not persist across reboot.

Note: If neither the -w boot or the -w now options are specified, then the mode change is made immediately and will persist across subsequent boots.

Example

Displaying the current SMT setting

To check the status of SMT, you can use smtctl command without flag. Example 4-84 on page 277 shows a sample of the **smtctl** command without flag. The following information is reported for current SMT status.

SMT Capability	Indicator that the physical processors are capable of simultaneous multi-threading
SMT Mode	Current runtime simultaneous multi-threading mode of disabled or enabled

SMT Boot Mode	Current boot time simultaneous multi-threading mode of disabled or enabled
SMT Threads	The number of simultaneous multi-threading threads per physical processor
SMT Bound	Indicator that the simultaneous multi-threading threads are bound on the same physical processor

Example 4-84 Displaying the current SMT status

```
r33n05:/ # smtctl

This system is SMT capable.

SMT is currently enabled.

SMT boot mode is not set.


Processor 1 has 2 SMT threads
SMT thread 0 is bound with processor 1
SMT thread 2 is bound with processor 1


Processor 2 has 2 SMT threads
SMT thread 1 is bound with processor 2
SMT thread 3 is bound with processor 2
r33n05:/ #
```

Changing the SMT mode

Using the **smtctl** with **-m on** flag, you can enable the SMT mode. Example 4-85 shows a sample of enabling SMT mode. In this example, the mode change is made immediately and will persist across subsequent boots because -w flag is not specified.

Example 4-85 Enabling the SMT mode

```
r33n05:/ # smtctl -m on
smtctl: SMT is now enabled and will persist across reboots.
        Note that the boot image must be remade with the bosboot
        command before the next reboot.
r33n05:/ #
```

Using the **smtctl** with the **-m off** flag, you can disable the SMT mode. Example 4-86 on page 278 shows a sample of disabling the SMT mode. In this example, the mode change immediately but will not persist across reboot because -w now option is specified.

Example 4-86 Disabling the SMT mode

```
r33n05:/ # smtctl -m off -w now
smtctl: SMT is now disabled.
r33n05:/ #
```

Useful combinations

- **smtctl**
- **smtctl -m on -w now**
- **smtctl -m off- w now**

4.3.2 The bindintcpu command

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 **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. 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.

Syntax

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

Examples

The **bindintcpu** command can be useful for redirecting an interrupt to a specific processor. In a shared processor LPAR, the **bindintcpu** command binds bus interrupt level to a virtual CPU. 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 4.3.3, “The bindprocessor command” on page 280 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 4-87. Here we see that the interrupt level is 85.

Example 4-87 How to determine the interrupt level of an adapter

```
# lsattr -El ent0
alt_addr        0x000000000000   Alternate Ethernet Address                 True
busintr         85               Bus interrupt level                       False
busmem          0xc8030000       Bus memory address                        False
chksum_offload  yes              Enable hardware transmit and receive checksum True
intr_priority   3                Interrupt priority                        False
ipsec_offload   no               IPsec Offload                             True
large_send      yes              Enable TCP Large Send Offload             True
media_speed     Auto_Negotiation Media Speed                               True
poll_link       no               Enable Link Polling                       True
poll_link_timer 500              Time interval for Link Polling            True
rom_mem         0xc8000000       ROM memory address                        False
rx_hog          1000             RX Descriptors per RX Interrupt           True
rxbuf_pool_sz   1024             Receive Buffer Pool Size                  True
rxdesc_que_sz   512              RX Descriptor Queue Size                  True
slih_hog        10               Interrupt Events per Interrupt            True
tx_preload      1520             TX Preload Value                          True
tx_que_sz       8192             Software TX Queue Size                    True
txdesc_que_sz   512              TX Descriptor Queue Size                  True
use_alt_addr    no               Enable Alternate Ethernet Address         True
```

To determine which CPUs are available on the system, the **bindprocessor** command can be used as in Example 4-88.

Example 4-88 How to determine the available CPUs

```
# bindprocessor -q
The available processors are:  0 1 2 3
```

In order to redirect the interrupt level 85 to CPU1 on the system, use the **bindintcpu** command as in Example 4-89 on page 280. All interrupts from bus interrupt level 85 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.

Example 4-89 redirect the specified interrupt to CPU

```
# bindintcpu 85 1
#
```

In Example 4-90, 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 4-90 Error message against incorrect CPU number

```
# bindintcpu 85 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 4-91 to obtain interrupt statistics. The column heading level shows the interrupt level, and the column heading count gives the number of interrupts since system startup.

Example 4-91 Displaying interrupt statistics with the vmstat command

```
# vmstat -i
priority level    type   count module(handler)
    0       2   hardware  6382 i_mpc_int_handler(793a34)
    1       4   hardware   195 /usr/lib/drivers/isa/rsdd_rspc(2439eec)
    3      85   hardware 150622 /usr/lib/drivers/pci/scentdd(250ba8c)
    3      87   hardware 84089 /usr/lib/drivers/pci/s_scsiddpin(22016e8)
    3      88   hardware    90 /usr/lib/drivers/pci/s_scsiddpin(22016e8)
    3     101   hardware    60 /usr/lib/drivers/pci/s_scsiddpin(22016e8)
    3     105   hardware  3823 /etc/drivers/pci/efcddpin(21cb8d8)
    3     115   hardware    39 /etc/drivers/pci/efcddpin(21cb8d8)
```

4.3.3 The bindprocessor command

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. This applies to the initial thread in the new process created by the fork subroutine: the new thread inherits the bind properties of the thread which called fork. When the exec subroutine is called, thread properties are left unchanged.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.

In a shared processor LPAR, the **bindprocessor** command binds to virtual CPUs instead of physical CPUs. This aspect could possibly cause problems for an application or kernel extension that is dependent on executing on a specific physical CPU.

Syntax

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.

Examples

Display the available processors

To display the available processors, the bindprocessor command can be used as in Example 4-92.

Example 4-92 Displaying available processors with the bindprocessor command

```
# bindprocessor -q
The available processors are:  0 1 2 3
```

Binding a thread to processor

Example 4-93 shows a sample of using the **bindprocessor** command. In this example, the cputest process is binded to processor 1. The ps command with **-o THREAD** option is useful to know whether a thread is bound to a processor or not.

Example 4-93 Bind a thread to processor

```
r33n05:/ # ps -o THREAD
    USER    PID   PPID      TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root 385176 495628        - A    0  60  1        -   240001  pts/1   - -ksh
    root 536592 557206        - A    0  60  1        -   200001  pts/1   - ps -o THREAD
```

```
    root 557206 659630        - A    0  60  1        -   200001  pts/1    - /usr/bin/ksh
    root 569598 557206        - A    0  68  1        -   200001  pts/1    - cputest
r33n05:/ # bindprocessor -q
The available processors are:  0 1 2 3
r33n05:/ # bindprocessor 569598 1
r33n05:/ # Os -o THREAD
/usr/bin/ksh: Os:  not found.
r33n05:/ # ps -o THREAD
    USER    PID   PPID      TID ST  CP PRI SC    WCHAN        F     TT BND COMMAND
    root 385176 495628        - A    0  60  1        -   240001  pts/1    - -ksh
    root 536602 557206        - A    0  60  1        -   200001  pts/1    - ps -o THREAD
    root 557206 659630        - A    0  60  1        -   200001  pts/1    - /usr/bin/ksh
    root 569598 557206        - A    0  68  1        -   200001  pts/1    1 cputest
r33n05:/ #
```

4.3.4 The schedo command

The **schedo** command is used to set or display current or next boot values for all CPU scheduler tuning parameters. This command can only be executed by root user. 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** command has replaced the **schedtune** command. In AIX 5.2, a compatibility script named **schedtune** is provided to help the transition. In AIX 5.3, the **schedtune** script is not available anymore. The **schedo** command resides in /usr/bin/schedo and is part of the bos.perf.tune fileset. This fileset is 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 5.3 Performance Management Guide*, SC23-4905, before using these tools.

Syntax

schedo [-p | -r] { -o Tunable[=Newvalue]}

schedo [-p | -r] { -d Tunable }

schedo [-p | -r] -D

schedo [-p | -r] -a

schedo -h [Tunable]

schedo -L [Tunable]

schedo -x [Tunable]

schedo -?

Flags

-h [Tunable] Displays help about the Tunable parameter if one is specified. Otherwise, displays the **schedo** command usage statement.

-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 Tunable Resets Tunable to its default value. If a tunable needs to be changed (that is, it is currently not set to its default value, and -r is not used in combination, it won't be changed but a warning is displayed.

-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 will not be changed but display warning message.

-o Tunable [=Newvalue]

Displays the value or sets Tunable to Newvalue. If a tunable needs to be changed (the specified value is different than 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. 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, 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 can't 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 in combination with -o, -d or -D, that is, 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 display instead of current values.

-L [Tunable] Lists the characteristics of one or all tunables.

-x [Tunable] Lists characteristics of one or all tunables.

Examples

Displaying current parameter value

Beginning with AIX 5L Version 5.3, several tuning parameters have been added to the **schedo** command. Example 4-94 shows all CPU scheduler parameters.

Example 4-94 Displaying current parameter values with the schedo command

```
r33n05:/ # 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
        big_tick_size = 1
            force_grq = 0
     smt_snooze_delay = 0
  setnewrq_sidle_mload = 384
    sidle_S1runq_mload = 64
```

```
    sidle_S2runq_mload = 134
    sidle_S3runq_mload = 134
    sidle_S4runq_mload = 4294967040
 search_globalrq_mload = 256
  search_smtrunq_mload = 256
     smtrunq_load_diff = 2
   shed_primrunq_mload = 64
        unboost_inflih = 1
    n_idle_loop_vlopri = 100
       hotlocks_enable = 0
         krlock_enable = 1
  krlock_conferb4alloc = 0
    krlock_spinb4alloc = 1
    krlock_confer2self = 0
   krlock_spinb4confer = 1024
    slock_spinb4confer = 1024
r33n05:/ #
```

Beginning with AIX 5L Version 5.3, the following parameters are supported. In an environment other than Power5 processor, these new parameter values are displayed as “N/A”.

smt_snooze_delay — Amount of time in microseconds in idle loop without useful work before snoozing (calling h_cede). A value of -1 indicates to disable snoozing, a value of 0 indicates to snooze immediately. Default: 0. Range: -1 to 100000000 (max. 100 seconds).

setnewrq_sidle_mload — Minimum system load above which idle secondary sibling threads will be considered for new work even when primary is not idle. Default: 384. Range: 0 to 4294967040 (0xFFFFFF00).

sidle_S1runq_mload — The minimum load above which idle load balancing for secondary sibling threads will search for work in the primary sibling thread's run queue. Default: 64. Range: 0 to 4294967040 (0xFFFFFF00)

sidle_S2runq_mload — Minimum load above which secondary sibling threads will look for work among other run queues owned by CPUs within their S2 affinity domain during idle load balancing. Default: 134. Range: 0 to 4294967040 (0xFFFFFF00). It is recommended that this tunable parameter be never set to a value that is less than the value of sidle_S1runq_mload.

sidle_S3runq_mload — Minimum load above which secondary sibling threads will look for work among other run queues owned by CPUs within their S3 affinity domain during idle load

	balancing. Default: 134. Range: 0 to 4294967040 (0xFFFFFF00). It is recommended that this tunable parameter be never set to a value that is less than the value of sidle_S2runq_mload.
sidle_S4runq_mload	Minimum load above which secondary sibling threads will look for work on any local run queues. Default: 4294967040 (0xFFFFFF00). Range: 0 to 4294967040 (0xFFFFFF00). It is recommended that this tunable parameter be never set to a value that is less than the value of sidle_S3runq_mload.
search_globalrq_mload	Minimum load above which secondary sibling threads will look for work in the global run queue in the dispatcher. Default: 256. Range: 0 to 4294967040 (0xFFFFFF00).
search_smtrunq_mload	Minimum load above which the dispatcher will also search the run queues belonging to its sibling hardware threads. This is meant for load balancing on a physical processor and is not the same as idle load balancing as this check is made in the dispatcher when choosing the next job to be dispatched. This works in conjunction with the smtrunq_load_diff tunable. Default: 256. Range: 0 to 4294967040 (0xFFFFFF00).
smtrunq_load_diff	Minimum load difference between sibling run queue loads for a task to be stolen from the sibling's run queue. This is enabled only when the load is greater than the value for the search_smtrunq_mload tunable. Default: 2. Range: 1 to 4294967040 (0xFFFFFF00).
shed_primrunq_mload	The maximum load below which the secondary sibling threads will try to shed work onto the primary sibling thread's run queue. Default: 64. Range: 0 to 4294967040 (0xFFFFFF00).
unboost_inflih	Enables (1) or disables (0) the unboost of the hot lock priority in the flih. When disabled, the unboost occurs in the dispatcher. Default: 1 (enabled). Range: 0 to 1.
n_idle_loop_vlopri	Number of times to run the low hardware priority loop each time in idle loop if no new work is found. Default: 100. Range: 0 to 1000000.
hotlocks_enable	Enables (1) or disables (0) the hardware priority boosting of hot locks. Default: 0 (disabled). Range: 0 to 1.

krlock_enable	Enables (1) or disables (0) krlocks. This parameter only applies to the 64bit kernel. Default: 1 (enabled). Range: 0 to 1.
krlock_conferb4alloc	Enables (1) or disables conferring after spinning slock_spinb4confer before trying to acquire or allocating krlock. This parameter only applies to the 64bit kernel. Default: 0 (disabled). Range: 0 to 1.
krlock_spinb4alloc	Number of additional aquisition attempts after spinning slock_spinb4confer, and conferring (if krlock_conferb4alloc is on), before allocating krlock. This parameter only applies to the 64bit kernel. Default: 1. Range: 1 to MAXINT.
krlock_confer2self	Enables (1) or disables (0) conferring to self after trying to acquire krlock krlock_spinb4confer times. This parameter only applies to the 64bit kernel. Default: 1 (enabled). Range: 0 to 1.
krlock_spinb4confer	Number of krlock acquisition attempts before conferring to the krlock holder (or self). This parameter only applies to the 64bit kernel. Default: 1024. Range: 0 to MAXINT.
slock_spinb4confer	Number of attempts for a simple lock before conferring. Default: 1024. Range: 0 to MAXINT.

Changing a parameter value

To change the current parameter value of **schedo** with the **-o** flag. Example 4-95 shows a sample of how to change a parameter value using the **-o** flag. In this example, *sched_R* parameter value is changed from 16 to 5. The *sched_R* and *sched_D* parameters are used for calculating the CPU scheduler's priority.

For more information about CPU scheduler, refer to Chapter 11. "CPU performance monitoring" of the *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905, which can be found at:

http://publib.boulder.ibm.com/infocenter/pseries/topic/com.ibm.aix.doc/aixbman/prftungd/prftungd.pdf

Example 4-95 Changing a parameter value

```
r33n05:/ # schedo -a | grep sched
               sched_D = 16
               sched_R = 16
r33n05:/ # schedo -o sched_R=5
Setting sched_R to 5
r33n05:/ # schedo -a | grep sched
               sched_D = 16
```

```
                sched_R = 5
r33n05:/ #
```

4.3.5 The nice command

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. The renice command is used to change the nice value of one or more processes that are running on a system.

The nice 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.

syntax

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 to 39, the nice value is set to the value of the limit that was exceeded.

Parameters

Command This is the actual command that will run with the modified nice value.

Examples

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.

Displaying the current nice value

To determine the nice value, use the **ps** command with **-l** flag as in Example 4-96. The nice value for a user process that is started in the foreground is 20 by default, and if the If the process is launched in the background, the nice value is 24 by default.

Example 4-96 Displaying the nice value using the ps command

```
r33n05:/ # ps -l
      F S UID    PID   PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 240001 A   0 385176 495628   0  60 20 177c77400   716           pts/1  0:00 ksh
 200001 A   0 557210 385176   0  60 20 187cd8400   820           pts/1  0:00 ps
 240001 A   0 569598      1   0  68 24 97d09400   212          pts/1  0:00 cputest
r33n05:/ #
```

Reducing the priority of a process

The priority of the process can be reduced by increasing the nice value. Example 4-97 shows a sample of reducing the nice value of a process. In this example, nice value is specified to reduce by 10.

Example 4-97 Reducing the priority of a process

```
r33n05:/ # nice -10 ps -l
      F S UID    PID   PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
 240001 A   0 385176 495628   0  60 20 177c77400   716           pts/1  0:00 ksh
 200001 A   0 557218 385176   0  80 30 187cd8400   820           pts/1  0:00 ps
 240001 A   0 569598      1   0  68 24 97d09400   212          pts/1  0:00 cputest
r33n05:/ #
```

4.3.6 The renice command

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 **renice** command resides in /usr/sbin/renice, is linked from /usr/bin/renice, and is part of the bos.adt.prof fileset, which is installable from the AIX base installation media.

syntax

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 or any other flag is not specified, 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). In the case where the -u flag is used, this value denotes the user identification number (UID).

Examples

Changing the thread's priority

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. Example 4-98 on page 290 shows a sample of reducing the thread's priority using the **renice** command.

Example 4-98 Changing the thread's priority using the renice command

```
r33n05:/ # nice -10 ps -l
      F S UID    PID   PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  240001 A   0 385176 495628   0  60 20 177c77400   716           pts/1  0:00 ksh
  200001 A   0 557218 385176   0  80 30 187cd8400   820           pts/1  0:00 ps
  240001 A   0 569598      1   0  68 24 97d09400   212           pts/1  0:00 cputest
r33n05:/ # renice -n 10 -p 569598
r33n05:/ # ps -l
      F S UID    PID   PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
  240001 A   0 385176 495628   0  60 20 177c77400   716           pts/1  0:00 ksh
  200001 A   0 557224 385176   0  60 20 187cd8400   820           pts/1  0:00 ps
  240001 A   0 569598      1   0  88 34 97d09400   212           pts/1  0:00 cputest
r33n05:/ #
```

Useful combinations

- **renice -n [increment] [ID]**
- **renice -n [increment] -g [ID]**
- **renice -n [increment] -u [ID]**

4.4 CPU summary

This section presents CPU related performance commands which help us summarize the data collected.

4.4.1 Other useful commands for CPU monitoring

Here are some other useful more commands

The alstat and emstat commands

The **alstat** command displays alignment exception statistics. The **emstat** command displays emulation exception statistics. /usr/bin/emstat is linked from /usr/bin/alstat, so both command has same binary code.

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.

Syntax

alstat [-e | -v] [Interval] [Count]

emstat [-a | -v] [Interval] [Count]

Useful combinations

- **alstat -e [interval] [count]**
- **alstat -v [interval] [count]**
- **emstat -a [interval] [count]**
- **emstat -v [interval] [count]**

The trcevgrp command

The **trcevgrp** command is used to maintain the trace event groups. The trcevgrp command reside in /usr/bin and are part of the bos.sysmgt.trace fileset, which is installable from the AIX base installation media.

syntax

trcevgrp -l [event-group [...]]

trcevgrp -r [event-group [...]]

trcevgrp -a -d "group-description" -h "hook-list" event-group

trcevgrp -u [-d "group-description"] [-h "hook-list"] event-group]

The gennames, genld, genkld, genkex, gensyms commands

The **gennames**, **genld**, **genkld**, **genkex**, and **gensyms** commands extract information from the 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.

These commands reside in /usr/bin and are part of the bos.perf.tools fileset, which can be installed from the AIX base installation media.

syntax

gennames[-f]

genld [-h | -l [-d]]

genkld [-dh]

genkex [-dh]

gensyms [-ofhs] [-k kernel] [-i file] [-b binary[,binary[,...]]] [-S path]

The locktrace command

The **locktrace** command is used to controls kernel lock tracing. If the machine has been rebooted after running the **bosboot -L** command, kernel lock tracing can be turned on or off for one or more individual lock classes, or for all lock classes. If **bosboot -L** was not run, lock tracing can only be turned on for all locks or none.

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.

Syntax

locktrace [-r ClassName | -s ClassName | -S | -R | -l]

-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.

Useful combinations

- **locktrace -S**
- **locktrace -l**
- **locktrace -R**

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.

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.

Syntax

stripnm [-x | -d] [-s] [-z] File

-d	Prints symbol address values in decimal format. This is the default with -z flag.
-x	Prints symbol address values in hexadecimal format. This is the default without -z flag.
-z	Uses the old format.

Useful combinations

- **stripnm [object_file] > [ouput_file]**
- **stripnm -d [object_file] > [ouput_file]**
- **stripnm -xz [object_file] > [ouput_file]**

Process-related commands

The /proc filesystem provides a mechanism to control processes. It also gives access to information about the current state of processes and threads, but in

binary form. The **proctools** commands provide ascii reports based on some of the available information. Following **proctools** commands are supported.

procwdx	Prints the current working directory of processes.
procfiles	Reports information about all file descriptors opened by processes.
procflags	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.
procmap	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.
procstop	Stops processes on the PR_REQUESTED event.
procrun	Starts a process that has stopped on the PR_REQUESTED event.
procwait	Waits for all of the specified processes to terminate.
proctree	Prints the process tree containing the specified process IDs or users.

These commands reside in /usr/bin and is part of the bos.perf.proctools fileset, which is installable from the AIX base installation media.

Syntax

procwdx [**-F**] [**ProcessID**] ...

procfiles [**-F**] [**-n**][**ProcessID**] ...

procflags [-r] [**ProcessID**] ...

proccred [**ProcessID**] ...

procmap [**-F**] [**ProcessID**] ...

procldd [**-F**] [**ProcessID**] ...

procsig [**ProcessID**] ...

procstack [**-F**] [**ProcessID**] ...

procstop [**ProcessID**] ...

procrun [**ProcessID**] ...

procwait [-v] [**ProcessID**] ...

proctree [-a] [{ **ProcessID** | User }]

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.

5

Memory analysis and tuning

In this chapter we discuss how to monitor and tune memory characteristics. By monitoring the memory you can observe when memory performance is degrading and then use the tuning techniques discussed to improve performance. This chapter describes the following tools:

- Memory monitoring
  - The **ps** command
  - The **sar** command
  - The **svmon** command
  - The **topas** monitoring tool
  - The **vmstat** command
- Memory tuning
  - The **vmo** command
- Other commands related to memory performance

5.1 Memory monitoring

Monitoring any performance characteristics is a very important part of achieving the best results possible. There are many ways to investigate different parameters and settings, but combining several tools and commands can give you the best overall picture of performance. These commands have many uses, in this section we will only discuss how they can be used to monitor memory. We will show how these commands can be used to gauge how the memory of the system is performing at any given moment.

5.1.1 The ps command

The ps (Process Status) command shows the current status of active processes. It is located /usr/bin, installed by default from the AIX base installation media, and is part of the bos.rte.commands fileset.

Syntax

```
Usage: ps [-AMNaedfklm] [-n namelist] [-F Format] [-o specifier[=header],...]
               [-p proclist][-G|-g grouplist] [-t termlist] [-U|-u userlist]
[-c classlist] [ -T pid] [ -L pidlist ]
Usage: ps [aceglnsuvwxU] [t tty] [processnumber]
```

Useful combinations of the ps command for memory statistics

- **ps aux**
- **ps v**
- **ps -ealf**

Using the ps command

The u and v flags report the following statistics

- %MEM, which is the percentage of real memory a process is using.
- RSS, the amount of real memory size of the process (in 1KB units).

The u flag also reports the SZ statistic, which represents the size of the core image of the process (in 1KB units).

The ps command can be used to determine what percentage of real memory a process is using. In Example 5-1 you can identify the processes using the highest percentages of real memory, by looking at the %MEM column, which is sorted in descending order.

Example 5-1 Example using ps aux

```
[p630n04][/]> ps aux | head -1 ; ps aux | sort -rn +3 | head
```

```
USER       PID %CPU %MEM   SZ  RSS    TTY STAT    STIME  TIME COMMAND
root     32958  0.0  1.0 19060 19076      - A      Oct 11  0:01 java -Djava.secur
root     29290  0.0  1.0 15316 15328      - A      Oct 08  0:04 /usr/java14/jre/b
root     38072  0.0  0.0  176  188  pts/8 A    10:01:37  0:00 sort -rn +3
root     37646  0.0  0.0 3640 3412       - A    14:21:39  0:00 Xvnc :5 -desktop
root     35352  0.0  0.0 1056 1116       - A    14:21:42  0:00 xterm
root     35078  0.0  0.0 1092 1120       - A    12:05:55  0:51 /usr/sbin/rsct/bi
root     34800  0.0  0.0  692  716  pts/8 A    10:01:37  0:00 ps aux
root     33848  0.0  0.0  668  708       - A      Oct 11  0:00 /bin/ksh /usr/per
root     33668  0.0  0.0  128  136  pts/8 A    10:01:37  0:00 head
root     33472  0.0  0.0  716  756  pts/2 A    14:21:42  0:00 -ksh
```

You can also see similar statistics using the v flag.

Example 5-2 Using ps v

```
r33n01:/ # ps v 868488
    PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 868488  pts/0 A     0:43    0    56    24    xx     2     8 19.9  0.0 cpu_load
```

The ps command can also be used to track how much virtual memory a process using. In Example 5-3 you can identify which processes are using the most amount of virtual memory, by looking at the SZ (size) column, which is listed in descending order.

Example 5-3 Example using ps -ealf

```
r33n01:/ # 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 856238 901306   0  39 20 177317400  3176        *   Oct 12      - 39:35
/usr/sbin/rsct/bin/IBM.CSMAgentRMd
  340001 A     root 811154 901306   0  39 20 b72ab400  2624 f1000588d000e740   Oct 12      -
0:05 /usr/sbin/rsct/bin/rmcd -r
  240001 A     root 807052 901306   0  60 20 97329400  2064        *   Oct 12      -  0:00
/usr/sbin/rsct/bin/IBM.ERrmd
  240001 A     root 479470 901306   0  60 20 1d723d400  1704           Oct 12      -  0:00
sendmail: accepting connections
  240001 A     root 790660 901306   0  60 20 37343400  1608        *   Oct 12      -  0:00
/usr/sbin/rsct/bin/IBM.AuditRMd
  240001 A     root 798856 901306   0  60 20 b734b400  1584        *   Oct 12      -  0:00
/usr/sbin/rsct/bin/IBM.DRMd
  240001 A     root 802954 901306   0  60 20 1f731f400  1576        *   Oct 12      -  0:00
/usr/sbin/rsct/bin/IBM.HostRMd
  240401 A     root 823450 901306   0  60 20 27322400  1492        *   Oct 12      -  0:00
/usr/sbin/rsct/bin/IBM.ServiceRMd
  240001 A     root 401502 901306   0  60 20 97249400  1036           Oct 12      -  0:04
/usr/sbin/snmpmibd
```

```
  240001 A    daemon 884926 901306   0  60 20 197279400   860            Oct 12      -  0:00
/usr/sbin/rpc.statd -d 0 -t 50
```

5.1.2 The sar command

The **sar** command is very useful in determining real time statistics about your system. It writes to standard output the contests of selected cumulative activity counters in the operating system. It is located in /usr/sbin, is installable from the AIX base installation media, and is part of the bos.rte.commands fileset.

Syntax

```
sar { -A | [-a][-b][-c][-d][-k][-m][-q][-r][-u][-v][-w][-y] } [-s hh[:mm[:ss]]]
[-e hh[:mm[:ss]]] [-P processor_id[,...] | ALL] [-f file] [-i seconds] [-o
file] [interval [number]][-X file] [-i seconds] [-o file] [interval [number]]
```

Useful combinations of the sar command

- **sar -r**

Using the sar command

The sar command with the -r flag will display paging statistics.

Example 5-4 Using sar -r

```
[p630n04][/]> sar -r 10 5

AIX p630n04 3 5 000685CF4C00    10/13/04

System configuration: lcpu=4 mem=8192MB

14:29:15   slots cycle/s fault/s  odio/s
14:29:25 2096685    0.00    1.75    0.00
14:29:35 2096685    0.00  112.00    0.00
14:29:45 2096685    0.00    0.00    0.90
14:29:55 2096685    0.00    0.00    0.00
14:30:05 2096685    0.00  170.00    0.00

Average  2096685       0      56       0
```

The output of Example 5-4 shows that there was approximately 8190 MB of free space on the paging spaces in the system (2096685 * 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 **cy** column reported by **vmstat**).

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.

5.1.3 The svmon command

The **svmon** command is an analysis tool for virtual memory.It captures the current state of memory, including real, virtual and paging space memory. The **svmon** command invokes the **svmon_back** command. Both are located in /usr/lib/perf, and both part of the perfagent.tools fileset.

Syntax

```
svmon [-G  [-i Intvl [NumIntvl] ][-z] ]
svmon [-P [pid1...pidn] [-r] [-u|-p|-g|-v] [-ns] [-wfc] [-q] [-t Count] [ -i
Intvl [NumIntvl] ] [-l] [-j] [-z]  [-m] ]
svmon [-S [sid1...sidn] [-r] [-u|-p|-g|-v] [-ns] [-wfc] [-q] [-t Count] [ -i
Intvl [NumIntvl] ] [-l] [-j] [-z] [-m] ]
svmon [-D sid1...sidn [-b] [-q] [ -i Intvl [NumIntvl] ][-z]]
svmon [-F [fr1...frn] [-q] [-i Intvl [NumIntvl] ][-z] ]
svmon [-C cmd1...cmdn [-r] [-u|-p|-g|-v] [-ns] [-wfc] [-q] [-t Count] [ -i
Intvl [NumIntvl] ] [-d] [-l] [-j] [-z] [-m] ]
svmon [-U [lognm1...lognmn] [-r] [-u|-p|-g|-v] [-ns] [-wfc] [-t Count] [ -i
Intvl [NumIntvl] ] [-d] [-l] [-j] [-z] [-m] ]
svmon [-W [class1...classn] [-e] [-r] [-u|-p|-g|-v] [-ns] [-wfc] [-q] [-t
Count] [ -i Intvl [NumIntvl] ] [-l] [-j] [-z] [-m] ]
svmon [-T [tier1...tiern] [-a superclass] [-x] [-e] [-r] [-u|-p|-g|-v] [-ns]
[-wfc] [-q] [-t Count] [ -i Intvl [NumIntvl] ] [-l] [-z] [-m] ]
```

If no option is given, **svmon -G** is the default.

Useful combinations of the svmon command

- **svmon or svmon -G**
- **svmon -P**
- **svmon -C**
- **svmon -i**

Using the svmon command

When you use the -G flag or give no flags with the **svmon** command, it will provide you with the global view. The global view shows system-wide memory utilization. In Example 5-5 on page 302, you can the amount of real memory pages that are

inuse and **free** are shown. The number of **pg space** pages **inuse** shows how much paging space is being used.

Example 5-5 Example using svmon or svmon -G

```
[p630n04][/]> svmon
               size       inuse        free         pin    virtual
memory      2097152      880310     1216842      167387     245557
pg space    2097152         533

               work       pers       clnt      lpage
pin          167168          0        219          0
in use       245557          0     634753          0
```

To print out global statistics over an interval, use the -G flag. In Example 5-6 we will repeat it five times at two-second intervals.

Example 5-6 Using svmon -G over an interval

```
r33n01:/ # svmon -G -i 2 5
               size       inuse        free         pin    virtual
memory      1835008      194772     1640236      124539     151495
pg space     131072         788

               work       pers       clnt      lpage
pin          124539          0          0          0
in use       171463          0      23309          0
               size       inuse        free         pin    virtual
memory      1835008      194776     1640232      124539     151499
pg space     131072         788

               work       pers       clnt      lpage
pin          124539          0          0          0
in use       171467          0      23309          0
               size       inuse        free         pin    virtual
memory      1835008      194776     1640232      124539     151499
pg space     131072         788

               work       pers       clnt      lpage
pin          124539          0          0          0
in use       171467          0      23309          0
               size       inuse        free         pin    virtual
memory      1835008      194776     1640232      124539     151499
pg space     131072         788

               work       pers       clnt      lpage
pin          124539          0          0          0
in use       171467          0      23309          0
```

```
               size       inuse        free         pin     virtual
memory      1835008      194776     1640232      124539      151499
pg space     131072         788

              work       pers       clnt      lpage
pin           124539          0          0          0
in use        171467          0      23309          0
```

The columns on the resulting svmon report are described as follows:

memory — Statistics describing the use of real memory, shown in 4 K pages.

- size — Total size of memory in 4 K pages.
- inuse — Number of pages in RAM that are in use by a process plus the number of persistent pages that belonged to a terminated process and are still resident in RAM. This value is the total size of memory minus the number of pages on the free list.
- free — Number of pages on the free list.
- pin — Number of pages pinned in RAM (a pinned page is a page that is always resident in RAM and cannot be paged out).

pg space — Statistics describing the use of paging space, shown in 4 K pages. This data is reported only if the -r flag is not used. The value reported is the actual number of paging space pages used, which indicates that these pages were paged out to the paging space. This differs from the vmstat command in that the vmstat command's avm column which shows the virtual memory accessed but not necessarily paged out.

- size — Total size of paging space in 4 K pages.
- inuse — Total number of allocated pages.

in use — Detailed statistics on the subset of real memory in use, shown in 4 K frames.

- work — Number of working pages in RAM.
- pers — Number of persistent pages in RAM.
- clnt — Number of client pages in RAM (client page is a remote file page).

pin — Detailed statistics on the subset of real memory containing pinned pages, shown in 4 K frames.

work	Number of working pages pinned in RAM.
pers	Number of persistent pages pinned in RAM.
clnt	Number of client pages pinned in RAM.

Using the **svmon** command, you can display memory usage statistics for processes. Using the -P flag, and specifying the process id (PID), If no PID is supplied it will provide statistics are displayed for all active processes. You can use Example 5-7 to read the output of the **svmon -P** command.

Example 5-7 Example using svmon -P

```
[p630n04][/]> svmon -P |grep -p Pid
-------------------------------------------------------------------------------
Pid    Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
68532  java             80615     5485        0    29922      N     Y     N
Pid    Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
29290  tnameserv        25022     5471        0    17630      N     Y     N
Pid    Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
15510  hagsd            18305     5487        0    15091      N     N     N
```

Process ID 68532 is using 80615 pages of real memory and no paging space.

The **svmon** command can also be used to track memory being used by a specific command, by using the -C flag of the command. In Example 5-8 the -C flag is used to track the memory usage of the hagsd (in fact this is the high availability group services daemon, part of RSCT) process. You can compare the output in Example 5-7 and Example 5-8 to see how the two flags relate to each other.

Example 5-8 Example using svmon -C

```
[p630n04][/]> svmon -C hagsd

===============================================================================
Command                             Inuse      Pin     Pgsp  Virtual
hagsd                               18305     5486        0    15091

-------------------------------------------------------------------------------
SYSTEM segments                     Inuse      Pin     Pgsp  Virtual
                                     6442     4237        0     6442

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp
Virtual
       0         0 work kernel seg                   -   6442  4237    0  6442

-------------------------------------------------------------------------------
EXCLUSIVE segments                  Inuse      Pin     Pgsp  Virtual
                                     4498        2        0     1293
```

```
    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp
Virtual
   c84b9         1 clnt code,/dev/hd2:18061          -   3143     0    -     -
   40488         f work shared library data          -    970     0    0   970
   18463         3 work working storage              -    204     0    0   204
   48469         2 work process private              -    119     2    0   119
     480         - clnt /dev/hd9var:463              -     47     0    -     -
   e04bc         - clnt /dev/hd9var:509              -     13     0    -     -
   38467         - clnt /dev/hd9var:473              -      2     0    -     -
   b84b7         4 work working storage              -      0     0    0     0

.....................................................................
SHARED segments                       Inuse      Pin     Pgsp  Virtual
                                       7365     1247        0     7356

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp
Virtual
      20         d work shared library text         -   7327  1218    0  7327
   90412         - work                             -     27    27    0    27
   18103         - clnt /dev/hd4:110                -      3     0    -     -
   98213         - clnt /dev/hd4:98                 -      1     0    -     -
   f845f         - work                             -      1     1    0     1
   5002a         - clnt /dev/hd4:23                 -      1     0    -     -
   48029         - clnt /dev/hd4:66                 -      1     0    -     -
   5044a         - work                             -      1     1    0     1
   6002c         - clnt /dev/hd4:861                -      1     0    -     -
   18183         - clnt /dev/hd4:99                 -      1     0    -     -
     220         - clnt /dev/hd4:94                 -      1     0    -     -
```

Memory-leaking programs

A memory leak is a program error that consists of repeatedly allocating memory, using it, and then neglecting to free it. A memory leak in a long-running program, such as an interactive application, is a serious problem, because it can result in memory fragmentation and the accumulation of large numbers of mostly garbage-filled pages in real memory and page space. Systems have been known to run out of page space because of a memory leak in a single program.

A memory leak can be detected with the **svmon** command, by looking for processes whose working segment continually grows. A leak in a kernel segment can be caused by an mbuf leak or by a device driver, kernel extension, or even the kernel. To determine if a segment is growing, use the **svmon** command with the -P and -i options to look at a process or a group of processes and see if any segment continues to grow.

Example 5-9 Using the svmon command with the -P and -i options

```
r33n01:/ # svmon -P 872520 -i 1 3

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
  872520 cpu_loader       14052     6661        0    14050      N     N     N

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
       0         0 work kernel segment               -  10885  6658    0 10885
  1d58bd         d work loader segment               -   3142     0    0  3142
   675c6         2 work process private              -     14     3    0    14
   675e6         f work shared library data          -      9     0    0     9
  1c75dc         1 clnt code,/dev/hd3:4138           -      2     0    -     -
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
  872520 cpu_loader       14052     6661        0    14050      N     N     N

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
       0         0 work kernel segment               -  10885  6658    0 10885
  1d58bd         d work loader segment               -   3142     0    0  3142
   675c6         2 work process private              -     14     3    0    14
   675e6         f work shared library data          -      9     0    0     9
  1c75dc         1 clnt code,/dev/hd3:4138           -      2     0    -     -
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
  872520 cpu_loader       14052     6661        0    14050      N     N     N

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
       0         0 work kernel segment               -  10885  6658    0 10885
  1d58bd         d work loader segment               -   3142     0    0  3142
   675c6         2 work process private              -     14     3    0    14
   675e6         f work shared library data          -      9     0    0     9
  1c75dc         1 clnt code,/dev/hd3:4138           -      2     0    -     -
```

Correlating the svmon output with other commands

Using more than one command to track memory is common, and can be a great asset if you use the right commands with each other. For correlating svmon and vmstat output, see Figure 5-1 on page 307.

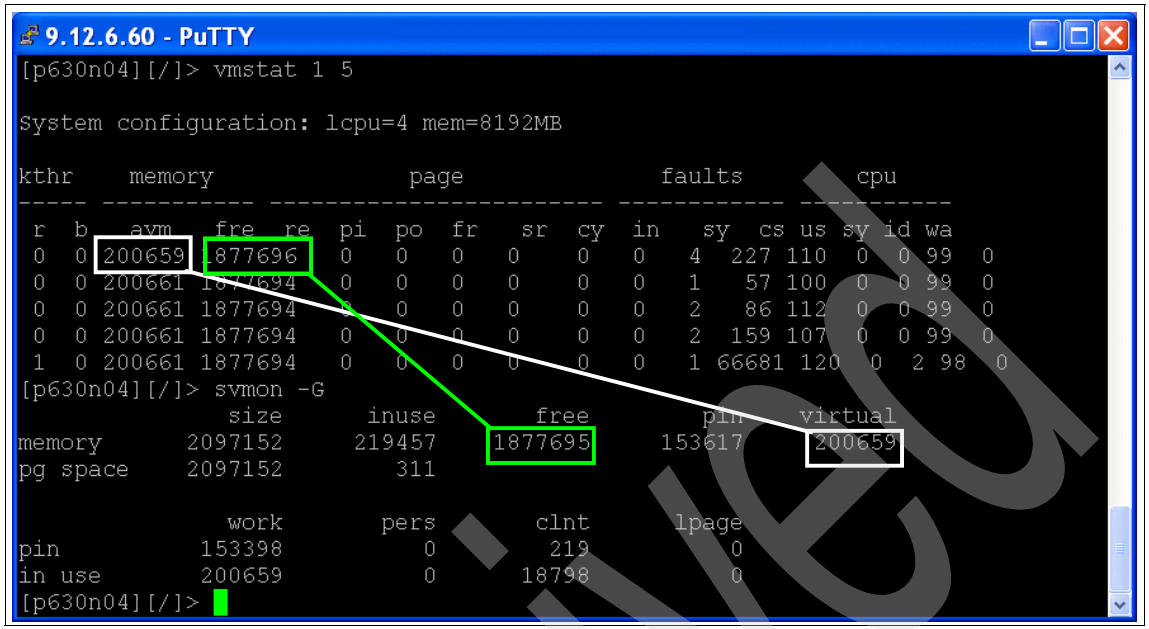

Figure 5-1 Correlating svmon and vmstat output

For correlating svmon and ps output see Example 5-10.

Example 5-10 Correlating svmon and ps output

```
[p630n04][/]> ps v 20948
   PID    TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ   TRS %CPU %MEM COMMAND
 20948      - A     0:00    0   724   912    xx   131   188  0.0  0.0 twm
[p630n04][/]> svmon -P 20948
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   20948 twm              13184     5421        0    13136      N     N     N

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
      20         d work shared library text         -   6522  1218    0  6522
       0         0 work kernel seg                  -   6433  4201    0  6433
   c0518         2 work process private             -     99     2    0    99
   10542         f work shared library data         -     82     0    0    82
   20504         1 clnt code,/dev/hd2:65851         -     47     0    -     -
     540         - clnt /dev/hd4:4279               -      1     0    -     -
```

In previous example we can calculate the memory consumed by a process:

99 + 82 = 181 * 4k blocks = 724

5.1.4 The topas monitoring tool

The **topas** command is a performance monitoring tool that is ideal for broad spectrum performance analysis.The **topas** command requires the perfagent.tools fileset to be installed on the system. The topas command resides in /usr/bin and is part of the bos.perf.tools fileset that is obtained from the AIX base installable media.

Syntax

```
topas [-d number_of_monitored_hot_disks]

              [-h show help information]

              [-i monitoring_interval_in_seconds]

              [-m Use monochrome mode - no colors]

              [-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]

              [-L show full-screen Logical Partition display]

              [-U username - show username owned processes with -P]

              [-W show full-screen WLM Display]
```

Useful combinations of the topas command

- **topas**
- **topas -i**

Using the topas monitoring tool

The topas monitoring tool tracks many statistics, including memory usage and paging information. In Example 5-11, you can see the output of the topas command.

Example 5-11 Using the topas monitoring tool

```
Topas Monitor for host:    r33n01              EVENTS/QUEUES    FILE/TTY
Wed Oct 27 10:47:56 2004   Interval:  2        Cswitch     147  Readch    11187
                                               Syscall     125  Writech     271
```

```
Kernel    0.1   |#                          |  Reads         4  Rawin         0
User      0.0   |#                          |  Writes       13  Ttyout      214
Wait      0.0   |                           |  Forks         0  Igets         0
Idle     99.9   |###########################|  Execs         0  Namei         1
Physc =  0.00                    %Entc=   0.2  Runqueue    0.0  Dirblk        0
                                               Waitqueue   0.0

Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out
en0       0.3      1.0     1.0     0.0     0.5  PAGING           MEMORY
lo0       0.1      2.0     2.0     0.1     0.1  Faults        0  Real,MB    7167
                                                Steals        0  % Comp     10.1
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp   2.4
hdisk0    0.0      0.0     0.0     0.0     0.0  PgspOut       0  % Client    2.6
                                                PageIn        0
Name            PID  CPU%  PgSp Owner           PageOut       0  PAGING SPACE
topas        741626   0.0   1.5 root            Sios          0  Size,MB     512
mpstat       766092   0.0   0.1 root                             % Used      1.1
mpstat       794688   0.0   0.1 root            NFS (calls/sec)  % Free     98.8
mpstat       778270   0.0   0.1 root            ServerV2       0
mpstat       737532   0.0   0.1 root            ClientV2       0   Press:
IBM.CSMAg    856238   0.0   2.7 root            ServerV3       0   "h" for help
getty        893156   0.0   0.4 root            ClientV3       0   "q" to quit
```

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.

To learn more about the topas monitoring tool, refer to 3.1, “The topas command” on page 64.

5.1.5 The vmstat command

The **vmstat** command is useful for reporting statistics about virtual memory. The **vmstat** command is located in /usr/bin, is part of the bos.acct fileset and is installable from the AIX base installation media.

The **vmstat** command summarizes the total active virtual memory used by all of the processes in the system, as well as the number of real-memory page frames on the free list. Active virtual memory is defined as the number of virtual-memory working segment pages that have actually been touched. This number can be larger than the number of real page frames in the machine, because some of the active virtual-memory pages may have been written out to paging space.

Syntax

vmstat [-fs**v**iItlw] [Drives] **[Interval [Count]]**

Useful combinations of the vmstat command

- **vmstat or vmstat Interval Count**
- **vmstat -v**

Using the vmstat command

The **vmstat** command gives data on virtual memory activity to standard output. The first line of data is an average since the last system reboot. In Example 5-12 you can see a summary of the virtual memory activity since the last system startup.

Example 5-12 Using vmstat

```
r33n01:/ # vmstat

System configuration: lcpu=4 mem=7168MB ent=0
```

```
kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 1  1 148162 1649286   0   0   0    0    0   0   0 29121 133  0  0 99  0  0.00   0.2
```

When determining if a system might be short on memory or if some memory tuning needs to be done, run the **vmstat** command over a set interval and examine the pi and po columns on the resulting report. These columns indicate the number of paging space page-ins per second and the number of paging space page-outs per second. If the values are constantly non-zero, there might be a memory bottleneck. Having occasional non-zero values is not a concern because paging is the main principle of virtual memory.

To use the **vmstat** command, specifying Interval and Count, you would input the interval for the update period in seconds, and the Count should represent the number of iterations to be performed. The first report contains statistics since the system startup. Each report after that contains data collected during the interval time period.

For memory data you should pay attention to the avm, fre, pi and po columns (see Example 5-13).

Example 5-13 Using vmstat Interval Count

```
r33n01:/ # vmstat 1 5

System configuration: lcpu=4 mem=7168MB ent=0

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 0  0 148175 1649272   0   0   0    0    0   0   2  167 134  0  0 99  0  0.00   0.2
 0  0 148177 1649270   0   0   0    0    0   0   1   21 138  0  0 99  0  0.00   0.1
 0  0 148177 1649270   0   0   0    0    0   0   1    9 130  0  0 99  0  0.00   0.1
 0  0 148177 1649270   0   0   0    0    0   0   1   11 132  0  0 99  0  0.00   0.1
 0  0 148177 1649270   0   0   0    0    0   0   5   17 134  0  0 99  0  0.00   0.1
```

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. 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 x (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.

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. 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.

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.

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 vmo command parameters minfree and max free, and the environmental variable PSALLOC.

fr — Number of pages freed. When the VMM requires memory, VMM's page-replacement algorithm is employed to scan the Page Frame Table 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.

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. 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.

Faults — Trap and interrupt rate averages per second over the sampling interval.

in — 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.

sy — 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.

cs — 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.

cpu — Breakdown of percentage use of CPU time. The columns us, sy, id, and wa are averages over all of the processors. I/O wait is a global statistic and is not processor specific.

us — 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.

sy — System time indicates the amount of time a program is in system mode; that is, processes using kernel processes (kprocs) 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.

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.

The vmstat output 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 VMM, and the time that NFS block I/O daemons spend in the VMM waiting for an I/O to complete is reported as I/O wait time.

Using the -v flag you can gather data on the VMM (Example 5-14 on page 315).

Example 5-14 Using vmstat -v

```
r33n01:/ # vmstat -v
              1835008 memory pages
              1741547 lruable pages
              1649277 free pages
                    2 memory pools
               124020 pinned pages
                 80.0 maxpin percentage
                 20.0 minperm percentage
                 80.0 maxperm percentage
                  0.7 numperm percentage
                13521 file pages
                  0.0 compressed percentage
                    0 compressed pages
                  1.0 numclient percentage
                 80.0 maxclient percentage
                17592 client pages
                    0 remote pageouts scheduled
                    0 pending disk I/Os blocked with no pbuf
                    0 paging space I/Os blocked with no psbuf
                 2740 filesystem I/Os blocked with no fsbuf
                  133 client filesystem I/Os blocked with no fsbuf
                    0 external pager filesystem I/Os blocked with no fsbuf
```

This list explains the output:

memory pages	Size of real memory in number of 4 KB pages.
lruable 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 page
: 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.

5.2 Memory tuning

Tuning memory performance is very important and dynamic part of achieving the best results possible. There are many settings that can be changed, and it can be difficult to get the right combination of changes to improve performance. The commands in this section can be used to change memory settings on the system. Some of these commands have multiple uses, however in this section we will only discuss how they can be used to tune memory performance.

5.2.1 The vmo command

The **vmo** command is a run time tool used to tune the VMM settings. It is located in /usr/sbin, and is installable from the base AIX Installation media. All the settings set by the **vmo** command are also saved in /etc/tunables.

Syntax

```
vmo -h [tunable] | {-L [tunable]} | {-x [tunable]}

vmo [-p|-r] (-a | {-o tunable})

vmo [-p|-r] (-D | ({-d tunable} {-o tunable=value}))
```

Useful combinations of the vmo command

- **vmo -a**
- **vmo -h tunable**
- **vmo -L tunable**
- **vmo -r -o tunable=value**

Using the vmo command

To see the current settings, you can use the -a flag. The -a flag shows the current settings of the tunables as in Example 5-15.

Example 5-15 Using vmo -a

```
r33n01:/ # vmo -a
        memory_frames = 1835008
      pinnable_frames = 1710666
              maxfree = 128
              minfree = 120
             minperm% = 20
              minperm = 348308
             maxperm% = 80
              maxperm = 1393237
       strict_maxperm = 0
              maxpin% = 80
               maxpin = 1468007
           maxclient% = 80
            lrubucket = 131072
                defps = 1
            nokilluid = 0
            numpsblks = 131072
              npskill = 1024
              npswarn = 4096
             v_pinshm = 0
pta_balance_threshold = n/a
         pagecoloring = n/a
            framesets = 2
             mempools = 1
            lgpg_size = 0
         lgpg_regions = 0
     num_spec_dataseg = 0
     spec_dataseg_int = 512
      memory_affinity = 1
             htabscale = n/a
  force_relalias_lite = 0
  relalias_percentage = 0
           rpgcontrol = 2
            rpgclean = 0
           npsrpgmin = 6144
           npsrpgmax = 8192
               scrub = 0
          scrubclean = 0
        npsscrubmin = 6144
        npsscrubmax = 8192
data_stagger_interval = 161
 large_page_heap_size = 0
    kernel_heap_psize = 4096
soft_min_lgpgs_vmpool = 0
vm_modlist_threshold = -1
      vmm_fork_policy = 1
      low_ps_handling = 1
```

```
      mbuf_heap_psize = 4096
     strict_maxclient = 1
       cpu_scale_memp = 8
```

Here are the tunable explanations:

cpu_scale_memp — Determines the ratio of CPUs per-mempool. For every cpu_scale_memp CPUs, at least one mempool will be created. Can be reduced to reduce contention on the mempools. Use in conjunction with the tuning of the maxperm parameter.

data_stagger_interval — Specifies what the staggering is that will be applied to the data section of a large-page data executable with LDR_CNTRL=DATA_START_STAGGER=Y.

defps — Turns on/off Deferred Page Space Allocation (DPSA) policy. May be useful to turn off DPSA policy if you are concerned about page-space overcommitment. Having the value on reduces paging space requirements.

force_relalias_lite — If set to 0, a heuristic will be used, when tearing down an mmap region, to determine when to avoid locking the source mmapped segment. This is a scalability trade-off, controlled by relalias_percentage, possibly costing more compute time used.

framesets — Specifies the number of real memory page sets per memory pool.

htabscale — On non-LPAR machines, the hardware page frame table (PFT) is completely software controlled and its size is based on the amount of memory being used. The default is to have 4 page table entries (PTE) for each frame of memory (sz=(M/4096)*4*16 where size of PTE is 16 bytes).

kernel_heap_psize — Sets the default page size to use for the kernel heap. This is an advisory setting and is only valid on the 64-bit kernel. If pages of the specified size cannot be allocated, the kernel heap will use pages of a different, smaller page size. 16M pages should only be used for the kernel heap under high performance environments.

large_page_heap_size — When kernel_heap_psize is set to 16M, this tunable sets the maximum amount of the kernel heap to try to back

	with 16M pages. After the kernel heap grows beyond this amount and 16M is selected kernel_heap_psize, 4K pages will be used for the kernel heap. If this tunable is set to 0, it is ignored, and no maximum is set for the amount of kernel heap that can be backed with 16M pages. This tunable should only be used in very special environments where only a portion of the kernel heap needs to be backed with 16M pages.
lgpg_regions	Specifies the number of pages in the large page pool. This parameter does not exist in 64-bit kernels running on non-POWER4 based machines. Using large pages improves performance in the case where there are many TLB misses and large amounts of memory is being accessed.
low_ps_handling	Specifies the action to change the system behavior in relation to process termination during low paging space conditions.
lrubucket	Specifies the number of memory frames per bucket. The page-replacement algorithm divides real memory into buckets of frames. On systems with multiple memory pools, the lrubucket parameter is per memory pool.
maxclient%	Specifies maximum percentage of RAM that can be used for caching client pages. Similar to maxperm% but cannot be bigger than maxperm%.
maxfree	Specifies the number of frames on the free list at which page-stealing is to stop.
maxperm%	Specifies the point above which the page-stealing algorithm steals only file pages.
maxpin%	Specifies the maximum percentage of real memory that can be pinned.
memory_affinity	This parameter can be used to instruct VMM to allocate memory frames in the same MCM that the executing thread is running in, if possible. This parameter only enables memory affinity, which can then be turned on for a given process by setting its MEMORY_AFFINITY environment variable to MCM. This parameter is only supported on POWER4 and POWER5 based machines.
mempools	Changes the number of memory pools that will be configured at system boot time. This parameter does not exist in UP kernels.
minfree	Specifies the minimum number of frames on the free list at which the VMM starts to steal pages to replenish the free list. Page

	replacement occurs when the number of free frames reaches minfree. If processes are being delayed by page stealing, increase minfree to improve response time. The difference between minfree and maxfree should always be equal to or greater than maxpgahead.
minperm%	Specifies the point below (in percentage of total number of memory frames) which the page-stealer will steal file or computational pages regardless of repaging rates.
nokilluid	User IDs lower than this value are exempt from getting killed due to low page-space conditions. If the system is out of paging space and system administrator's processes are getting killed, set to 1 in order to protect specific user ID processes from getting killed due to low page space or ensure there is sufficient paging space available.
npskill	Specifies the number of free paging space pages at which the operating system begins killing processes. Increase this value if you experience processes being killed because of low paging space.
npswarn	Specifies the number of free paging space pages at which the operating system begins sending the SIGDANGER signal to processes. Increase this value if you experience processes being killed because of low paging space.
npsrpgmax	Specifies the number of free paging space blocks at which the Operating System stops freeing disk blocks on pagein of Deferred Page Space Allocation Policy pages.
npsrpgmin	Specifies the number of free paging space blocks at which the Operating System starts freeing disk blocks on pagein of Deferred Page Space Allocation Policy pages.
npsscrubmax	Specifies the number of free paging space blocks at which the Operating System stops Scrubbing in memory pages to free disk blocks from Deferred Page Space Allocation Policy pages. V
npsscrubmin	Specifies the number of free paging space blocks at which the Operating System starts Scrubbing in memory pages to free disk blocks from Deferred Page Space Allocation Policy pages.
num_spec_dataseg	Reserve special data segment IDs for use by processes executed with the environment variable DATA_SEG_SPECIAL=Y. These data segments are assigned so that the hardware page table entries for pages within these segments are better distributed in the

	cache to reduce cache collisions. As many are reserved as possible up to the requested number. Running vmo -a after reboot displays the actual number reserved. This parameter is only supported in 64-bit kernels running on POWER4 based machines. The correct number to reserve depends on the number of processes run simultaneously with DATA_SEG_SPECIAL=Y and the number of data segments used by each of these processes.
pagecoloring	Turns on or off page coloring in the VMM. This parameter is not supported in 64-bit kernels.
pta_balance_threshold	Specifies the point at which a new pta segment will be allocated. This parameter does not exists in 64-bit kernels.
relalias_percentage	If force_relalias_lite is set to 0, then this specifies the factor used in the heuristic to decide whether to avoid locking the source mmapped segment or not.This is used when tearing down an mmapped region and is a scalability statement, where avoiding the lock may help system throughput, but, in some cases, at the cost of more compute time used. If the number of pages being unmapped is less than this value divided by 100 and multiplied by the total number of pages in memory in the source mmapped segment, then the source lock will be avoided. A value of 0 for relalias_percentage, with force_relalias_lite also set to 0, will cause the source segment lock to always be taken. The Default value is 0. Effective values for relalias_percentage will vary by workload, however, a suggested value is: 200.
rpgclean	Enables or Disables freeing paging space disk blocks of Deferred Page Space Allocation Policy pages on read accesses to them.
rpgcontrol	Enables or Disables freeing of paging space disk blocks at pagein of Deferred Page Space Allocation Policy pages.
scrub	Enables or Disables freeing of paging space disk blocks from pages in memory for Deferred Page Space Allocation Policy pages. V

scrubclean	Enables or Disables freeing paging space disk blocks of Deferred Page Space Allocation Policy pages in memory that are not modified.
soft_min_lgpgs_vmpool	When soft_min_lgpgs_vmpool is non-zero, large pages will not be allocated from a vmpool that has fewer than soft_min_lgpgs_vmpool % of its large pages free. If all vmpools have less than soft_min_lgpgs_vmpool % of their large pages free, allocations will occur as normal.
spec_dataseg_int	Modify the interval between the special data segment IDs reserved with num_spec_dataseg. This parameter is only supported in 64-bit kernels running on POWER4 based machines.
strict_maxclient	If set to 1, the maxclient value will be a hard limit on how much of RAM can be used as a client file cache. Set to 0 in order to make the maxclient value a soft limit if client pages are being paged out when there are sufficient free pages. Use in conjunction with the tuning of the maxperm and maxclient parameters.
strict_maxperm	If set to 1, the maxperm value will be a hard limit on how much of RAM can be used as a persistent file cache. Set to 1 in order to make the maxperm value a hard limit (use in conjunction with the tuning of the maxperm parameter).
v_pinshm	If set to 1, will allow pinning of shared memory segments. Change when there is too much overhead in pinning or unpinning of AIO buffers from shared memory segments. Tuning Useful only if application also sets SHM_PIN flag when doing a shmget call and if doing async I/O from shared memory segments.
vm_modlist_threshold	Determines whether to keep track of dirty file pages. Special values: -2: Never keep track of modified pages. This provides the same behavior as on a system prior to AIX 5.3. -1: Keep track of all modified pages. Other values: >= 0: Keep track of all dirty pages in a file if the number of frames in memory at full sync time is greater than or equal to vm_modlist_threshold. This parameter can be modified at any time, changing the behavior of a running system. In general, a new value will not be seen until the next full sync for the file. A full sync occurs when

the VW_FULLSYNC flag is used or all pages in the file (from 0 to maxvpn) are written to disk.

To display help for any particular tunable, you can use the -h flag, as in Example 5-16.

Example 5-16 Usnig vmo -h

```
r33n01:/ # vmo -h lgpg_regions
Help for tunable lgpg_regions:
Specifies the number of large pages to reserve for implementing with the shmget() system call
with the SHM_LGPAGE flag. Default: 0; Range: 0 - number of pages. lpgpg_size must also be used
in addition to this option. The application has to be modified to specify the SHM_LGPAGE flag
when calling shmget(). This will improve performance in the case where there are many TLB
misses and large amounts of memory is being
accessed.
```

Using the -L flag provides a very detailed report on the tunable specified and all of its values, as in Example 5-17.

Example 5-17 Using vmo -L

```
r33n01:/ # vmo -L minfree
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
------------------------------------------------------------------------------
minfree                   120    120    120    8      200K   4KB pages          D
     maxfree
     memory_frames
------------------------------------------------------------------------------
```

To change any of the tunables you would use the **-o** flag. To set a value at the next reboot requires the use of the -r flag. Some values also require that the bosboot command be run, and the value will take effect after next reboot following the running of the **bosboot** command.

Example 5-18 below turns memory affinity off, which is on by default.

Example 5-18 Changing vmo tunables

```
r33n01:/ # vmo -r -o memory_affinity=0
Setting memory_affinity to 0 in nextboot file
Warning: some changes will take effect only after a bosboot and a reboot
Run bosboot now? y

bosboot: Boot image is 22476 512 byte blocks.
Warning: changes will take effect only at next reboot
```

Not all tunables require a bosboot and a system reboot to set them. Check the help (wmo -h) for each option.

To tune the page replacement algorithm, you would make changes to the minperm, maxperm, minfree and maxfree tunables. To tune persistent file reads, you would make changes to the minpageahead and maxpageahead tunables. To tune persistent file writes, you would make changes to numclust, maxrandwrt, and sync_release_ilock.

Memory pools

The **vmo -o mempools=number_of_memory_pools** command allows you to change the number of memory pools that are configured at system boot time. The mempools option is therefore not a dynamic change. It is recommended to not change this value without a good understanding of the behavior of the system and the VMM algorithms. You cannot change the mempools value on a UP kernel and on an MP kernel, the change is written to the kernel file.

Reduce memory scanning overhead with lrubucket

Tuning with the *lrubucket* parameter can reduce scanning overhead on large memory systems. The page-replacement algorithm scans memory frames looking for a free frame. During this scan, reference bits of pages are reset, and if a free frame has not been found, a second scan is done. In the second scan, if the reference bit is still off, the frame will be used for a new page (page replacement).

On large memory systems, there may be too many frames to scan, so now memory is divided up into buckets of frames. The page-replacement algorithm will scan the frames in the bucket and then start over on that bucket for the second scan before moving on to the next bucket. The default number of frames in this bucket is 131072 or 512 MB of RAM. The number of frames is tunable with the command vmo -o lrubucket=new value, and the value is in 4 KB frames.

Values for minfree and maxfree parameters

On a large memory system the maxfree and minfree defaults are a very small percentage of real memory. If memory demand continues after the minfree value is reached, then processes may be killed.

The purpose of the free list is to keep track of real-memory page frames released by terminating processes and to supply page frames to requestors immediately, without forcing them to wait for page steals and the accompanying I/O to complete. The minfree limit specifies the free-list size below which page stealing to replenish the free list is to be started. The maxfree parameter is the size above which stealing will end.

The objectives in tuning these limits are to ensure that:

- Any activity that has critical response-time objectives can always get the page frames it needs from the free list.
- The system does not experience unnecessarily high levels of I/O because of premature stealing of pages to expand the free list.

  The default value of minfree and maxfree depend on the memory size of the machine. The default value of maxfree is determined by this formula:

  maxfree = minimum (# of memory pages/128, 128)

By default the minfree value is the value of maxfree - 8. However, the difference between minfree and maxfree should always be equal to or greater than maxpgahead. Or in other words, the value of maxfree should always be greater than or equal to minfree plus the size of maxpgahead. The minfree/maxfree values will be different if there is more than one memory pool. Memory pools were introduced in AIX 4.3.3 for MP systems with large amounts of RAM. Each memory pool will have its own minfree/maxfree which are determined by the previous formulas, but the minfree/maxfree values shown by the vmo command will be the sum of the minfree/maxfree for all memory pools.

Remember, that minfree pages in some sense are wasted, because they are available, but not in use. If you have a short list of the programs you want to run fast, you can investigate their memory requirements with the svmon command, and set minfree to the size of the largest. This technique risks being too conservative because not all of the pages that a process uses are acquired in one burst. At the same time, you might be missing dynamic demands that come from programs not on your list that may lower the average size of the free list when your critical programs run.

Values for minperm and maxperm parameters

The operating system takes advantage of the varying requirements for real memory by leaving in memory pages of files that have been read or written. If the file pages are requested again before their page frames are reassigned, this technique saves an I/O operation. These file pages may be from local or remote (for example, NFS) file systems.

The goals for maxperm and minperm is to find the appropriate value for maxperm to ensure that the systems favors filepages.

The ratio of page frames used for files versus those used for computational (working or program text) segments is loosely controlled by the minperm and maxperm values:

- If percentage of RAM occupied by file pages rises above maxperm, page-replacement steals only file pages.

- If percentage of RAM occupied by file pages falls below minperm, page-replacement steals both file and computational pages.
- If percentage of RAM occupied by file pages is between minperm and maxperm, page-replacement steals only file pages unless the number of file repages is higher than the number of computational repages.

In a particular workload, it might be worthwhile to emphasize the avoidance of file I/O. In another workload, keeping computational segment pages in memory might be more important. To understand what the ratio is in the untuned state, use the vmstat command with the -v option, as in Example 5-14 on page 315.

If you notice that the system is paging out to paging space, it could be that the file repaging rate is higher than the computational repaging rate since the number of file pages in memory is below the maxperm value. So, in this case we can prevent computational pages from being paged out by lowering the maxperm value to something lower than the numperm value.

Persistent file cache limit with the strict_maxperm option

The strict_maxpermoption of the **vmo** command, when set to 1, places a hard limit on how much memory is used for a persistent file cache by making the maxperm value be the upper limit for this file cache. When the upper limit is reached, the least recently used (LRU) is performed on persistent pages.

Attention: The strict_maxperm option should only be enabled for those cases that require a hard limit on the persistent file cache. Improper use of strict_maxperm can cause unexpected system behavior because it changes the VMM method of page replacement.

JFS2 file system cache limit with the maxclient parameter

The enhanced JFS file system uses client pages for its buffer cache, which are not affected by the maxperm and minperm threshold values. To establish hard limits on enhanced JFS file system cache, you can tune the maxclient parameter. This parameter represents the maximum number of client pages that can be used for buffer cache. To change this value, you can use the vmo -o maxclient command. The value for maxclient is shown as a percentage of real memory.

Example 5-19 shows how to tune the maximum number of client pages.

Example 5-19 Setting maxclient% using vmo -o

```
r33n01:/ # vmo -o maxclient%=75
Setting maxclient% to 75
```

After the maxclient threshold is reached, LRU begins to steal client pages that have not been referenced recently. If not enough client pages can be stolen, the

LRU might replace other types of pages. By reducing the value for maxclient, you help prevent Enhanced JFS file-page accesses from causing LRU to replace working storage pages, minimizing paging from paging space. The maxclient parameter also affects NFS clients and compressed pages. Also note that maxclient should generally be set to a value that is less than or equal to maxperm, particularly in the case where strict_maxperm is enabled.

Minimum memory requirement calculation

The formula to calculate the minimum memory requirement of a program is the following:

Total memory pages (4 KB units) = T + (N * (PD + LD)) + F

where:

T= Number of pages for text (shared by all users)

N = Number of copies of this program running simultaneously

PD = Number of working segment pages in process private segment

LD = Number of shared library data pages used by the process

F = Number of file pages (shared by all users)

Multiply the result by 4 to obtain the number of kilobytes required. You may want to add in the kernel, kernel extension, and shared library text segment values to this as well even though they are shared by all processes on the system. For example, some applications like databases use very large shared library modules.

5.2.2 Paging space thresholds tuning

If available paging space depletes to a low level, the operating system attempts to release resources by first warning processes to release paging space and finally by killing processes if there still is not enough paging space available for the current processes.

Values for the npswarn and npskill paramaters

The npswarn and npskill thresholds are used by the VMM to determine when to first warn processes and eventually when to kill processes.

These parameters can be set through the vmo command:

npswarn — Specifies the number of free paging space pages at which the operating system begins sending the SIGDANGER signal to processes. If the npswarn threshold is reached and a process is handling this signal, the process can

choose to ignore it or do some other action such as exit or release memory using the disclaim() subroutine. The value of npswarn must be greater than zero and less than the total number of paging space pages on the system. It can be changed with the command vmo -o npswarn=value.

npskill — Specifies the number of free paging space pages at which the operating system begins killing processes. If the npskill threshold is reached, a SIGKILL signal is sent to the youngest process. Processes that are handling SIGDANGER or processes that are using the early page-space allocation (paging space is allocated as soon as memory is requested) are exempt from being killed. The formula to determine the default value of npskill is as follows:

```
npskill = maximum (64, number_of_paging_space_pages/128)
```

The npskill value must be greater than zero and less than the total number of paging space pages on the system. It can be changed with the command vmo -o npskill=value.

nokillroot and nokilluid — By setting the nokillroot option to 1 with the command vmo -o nokillroot=1, processes owned by root will be exempt from being killed when the npskill threshold is reached. By setting the nokilluid option to a nonzero value with the command vmo -o nokilluid, user IDs lower than this value will be exempt from being killed because of low page-space conditions.

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.

5.3 Memory summary

This section contains some other useful commands for memory monitoring and tuning.

5.3.1 Other useful commands for memory performance

lsattr

Displays attribute characteristics and possible values of attributes for devices in the system.

Syntax

```
lsattr {-D[-O]| -E[-O] | -F Format [-Z Character]} -l Name [-a
Attribute]...[-H]
        [-f File]
lsattr {-D[-O]| -F Format [-Z Character]}{[-c Class][-s Subclass][-t Type]}
        [-a Attribute]... [-H][-f File]
lsattr -R {-l Name | [-c Class][-s Subclass][-t Type]} -a Attribute [-H]
        [-f File]
lsattr {-l Name | [-c Class][-s Subclass][-t Type]} -o Operation [...]
        -F Format [-Z Character][-f File][-H]
lsattr -h
```

To find out how much physical memory a system has, you can use the -E and -l flags.

Example 5-20 Using lsattr -El

```
r33n01:/ # lsattr -El mem0
goodsize 7168 Amount of usable physical memory in Mbytes False
size     7168 Total amount of physical memory in Mbytes  False
```

ipcs

The ipcs command reports status about active Inter Process Communication (IPC) facilities.

Syntax

```
ipcs [ - [ [ at ] | T ] bcmopqrsX [ [S1] | P ] [ -C corefile ] [ -N namelist ]
]
```

rmss

The rmss (Reduced Memory System Simulator) command is used to estimate the effects of reducing the amount of available memory on a system without having to physically remove memory.

Syntax

```
rmss [-s startmemsize] [-f finalmemsize] [-d deltamemsize]
            [-n numiterations] [-o outputfile] command
rmss -c memsize
rmss -r
```

```
rmss -p
```

Examples of rmss

To display the current memory size, use the -p flag.

Example 5-21 Using rmss -p

```
r33n01:/ # rmss -p
Simulated memory size is 7168 Mb.
```

To change the memory size, use the -c flag.

Example 5-22 Using rmss -c

```
r33n01:/ # rmss -c 2048
Simulated memory size changed to 2048 Mb.
r33n01:/ # rmss -p
Simulated memory size is 2048 Mb.
```

To reset the memory back to the real size, use the -r flag.

Example 5-23 Using rmss -r

```
r33n01:/ # rmss -r
Simulated memory size is 7168 Mb.
```

5.3.2 Paging space commands

The Virtual Memory Manager uses disk paging space as a temporary repository for processes that are not using active memory. Paging space performance is an important component of overall memory and system performance, thus we present the paging space related monitoring and tuning commands.

mkps

Adds additional paging space.

Syntax

```
mkps [-a] [-n] [-t lv] -s NumLPs Vgname Pvname
mkps [-a] [-n] -t nfs hostname pathname
```

chps

Used to change the attributes of defined paging spaces.

Syntax

```
chps [-s NewLPs | -d DecrLPs] [-a {y|n}] Psname
```

lsps

Displays the characteristics of defined paging spaces.

Syntax

```
lsps {-s | [-c | -l] {-a | Psname | -t {lv|nfs} } }
```

Example

The following Example 5-24 on page 332 shows how to list the paging spaces on a system.

Example 5-24 Using lsps

```
r33n01:/ # lsps -a
Page Space      Physical Volume   Volume Group    Size %Used Active  Auto  Type
hd6             hdisk0            rootvg          512MB     1   yes   yes    lv
r33n01:/ # lsps -s
Total Paging Space   Percent Used
      512MB               1%
```

rmps

Removes a paging space.

Syntax

```
rmps Psname
```

swapoff

Deactivates a paging space.

Syntax

```
swapoff DeviceName {DeviceName...}
```

swapon

Activates a paging space.

Syntax

```
swapon {-a | DeviceName...}
```

6

Network performance

In this chapter we discuss components that affect network performance and look at commands used to monitor and tune AIX network components.

This chapter covers:

- Factors that effect network performance
- Hardware considerations in a network environment
- AIX commands used for network monitoring
- AIX commands used for network tuning

6.1 Network overview

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.

Note: When tuning network buffers, indiscriminately using buffers that are too large can in fact reduce performance

Knowledge of you network topology is essential, this will help you find the 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 easily available.

TCP/IP protocol

Application programs transmit data over the network by making use of one of the 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 layer.

The Internet 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 by looking at the IP address of the destination) 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, such as Ethernet or token-ring networks. For more detailed information about the TCP/IP protocol, refer to *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909. To interpret the data created by programs such as the **iptrace** and **tcpdump** commands, formatted by **ipreport**, and summarized with **ipfilter**. For a diagram of the TCP/IP layers in AIX, see Figure 6-1 on page 335.

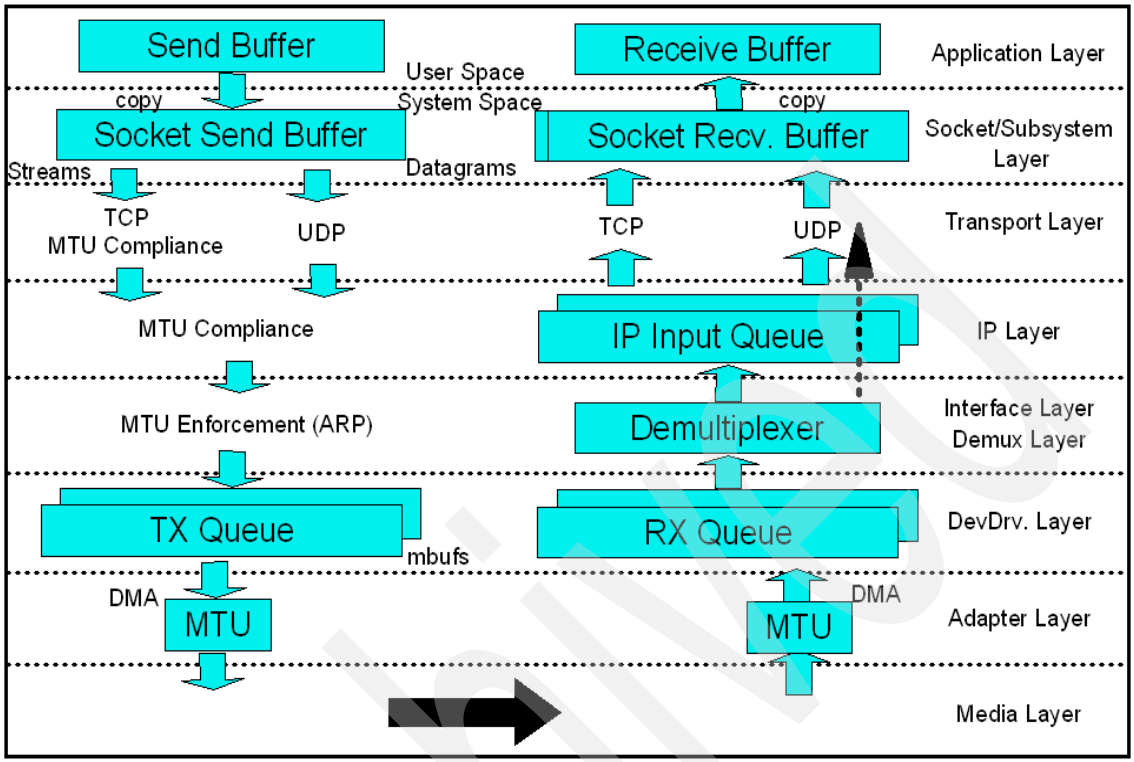

Figure 6-1 AIX TCP/IP communication model

In most cases you need to adjust some network tunables on server systems. Most of these settings effect different protocol buffers. You can set these buffer sizes system wide with the **no** command, or you can enable the Interface Specific Network Options (ISNO) option with the **no** command see “The Interface Specific Network Options (ISNO)” on page 415, and Example 6-1.

Example 6-1 Setting ISNO with the no command

```
[p630n04][/]> no -o use_isno=1
Setting use_isno to 1
[p630n04][/]>
```

By enabling **use_isno** option with **no**, will allow you to set buffer settings on a specific interface, giving you better control over performance management.

Network memory overview

The network subsystem uses a memory management facility that revolves around a data structure called an mbuf. Mbufs are mostly used to store data in

the kernel for incoming and outbound network traffic. Having mbuf pools of the right size can have a positive effect on network performance. If the mbuf pools are configured incorrectly, both network and system performance can suffer.

There are two tunables that can be used to define the upper limit for the amount of memory that can be used by the network subsystem. The **thewall** and **maxmbuf.**

The thewall tunable

AIX uses a network tunable called **thewall,** this defines the upper limit for network kernel buffers.

The size of thewall is defined at installation time and is based on ow much memory your machine has and type of kernel used. When running AIX 5L V5.3 running a 32 bit kernel is 1GB or half the size of real memory depending on which of the two is the smallest. If you have AIX 5L V5.3 running a 64bit kernel the size of **thewall** will be 65GB or half the size of real memory, depending on which of the two is smaller.

To display the size of the **thewall** value make use of the **no** command (see Example 6-2).

Example 6-2 Displaying the thewall value

```
[p630n02][/home/hennie]> no -o thewall
thewall = 1048576
[p630n02][/home/hennie]>
```

Attention: Take note that the size of **thewall** is static from AIX 5L Version 5.1 and later, and cannot be changed, to reduce the upper limit of memory used for networking make use of the **maxmbuf** tunable.

6.1.1 The maxmbuf tunable

The **maxmbuf** tunable used by AIX specifies the maximum amount of memory that can be used by the networking subsystem. This value can be displayed using the **lsattr** command as in Example 6-3.

Example 6-3 lsattr command to display sys0 attributes

```
[p630n02][/usr/include/sys]> lsattr -El sys0
SW_dist_intr    false               Enable SW distribution of interrupts             True
autorestart     true                Automatically REBOOT system after a crash        True
boottype        disk                N/A                                              False
capacity_inc    1.00                Processor capacity increment                     False
capped          true                Partition is capped                              False
```

```
conslogin       enable               System Console Login                              False
cpuguard        enable               CPU Guard                                         True
dedicated       true                 Partition is dedicated                            False
ent_capacity    4.00                 Entitled processor capacity                       False
frequency       484000000            System Bus Frequency                              False
fullcore        false                Enable full CORE dump                             True
fwversion       IBM,RG031014_d65e06_s Firmware version and revision levels             False
id_to_partition 0X036E80909F92EB01   Partition ID                                      False
id_to_system    0X036E80909F92EB01   System ID                                         False
iostat          false                Continuously maintain DISK I/O history            True
keylock         normal               State of system keylock at boot time              False
max_capacity    4.00                 Maximum potential processor capacity              False
max_logname     9                    Maximum login name length at boot time            True
maxbuf          20                   Maximum number of pages in block I/O BUFFER CACHE True
maxmbuf         0                    Maximum Kbytes of real memory allowed for MBUFS   True
maxpout         0                    HIGH water mark for pending write I/Os per file   True
maxuproc        128                  Maximum number of PROCESSES allowed per user      True
min_capacity    1.00                 Minimum potential processor capacity              False
minpout         0                    LOW water mark for pending write I/Os per file    True
modelname       IBM,7028-6C4         Machine name                                      False
ncargs          6                    ARG/ENV list size in 4K byte blocks               True
pre430core      false                Use pre-430 style CORE dump                       True
pre520tune      disable              Pre-520 tuning compatibility mode                 True
realmem         8388608              Amount of usable physical memory in Kbytes        False
rtasversion     1                    Open Firmware RTAS version                        False
systemid        IBM,0110685BF        Hardware system identifier                        False
variable_weight 0                    Variable processor capacity weight                False
[p630n02][/usr/include/sys]>
```

By default the **maxmbuf** tunable is disabled, it is set to 0, this means that the value of ***thewall*** will be used to define the maximum amount of memory used for network communications. By setting a non zero value to **maxmbuf** will override the value of **thewall**. This is the only way of reducing the value set by thewall.

To change the value of maxmbuf the chdev command can be used. In Example 6-4 the size of **maxmbuf** has been changed to 1 Gigabyte, the value of maxmbuf's are defined in 1Kbyte units.

Example 6-4 Change maxmbuf value with chdev

```
[p630n01][/]> chdev -l sys0 -a maxmbuf=1000000
sys0 changed
[p630n01][/]> lsattr -El sys0
```

The SMIT can also be used to change the **maxmbuf** attribute. To change the maxmbuf. Type **smitty system** -> **Change / Show Characteristics of Operating**

System. Here you can change the value **Maximum Kbytes of real memory allowed for MBUFS** (see Example 6-5).

Example 6-5 smitty screen to change maxmbuf value

```
                    Change / Show Characteristics of Operating System

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
  System ID                                            0X036E80909F92EB01
  Partition ID                                         0X036E80909F92EB01
  Maximum number of PROCESSES allowed per user         [128]                +#
  Maximum number of pages in block I/O BUFFER CACHE    [20]                 +#
  Maximum Kbytes of real memory allowed for MBUFS      [0]                  +#
  Automatically REBOOT system after a crash            true                 +
  Continuously maintain DISK I/O history               false                +
  HIGH water mark for pending write I/Os per file      [0]                   +#
  LOW water mark for pending write I/Os per file       [0]                   +#
  Amount of usable physical memory in Kbytes           8388608
  State of system keylock at boot time                 normal
  Enable full CORE dump                                false                +
  Use pre-430 style CORE dump                          false                +
[MORE...11]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

The sockthresh and strthresh tunables

The **sockthresh** and **strthresh** are tunables that limit the upper number for new sockets or TCP connections and new streams resource connections.

Sockets are used to store IP connection information, for every connection there is a socket associated with it. Sockets store the following information about each connection:

- Protocol used by the connection.
- Source address of the connection.
- Destination address of the connection.
- Source port number used by the connection.
- Destination port number.

To display information about sockets used on your system use the **netstat** command with the **-a** option as in Example 6-6.

Example 6-6 Output of netstat -a command

```
[p630n04][/home/hennie]> netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
........Line omited........
tcp        0      0  *.telnet               *.*                    LISTEN
tcp4       0      0  *.smtp                 *.*                    LISTEN
tcp4       0      0  *.time                 *.*                    LISTEN
tcp4       0      0  *.sunrpc               *.*                    LISTEN
tcp4       0      0  *.smux                 *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.login                *.*                    LISTEN
tcp        0      0  *.shell                *.*                    LISTEN
tcp4       0      0  *.klogin               *.*                    LISTEN
tcp4       0      0  *.kshell               *.*                    LISTEN
tcp4       0      0  *.rmc                  *.*                    LISTEN
tcp4       0      0  *.hp-manag             *.*                    LISTEN
tcp4       0      0  *.nim                  *.*                    LISTEN
tcp4       0      0  *.nimreg               *.*                    LISTEN
tcp4       0      0  *.writesrv             *.*                    LISTEN
tcp4       0      0  loopback.33749         loopback.33750         ESTABLISHED
tcp4       0      0  loopback.33750         loopback.33749         ESTABLISHED
tcp4       0      0  *.shilp                *.*                    LISTEN
tcp4       0      0  loopback.33749         loopback.34085         CLOSE_WAIT
tcp4       0      0  p630n01.ssh            tot191.itso.ibm..4929  ESTABLISHED
.............Lines omited............
```

The **sockthresh** tunable specifies the memory usage limit for socket connections. New socket connections are not allowed to exceed the value of the **sockthresh** tunable. The default value for the **sockthresh** tunable is 85%, once the total amount of allocated memory reaches 85% of the **thewall** or **maxmbuf** tunable value, the systems will not permit more socket connections, until the buffer usage drops below 85%.

Similarly, the **strthresh** tunable limits the amount of **mbuf** memory used for streams resources and the default value for the **strthresh** tunable is 85%. The **async** and TTY subsystems run in the streams environment. The **strthresh** tunable specifies that once the total amount of allocated memory reaches 85% of the thewall tunable value, no more memory goes to streams resources, to open streams, push modules or write to streams devices.

The **no** command can be used to set the percentage values of the **sockthresh** and **strthresh.**

To display the **sockthresh** and **strthresh** values use the no command as in Example 6-7.

Example 6-7 no command to display sockthresh

```
[p630n02][/]> no -o sockthresh -o strthresh
sockthresh = 85
strthresh = 85
[p630n02][/]>
```

6.2 Hardware considerations

When setting up a network it is very important to understand the role hardware plays in performance. Configuring your adapter or device correctly is important for optimal performance and stability.

Today almost all systems are shipped with on board network adapters ranging in speed from 10 Mbps to 1000 Mbps ethernet. Additional adapters can come in PCI 32 bit or PCI 64 bit, thus it is important to place these adapters in the correct slots for optimal performance. The way you interconnect all these adapters will have a big impact on you network performance.

There are a few factors that should be taken into account when connecting systems to a network:

- Firmware levels
- Media Speed
- MTU size

6.2.1 Firmware levels

Firmware (sometimes referred to as *microcode*) is code that is permanently loaded into the ROM (Read Only Memory) of an adapter or bus that enables the base functions of that device. Thus, keeping the firmware up to date especially on older systems is crucial to achieve optimal performance.

To display the firmware level of your system you can run the **lscfg -vp** command d see Example 6-8. This displays platform specific information and vital product data as it is found in the customized database (ODM - Object Data Manager).

Example 6-8 lscfg -v command

```
[node6][/]> lscfg -vp|grep -p ROM
.......lines omited.......
```

```
      10/100 Mbps Ethernet PCI Adapter II:
        Network Address.............0002554F5E8B
        ROM Level.(alterable).......SCU015
        Product Specific.(ZO).......A5204205
        Device Specific.(YL)........U0.1-P1/E2

      Platform Firmware:
        ROM Level.(alterable).......3R030501
        Version.....................RS6K
        System Info Specific.(YL)...U0.1-P1/Y1
      Physical Location: U0.1-P1/Y1

      System Firmware:
        ROM Level.(alterable).......RG030430_d54e07_sfw132
        Version.....................RS6K
        System Info Specific.(YL)...U0.1-P1/Y2
      Physical Location: U0.1-P1/Y2
.......lines omited..
```

You can see from the output in Example 6-8 on page 340 that the ROM level of the 10/100 Mbps adapter is on SCU015.

The latest firmware release information can be obtained from the following IBM link:

https://techsupport.services.ibm.com/server/mdownload/download.html

At this link you find a list of all the different hardware platforms and there related Firmware/Microcode levels. By clicking on the ***description*** button you will get information on how to update your firmware level.

6.2.2 Media speed considerations

Normally when you connect your system to a network it will by default try and detect the speed and duplex settings of the network. The adapter communicates with the devices on the other end of the cable (normally the with) to detect the speeds.

If you are setting up a point-point connection both ends should be setup to use Auto_Negotiation. This will have the adapters negotiate the speed and duplex rating to the highest possible speeds between the two adapters. If one of the adapters are not set to Auto_Negotiate then both adapters must be manually configured with the same settings.

This can be done by using **smitty -> Devices -> Communication -> Ethernet Adapter -> Adapter -> Change /Show Characteristics**, then you select the Media speed that should be used. The **chdev** command can also be used to

perform the same task. Make sure that both systems are setup at the same speed and duplex setting.

Ethernet adapters can be setup with the following modes:

- 10_Half_Duplex
- 10_Full_Duplex
- 100_Half_Duplex
- 100_Full_Duplex
- Auto_Negotiation

The same options apply for Gigabit Ethernet. To set the same values on the network switch, see the switch documentation.

To display the speed of the adapter use the **netstat -v** command as in Example 6-9.

Example 6-9 netstat -v command to display media speed

```
[p630n05][/]> netstat -v ent0 |grep Media
Media Speed Selected: Auto negotiation
Media Speed Running: 100 Mbps Full Duplex
[p630n05][/]>
```

The output of the above example displays that adapter ent0 is selected to connect with *Auto_negotiation* and is currently running with a Media speed of 100_Full_Duplex.

6.2.3 MTU size

When large amounts of data need to be transferred over a network, the data is packaged and transferred in a series of IP datagrams. The size of these packets is determined by the MTU (maximum transfer unit) size, this is the largest packet or frame that can be send over a network.

Different network adapters support different MTU sizes. The default MTU size for ethernet is 1500. Table 6-1 displays the default MTU sizes used by various network types.

Table 6-1 Default MTU sizes

Network	Default MTU size
16Mbit Token Ring	17914
4 Mbit Token Ring	4464
FDDI	4352

Network	Default MTU size
Ethernet	1500
Ethernet with Jumbo Frames enabled	9000
IEEE 802.3/802.2	1492
X.25	576
ATM	9180

All devices on the same physical or logical (VLAN) network should use the same MTU size.

The MTU size used within a network can have a large impact on performance depending on the workload type. Using larger MTU sizes on a network with large packet transfers will mean less packets, which in turn means less acknowledgements and better bandwidth utilization. But if applications use of smaller packets to transfer information, bigger MTU sizes will not increase the performance of your network.

When two hosts communicate over multiple networks, the packets can get fragmented if the interconnecting networks use smaller MTU sizes (specially in a WAN environment, the routers limit the MTU size to 572 bytes). This could put additional overhead on the gateways or bridges interconnecting these networks. This off course means reduced network performance. AIX supports path MTU (PMTU) discovery, as described in RFC1191. This means that AIX will chose the proper MTU size when sending packets outside the local network.

PMTU is enabled by default, the **no** command can be used to enable or disable **tcp_pmtu_discover** or **udp_pmtu_discover** options.

To display the current tcp_pmtu_discover setting use the **no -o tcp_pmtu_discover** command as in Example 6-10.

Example 6-10 tcp_pmtu_discover example

```
[p630n04][/]> no -o tcp_pmtu_discover
tcp_pmtu_discover = 1
```

To disable TCP PMTU use the same **no** command, as in Example 6-11.

Example 6-11 no command to disable tcp_mtu_discover

```
[p630n05][/]> no -o tcp_pmtu_discover=0
Setting tcp_pmtu_discover to 0
[p630n05][/]>
```

With Gigabit Ethernet you can also use the *Jumbo frames* option, which permits MTU sizes larger than 6000 bytes (default 9000). You need to enable jumbo frames when all the machines on the network use Gigabit Ethernet adapters, and also the switch must support this feature. The 10/100 Ethernet adapters do not support jumbo frames.

To enable Jumbo frames on a Gigabit Ethernet adapter, use the **chdev** command or **smit**. To enable jumbo frames both en* and et* interfaces should be disabled first otherwise the command will fail. You need to use the **chdev -l en* -a state=detach** command as in Example 6-12.

Example 6-12 The chdev command to detach an interface

```
[p630n05][/]> chdev -l en1 -a state=detach
en1 changed
[p630n05][/]>
```

Once the device is detached, use chdev or smitty to enable jumbo frames as in Example 6-13.

Example 6-13 Enabling jumbo_frames

```
[p630n04][/]> chdev -l ent1 -a jumbo_frames=yes
ent1 changed
[p630n04][/]>
```

You can also use SMIT: **smitty chgenet** (see Example 6-14).

Example 6-14 Enabling jumbo frames via SMIT

```
                Change / Show Characteristics of an Ethernet Adapter

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                  [Entry Fields]
  Ethernet Adapter                               ent1
  Description                                    10/100/1000 Base-TX P>
  Status                                         Available
  Location                                       1Z-08
  Rcv descriptor queue size                      [1024]                  +#
  TX descriptor queue size                       [512]                   +#
  Software transmit queue size                   [8192]                  +#
  Transmit jumbo frames                          yes                     +
  Enable hardware TX TCP resegmentation          yes                     +
  Enable hardware transmit and receive checksum  yes                     +
  Media speed                                    Auto_Negotiation        +
  Enable ALTERNATE ETHERNET address              no                      +
```

```
  ALTERNATE ETHERNET address                          [0x000000000000]        +
  Apply change to DATABASE only                        no                     +

F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

After you have enabled jumbo frames, the en* interface needs to be enabled again as in Example 6-15.

Example 6-15 Using chdev to enable en1

```
[p630n05][/]> chdev -l en1 -a state=up
en1 changed
[p630n05][/]>
```

We ran some tests to measure the throughput increase when using jumbo frames. We use two methods to transfer an 8GB file.

First, we check the MTU size using **netstat -in** command as in Example 6-16:

Example 6-16 netstat -in to check mtu size

```
[p630n05][/]> netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
en0   1500  link#2      0.2.55.4f.cb.f5   2705991     0  4277041     0     0
en0   1500  192.168.100 192.168.100.35    2705991     0  4277041     0     0
en1   1500  link#3      0.2.55.53.b1.95  35874354     0  8753187     4     0
en1   1500  10.1.1      10.1.1.5         35874354     0  8753187     4     0
en2   1500  link#4      0.2.55.4f.cb.f4    649378     0       10     0     0
en2   1500  172.16.100  172.16.100.35      649378     0       10     0     0
lo0   16896 link#1                          150     0      307     0     0
lo0   16896 127         127.0.0.1          150     0      307     0     0
lo0   16896 ::1                            150     0      307     0     0
[p630n05][/]>
```

The interface we use for out test is **en1**. As can be seen from the output, the second column of the netstat -in command displays the current MTU size, which is currently set to 1500 bytes.

The first test was to ftp a large file from one system to another. Using dd in conjunction with the **/dev/zero** and **/dev/null** files will make sure that disk I/O doesn't effect our tests. The syntax of **dd** is as follows.

```
dd [if=InputFile] [of=OutputFile] [cbs=Number] [fskip=Number] [skip=Number]
[seek=Number] [count=Number] [bs=Number] [span=yes|no] [ibs=Number]
[obs=Number] [files=Number] [conv=Parameter[, ...]]
```

Using **dd** in combination with **ftp** allows us to test the network with virtually any file size. In Example 6-17 we show a 8GB transfer via ftp (1,000,000 blocks of 8k).

Example 6-17 Using dd to ftp a large file

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=8k count=1000000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
1000000+0 records in.
1000000+0 records out.
226 Transfer complete.
8192000000 bytes sent in 70.43 seconds (1.136e+05 Kbytes/s)
local: |dd if=/dev/zero bs=8k count=1000000 remote: /dev/null
ftp>
ftp>  put "|dd if=/dev/zero bs=8k count=1000000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
1000000+0 records in.
1000000+0 records out.
226 Transfer complete.
8192000000 bytes sent in 70.4 seconds (1.136e+05 Kbytes/s)
local: |dd if=/dev/zero bs=8k count=1000000 remote: /dev/null
ftp>
```

Running the test twice, the results we obtained were **70.43** and **70.4** seconds respectively.

For the second phase, we enabled jumbo frames (see Example 6-18):

- Detach interface
- Enable jumbo frames on the adapter
- Re-activate the interface

Example 6-18 Enabling jumbo frames

```
[p630n05][/]> chdev -l ent1 -a jumbo_frames
ent1 changed
[p630n05][/]> chdev -l ent1 -a jumbo_frames=yes
ent1 changed
[p630n05][/]>
[p630n05][/]> chdev -l en1 -a state=up
en1 changed
[p630n05][/]>
```

Make sure that all the other systems connected to the same switch (VLAN) have **jumbo_frames** enabled, and the switch supports jumbo frames.

To verify the new MTU size, use **lsattr -El en*** (see Example 6-19), or **netstat -in**.

Example 6-19 Attribute information of interface en1

```
[p630n05][/]> lsattr -El en1
alias4                      IPv4 Alias including Subnet Mask           True
alias6                      IPv6 Alias including Prefix Length         True
arp           on            Address Resolution Protocol (ARP)          True
authority                   Authorized Users                           True
broadcast                   Broadcast Address                          True
mtu           9000          Maximum IP Packet Size for This Device     True
netaddr       10.1.1.5      Internet Address                           True
netaddr6                    IPv6 Internet Address                      True
netmask       255.255.255.0 Subnet Mask                                True
prefixlen                   Prefix Length for IPv6 Internet Address    True
remmtu        1500          Maximum IP Packet Size for REMOTE Networks True
rfc1323       1             Enable/Disable TCP RFC 1323 Window Scaling True
security      none          Security Level                             True
state         up            Current Interface Status                   True
tcp_mssdflt   1448          Set TCP Maximum Segment Size               True
tcp_nodelay                 Enable/Disable TCP_NODELAY Option          True
tcp_recvspace 131072        Set Socket Buffer Space for Receiving      True
tcp_sendspace 131072        Set Socket Buffer Space for Sending        True
[p630n05][/]>
```

We ran the test again with the **dd** command (see Example 6-20).

Example 6-20 Running test with MTU size 9000

```
ftp> bin
200 Type set to I.
ftp> put "|dd if=/dev/zero bs=8k count=1000000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
1000000+0 records in.
1000000+0 records out.
226 Transfer complete.
8192000000 bytes sent in 66.97 seconds (1.195e+05 Kbytes/s)
local: |dd if=/dev/zero bs=8k count=1000000 remote: /dev/null
ftp>
ftp>put "|dd if=/dev/zero bs=8k count=1000000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
1000000+0 records in.
1000000+0 records out.
```

```
226 Transfer complete.
8192000000 bytes sent in 66.94 seconds (1.195e+05 Kbytes/s)
local: |dd if=/dev/zero bs=8k count=1000000 remote: /dev/null
ftp>
```

As you can see from the output again the results are **66.97** and **66.94** seconds respectively. By comparing the output of the two tests we can see that there is a performance gain of 5%.

6.3 Network monitoring

AIX offers a wide range of tools to monitor networks, including network adapters, network interfaces, and system resources used by the network software. These tools are covered in this chapter. 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.

System load

Poor system performance may not necessarily come from a network problem. In case your system is short on local resources, such as CPU or memory, you may start performance problem resolution with these subsystems. For details, refer to Chapter 4, “CPU analysis and tuning” on page 171, and Chapter 5, “Memory analysis and tuning” on page 297.

Gathering information

Gathering configuration information from the server and client systems and keeping a soft copy of the information is important, as 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 can be very difficult. A very useful command for gathering a snapshot of the system information is **snap -a.** The snap command has the following syntax:

```
snap [ -a ] [ -A ] [ -b ] [ -c ] [ -D ] [ -f ] [ -g ] [ -G ] [ -i ] [ -k ] [ -l
] [ -L ][ -n ] [ -N ] [ -p ] [ -r ] [ -s ] [ -S ] [ -t ] [ -T ] [ -w ] [ -o
OutputDevice ] [ -d Dir ] [ -v Component ]
```

Example 6-21 The snap -a command

```
[p630n01][/]snap -a
Checking space requirement for general information..... done.
```

```
Checking space requirement for tcpip information...................done
Checking space requirement for nfs information.............. done.
Checking space requirement for kernel information.............. done.
Checking space requirement for printer information.... done.
Checking space requirement for dump information......Checking space.....

........lines omited............
```

The **snap -a** command (see Example 6-21 on page 348) collects configuration information about your whole system and stores it in the **/etc/ibmsupt** directory. If you want to specify a different directory use the **-d** option.

Note: Make sure you have enough space in the directory you plan to store the configuration data.

To view TCP/IP specific information gathered by **snap**, see the **/tmp/ibmsupt/tcpip** directory. This directory contains configuration information about the TCP/IP subsystem stored in a file named **tcpip.snap**, which contains output of the following commands:

- **lssrc -a**
- **netstat -m**
- **netstat -in**
- **netstat -v**
- **netstat -s**
- **netstat -an**
- **netstat -sr**
- **arp -a**
- **arp -t atm -a**
- **uname -xM**

The commands used by **snap -a** are also useful for monitoring your system. We discuss some of these commands in more detail further in this chapter.

6.3.1 Creating network load

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. For network analysis, you may have use additional tools, such as sniffers, network analyzers etc.

However, tools such as **ping** or **traceroute** can be used to gather turnaround times for data on the network, to test if hosts availability on the network and

whether or not the correct routes are being used to communicate with remote hosts.

A good method for testing the network is to create some load to simulate real traffic over the network. We used two methods for creating “artificial” load. The first one will be to **ftp** large chunks of data using the **dd** command (see Example 6-17 on page 346). The other will be creating a *pipe* file and then using **rsh** with **dd** to transfer data over the network.

To create network load using **ftp**, in this example we will transfer 10GB of data using the **/dev/zero** and **/dev/null** files. These files are used so that disk I/O operation is not effected and disk does not become a bottleneck.

Testing using transfers using a FIFO file.

A pipe file can also be used to generate network load. Use the **mknod** command to create a FIFO (named pipelines) file. See Example 6-22.

In our test example we are going to test our network between two systems via the same Gbit Ethernet network. The IP labels for the two Gbit adapters are gp01 and gp05.

Example 6-22 Creating a pipe file (FIFO) with mknod

```
[gp01][/tmp]> mknod fifo p
[gp01][/tmp]> ls -l fifo
prw-r--r--   1 root     system            0 Oct 14 16:37 fifo
[gp01][/tmp]>
```

Once the pipe file has been created on host gp01, you can use the **dd** command to write data to this file (see Example 6-23). The command will wait until data will be extracted (read) from the pipe with another command, or interrupted with **Ctrl-C**.

Example 6-23 Using dd with a pipe file on

```
[gp01][/tmp]> dd if=/dev/zero bs=10k count=1000000 of=/tmp/fifo
```

On the remote node gp05, use the **rsh** command to connect to remote node gp01, and read data from the pipe with **dd**, as in Example 6-24.

Example 6-24 Extracting data from pipe on node gp05

```
[gp05][/]> timex rsh gp01  "dd if=/tmp/fifo bs=8k"|dd of=/dev/null bs=8k
1250000+0 records in.
1250000+0 records out.

real 83.72
```

```
user 8.64
sys  40.59

184+2499632 records in.
184+2499632 records out.
[gp05][/].
```

We used the **timex** command to measure the time it takes to transfer the data.

6.4 Network monitoring commands

This section presents the network monitoring commands, with usage examples, and useful combinations. Some of the commands may also be used for monitoring other system subsystems, but we emphasize the network related aspects.

6.4.1 The entstat command

The **entstat** command displays ethernet device driver and device statistics.

Syntax:

```
entstat [ -drt ] Device_Name
```

Useful options

entstat -d	This option will display interface as well device driver information
entstat -r	This option resets statistics collected by **entstat**

Description

The **entstat** command displays the statistics gathered by the specified Ethernet device driver. The user can optionally specify that the device-specific statistics be displayed in addition to the device generic statistics. If no flags are specified, only the device generic statistics are displayed.

The **entstat** command is part of the devices.common.IBM.ethernet.rte fileset and the path of the executable is /usr/bin/entstat

Examples

When using the **entstat** command you must specify the ethernet device to check. See Example 6-25 on page 352 shows detailed output of the ent1 device driver and communication statistics.

Example 6-25 The entstat - d ent1

```
[p630n04][/]> entstat -d ent1
-------------------------------------------------------------
ETHERNET STATISTICS (ent1) :
Device Type: 10/100/1000 Base-TX PCI-X Adapter (14106902)
Hardware Address: 00:02:55:53:b1:88
Elapsed Time: 0 days 1 hours 40 minutes 37 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 2260                                 Packets: 3309
Bytes: 583730                                 Bytes: 255115
Interrupts: 0                                 Interrupts: 3297
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0

Max Packets on S/W Transmit Queue: 7
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 2135                       Broadcast Packets: 2203
Multicast Packets: 0                          Multicast Packets: 0
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: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 0                     Receiver Start Count: 0
Multiple Collision Count: 0
Current HW Transmit Queue Length: 1

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 2000
Driver Flags: Up Broadcast Running
        Simplex 64BitSupport ChecksumOffload
        PrivateSegment LargeSend DataRateSet

10/100/1000 Base-TX PCI-X Adapter (14106902) Specific Statistics:
---------------------------------------------------------------------
Link Status : Up
Media Speed Selected: Auto negotiation
Media Speed Running: 1000 Mbps Full Duplex
```

```
PCI Mode: PCI-X (100-133)
PCI Bus Width: 64-bit
Latency Timer: 144
Cache Line Size: 128
Jumbo Frames: Enabled
TCP Segmentation Offload: Enabled
TCP Segmentation Offload Packets Transmitted: 3
TCP Segmentation Offload Packet Errors: 0
Transmit and Receive Flow Control Status: Disabled
Transmit and Receive Flow Control Threshold (High): 24576
Transmit and Receive Flow Control Threshold (Low): 16384
Transmit and Receive Storage Allocation (TX/RX): 24/40
```

The following list contains short descriptions of the bolded fields in Example 6-25 on page 352:

Device Type Displays the description of the adapter type.

Hardware Address Displays the Ethernet network address currently used by the device.

Elapsed Time Displays the real time period which has elapsed since last time the statistics were reset. Part of the statistics may be reset by the device driver during error recovery when a hardware error is detected. There will be another Elapsed Time displayed in the middle of the output when this situation has occurred in order to reflect the time differences between the statistics.

Transmit Statistics Fields

Packets The number of packets transmitted successfully by the device.

Bytes The number of bytes transmitted successfully by the device.

Transmit Errors The number of output errors encountered on this device. This is a counter for unsuccessful transmissions due to hardware/network errors.

Packets Dropped The number of packets accepted by the device driver for transmission which were not (for any reason) given to the device.

S/W Transmit Queue Overflow
The number of outgoing packets which have overflowed the software transmit queue.

No Carrier Sense The number of unsuccessful transmissions due to the no carrier sense error.

Single Collision Count
The number of outgoing packets with single (only one) collision encountered during transmission.

Multiple Collision Count
The number of outgoing packets with multiple (2 - 15) collisions encountered during transmission.

Current HW Transmit Queue Length
The number of outgoing packets which currently exist on the hardware transmit queue.

No Resource Errors The number of incoming packets dropped by the hardware due to the resource error. This error usually occurs because the receive buffers on the adapter were exhausted. Some adapters may have the size of the receive buffers as a configurable parameter. Check the device configuration attributes for possible tuning information.

Receive Collision Errors
The number of incoming packets with the collision errors during the reception.

General Statistics Fields

No mbuf Errors The number of times that 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. The **`netstat -m`** command should be used to confirm this.

Adapter Reset Count
The number of times that the adapter has been restarted (re-initialized).

Device Specific Statistics Fields

Link Status The state of the interface at this time.

Media Speed Selected
The speed at which the adapter should connect to the network the default is auto-negotiate. Options are. 10_Half_Duplex, 10_Full_Duplex, 100_Half_Duplex, 100_Full_Duplex,Auto_Negotiation. Use **`chdev`** or SMIT to change these values.

Media Speed Running
: The speed at which the adapter is connected to the network.

Jumbo Frames
: Specifies if Jumbo frames is enabled or not, this option is only available for Gigabit Ethernet.

An increasing number of collisions could be caused by too much load on the subnetwork. It may be necessary to split the sub-net into two or more smaller subnets in a case like this. If making use of switches it is unlikely that you will see any collisions.

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** to reset the counters to zero, then run the command. After the completion of the application or task, run **entstat** again to get this information as in Example 6-26. In this example the **entstat** counters are reset (set to zero), and the **ping** command is used with the **-f** (flood) option. When **ping -f** is stopped, use again the **entstat** command to report any errors.

Example 6-26 entstat report while executing a command

```
[p630n01][/]> entstat -r ent0; ping -f p630n05 64 2048; entstat ent0
```

Incorporating the **sleep** command with **entstat** it is possible to record **entstat** information over a period of time as in Example 6-27.

Example 6-27 entstat report over a period of time

```
[p630n01][/]> entstat -r ent0;sleep 300;entstat ent0
```

The numbers of packets, bytes, and broadcasts transmitted and received depend on several factors, like the applications running on the system, or the number of systems connected to the same physical network. 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, tuning the adapter parameters using the **chdev** command, or tuning network options using the **no** command.

6.4.2 The netstat command

The **netstat** command is a tool that displays network statistics. It is used for analyzing the system network stack, and to display information about the network traffic, the amount of data send and received by each protocol, and memory usage for network buffers.

The netstat command is a symbolic link to the /usr/sbin/netstat command and is part of the **bos.net.tcp.client** fileset.

syntax:

netstat [-Aan] [-f address_family] [core unix netinet addr]

[-D]

[-cCgimnrsPv] [-f address_family] [-p proto] [core unix netinet addr]

[-n] [-I interface] [interval] [core unix netinet addr]

Useful options

netstat -v Displays the same output as **entstat -d**. Refer to Example 6-25 on page 352.

netstat -in Displays network interface information related to maximum transmission Unit (MTU) sizes, packets received and transmitted, and errors received and transmitted.

netstat -rn Displays routing information associated with the different interfaces your system has connected. Information about the path mtu, amount of times a particular route has been used.

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. The **extendednetstats** is set to 0 (zero) by default.

netstat -s The output of this command shows detailed statistics for ALL THE protocols used. This includes packets sent and received, packets dropped, and error counters. The **netstat -p** command can be used to display the information for a specific protocol. This is useful if you are only interested in the statistics for a particular protocol, for example Transmission control protocol (TCP). Using the **netstat -p tcp** command

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.

netstat examples

The first example we look at is the **netstat -v ent0** (Example 6-28) command. This will display device driver information that gets extracted from the **entstat** command. You will see in the output that the command gives you the exact output as if you were to run the **entstat -d ent0** command.

Example 6-28 netstat -v ent0

```
[p630n04][/]> netstat -v ent0
-------------------------------------------------------------
ETHERNET STATISTICS (ent0) :
Device Type: 10/100 Mbps Ethernet PCI Adapter II (1410ff01)
Hardware Address: 00:02:55:4f:d6:74
Elapsed Time: 0 days 1 hours 45 minutes 46 seconds

Transmit Statistics:                          Receive Statistics:
--------------------                          -------------------
Packets: 4131                                 Packets: 27064
Bytes: 535698                                 Bytes: 3360732
Interrupts: 2                                 Interrupts: 26548
Transmit Errors: 0                            Receive Errors: 0
Packets Dropped: 0                            Packets Dropped: 0
                                              Bad Packets: 0

Max Packets on S/W Transmit Queue: 4
S/W Transmit Queue Overflow: 0
Current S/W+H/W Transmit Queue Length: 1

Broadcast Packets: 64                         Broadcast Packets: 20754
Multicast Packets: 2                          Multicast Packets: 6
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: 0
Deferred: 0                                   Packet Too Short Errors: 0
SQE Test: 0                                   Packet Too Long Errors: 0
Timeout Errors: 0                             Packets Discarded by Adapter: 0
Single Collision Count: 0                     Receiver Start Count: 0
Multiple Collision Count: 0
```

```
Current HW Transmit Queue Length: 1

General Statistics:
-------------------
No mbuf Errors: 0
Adapter Reset Count: 0
Adapter Data Rate: 200
Driver Flags: Up Broadcast Running
        Simplex AlternateAddress 64BitSupport
        ChecksumOffload PrivateSegment LargeSend
        DataRateSet

10/100 Mbps Ethernet PCI Adapter II (1410ff01) Specific Statistics:
--------------------------------------------------------------------
Link Status : up
Media Speed Selected: Auto negotiation
Media Speed Running: 100 Mbps Full Duplex
Receive Pool Buffer Size: 1024
Free Receive Pool Buffers: 1012
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: 12
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
[p630n04][/]>
```

Refer to the **entstat** command for more detail (Example 6-25 on page 352).

The **netstat -in** command can be used to display network interface statistics as in Example 6-29.

Example 6-29 netstat -in

```
[p630n04][/]> netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
en0   1500  link#2      0.2.55.4f.d6.74     2875     0      830     0     0
en0   1500  192.168.100 192.168.100.34      2875     0      830     0     0
en1   9000  link#3      0.2.55.53.b1.88        0     0        3     3     0
```

```
en1   9000  10.1.1      10.1.1.4                0     0        3     3     0
lo0   16896 link#1                           5974     0     7554     0     0
lo0   16896 127         127.0.0.1             5974     0     7554     0     0
lo0   16896 ::1                               5974     0     7554     0     0
[p630n04][/]>
```

The fields displayed are:

name This field displays the name of the interface of which statistics will be displayed. Only interfaces that are currently up will be displayed.

MTU This field displays the interface MTU size used. From the output of Example 6-29 on page 358 you will note that interface en1 are using an MTU of 9000 which means the interface has jumbo_frames enabled.

Network This field displays the network address of the network that the interface is connected to.

Address This field displays the adapter hardware address and interface IP address.

Ipkts This filed displays the count of packets received by the interface.

Ierrs This field displays a count of the errors received by the interface.

Opkts This field displays a count of packets transmitted from this interface.

Oerrs This field displays a count of error packets generated from this interface.

Coll This field displays a count of the collisions occurred on this adapter. The collision count for ethernet is not supported.

When running the **netstat -in** command check that all network interfaces of systems on the same networks have the same network address. The MTU size of systems on the same physical network or VPN must be the same. The Ierrs and Oerrs, should always be zero, if not the network hardware and interfaces should be checked for problems. On ethernet the collision field is not supported and will always display 0 (zero).

The **netstat -rn** command will display routing information a your systems in Example 6-30.

Example 6-30 netstat -rn command

```
[p630n04][/]> netstat -rn
Routing tables
Destination      Gateway           Flags   Refs     Use  If   PMTU Exp Groups

Route Tree for Protocol Family 2 (Internet):
default          192.168.100.60    UG        1      831  en0     -   -
```

```
10.1.1.0         10.1.1.4          UHSb      0        0  en1     -   -  =>
10.1.1/24        10.1.1.4          U         0      188  en1     -   -
10.1.1.4         127.0.0.1         UGHS      0      269  lo0     -   -
10.1.1.255       10.1.1.4          UHSb      0      219  en1     -   -
127/8            127.0.0.1         U         7      883  lo0     -   -
192.168.100.0    192.168.100.34    UHSb      0        0  en0     -   -  =>
192.168.100/24   192.168.100.34    U         4      203  en0     -   -
192.168.100.34   127.0.0.1         UGHS      0      489  lo0     -   -
192.168.100.255  192.168.100.34    UHSb      0       88  en0     -   -

Route Tree for Protocol Family 24 (Internet v6):
::1              ::1               UH        0       16  lo0     -   -
[p630n04][/]>
```

The fields displayed are:

Destination — This field displays the destination address of either a host or network for a specific route. Normally there will be, this is the route that will be used by your system if no route is specifically defined for a destination.

Gateway — This field displays the gateway that will be used to connected to a defined destination.

Flags — This field displays the state and type of route a list of possible options is:

A - An Active Dead Gateway Detection is enabled on the route. This field only applies to AIX 5.1 or later.

U - The route is Up.

H - The route is to a host rather than to a network.

G - The route is to a gateway.

D - The route was created dynamically by a redirect.

M - The route has been modified by a redirect.

L - The link-level address is present in the route entry.

c - Access to this route creates a cloned route.

W - The route is a cloned route.

1 - Protocol specific routing flag #1.

2 - Protocol specific routing flag #2.

3 - Protocol specific routing flag #3.

b - The route represents a broadcast address.

e - Has a binding cache entry.

l - The route represents a local address.

m - The route represents a multicast address.

P - Pinned route.

R - Host or net unreachable.

S - Manually added.

u - Route usable.

s - The Group Routing *stopsearch* option is enabled on the route.

Refs — This field displays the current number of active uses for the route. Connection-oriented protocols hold on to a single route for the duration of a connection, while connectionless protocols obtain a route while sending to the same destination.

Use — This field displays a count of number of packets sent making use of this route.

If — This field displays a count of the network interface utilization for this route.

PMTU — This field displays the path MTU size for the for this route. AIX 5.3 does not display a value for this field. See the "The pmtu command" on page 370.

Note: In AIX 5.3 the PMTU field does not display any information with the **netstat -rn** command, the **pmtu** command should be used to display or delete PMTU values.

Exp — This field displays the time in minutes before this route expires.

Groups — This field displays a list of group id's associated with this route.

The various layers of the communication subsystem share common buffer pools called 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 -El sys0 -a maxmbuf** to control the current value set, and **lsattr -Rl 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 activate.

If **maxmbuf** in the system configuration is zero, then the network option **thewall** defines the maximum amount of memory to be used. The thewall value is a static value in AIX 5.3 and cannot be changed. You can only use the mamxbuf attribute to manage the size of the mbuf pool.

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 as in Example 6-31.

Example 6-31 netstat -m command

```
[p630n06][/]> netstat -m

Kernel malloc statistics:

******* CPU 0 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
32                  108       136      0         0      20   10484       0
64                  133      8947      0         1      59   10484       0
128                 138     39544      0         2      86    5242       0
256                 153  14089423      0         8   10471   10484    5726
512                3271  19217932      0       469     505   13105       0
1024                127      5326      0        32      21    5242       0
2048               2098      9834      0      2054    2018    7863       0
4096                170      7668      0        42    1598    2621       0
8192                  4       282      0         8       3    1310       0
16384              2560      7014      0       723       8     655    2588
32768                 0        31      0         1       4     327       0
65536                 2         3      0         2       0     327       0
131072                0         0      0         0     204     409       0


******* CPU 1 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
32                    2         7      0         0     126   10484       0
64                  105      5260      0         2      23   10484       0
128                  54     26973      0         0      42    5242       0
256                   7   9657550      0         0    6297   10484       0
......lines ommitted..........
```

The **netstat -m** command displays mbuf usage per CPU (see Example 6-32 on page 363). By enabling the option **extendednetstats** with the **no** command, the system will display detailed output. The **extendednetstats** option of the **no** command is defined as a reboot option (you have to reboot the system for this option to become active after a change).

Example 6-32 netstat -m example

```
[p630n04][/]> netstat -m
3309 mbufs in use:
3236 mbuf cluster pages in use
14598 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
32                   41      7507      0         1      87    2620       0
64                   81       436      0         1      47    2620       0
128                  52     12271      0         0      44    1310       0
256                   7     17693      0         1      73    2620       0
512                  71    275975      0       320    2489    3275       0
1024                 11       183      0         1       5    1310       0
2048                  1      2079      0      1023    1963    1965      44
4096                  3       184      0        41     214     655       0
8192                  5        20      0         3      12     327       0
16384                 0      1952      0       245       0     163    1796
131072                0         0      0         0      86     102     172

..........lines omited CPU 1 2 and 3

By type          inuse   calls failed delayed  memuse  memmax  mapb
mbuf              3309  1812160      0     883 1694208 2964992    12
mcluster          3236   606715      0    3332 36308992 38930432   252
socket             284   116948      0      15   98752  102496     0
pcb                100    57464      0       0   11200   16704     0
routetbl            36    34951      0       0    5120    7456     0
fragtbl              0        9      0       0       0      32     0
ifaddr              45       43      0       2    7456    7456     0
mblk                19    64109      0       0    2688   20480     0
mblkdata            46      596      0       6   65536   67584     0
strhead             31       77      0       0    9632    9632     0
strqueue            49      174      0       2   25088   25088     0
strmodsw            22        4      0       0    1408    1408     0
strosr               0     8315      0       0       0     256     0
strsyncq            56      446      0       1    6624    6688     0
streams            182     2893      0       1   28928   28928     0
devbuf            1538     8711      0    1698 10486048 12583200    41
kernel tablemoun    59      112      0       2  156320  160416     0
spec buf             1        0      0       0     128     128     0
locking             90       89      0       6   23040   23040     0
temp                28   109653      0       1   26496   30720     0
```

```
mcast opts            0         4       0       0       0      32      0
mcast addrs           3         3       0       0     192     192      0

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
[p630n04][/]>
```

If the request for mbufs field is non zero, this is a good indication that the maxmbuf attribute needs to be increased. See Example 6-4 on page 337.

If the “sockets not created because sockthresh was reached” field is non-zero, the **sockthresh** attribute should be increased with the **no** command. See 6.7.1, “The no command” on page 396.

Note: If you increase the maxmbuf attribute, this will automatically allow more space for sockets, as sockthresh is a percentage value of the maxmbuf or thewall attributes.

From the output of the **netstat -m** command with *extendednetstats* enable you should note that additional information gets displayed at the end of the normal CPU report. This gives detailed information on memory utilization and mbufs used by the networking subsystem. The **netstat -p tcp** displays detailed information about the tcp protocol (see Example 6-33).

Example 6-33 netstat -p to monitor tcp

```
[p630n04][/home/hennie]> netstat -p tcp
tcp:
        147575 packets sent
                93370 data packets (2160063890 bytes)
                143 data packets (178440 bytes) retransmitted
                5196 ack-only packets (4278 delayed)
                0 URG only packets
                0 window probe packets
                48850 window update packets
                98 control packets
                65622 large sends
                2157856140 bytes sent using largesend
                64240 bytes is the biggest largesend
        334159 packets received
                203334 acks (for 2160063991 bytes)
                792 duplicate acks
                0 acks for unsent data
                204227 packets (169162567 bytes) received in-sequence
                51 completely duplicate packets (88 bytes)
```

```
        0 old duplicate packets
        0 packets with some dup. data (0 bytes duped)
        709 out-of-order packets (54816 bytes)
        0 packets (0 bytes) of data after window
        0 window probes
        8 window update packets
        1 packet 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
        10 discarded by listeners
        0 discarded due to listener's queue full
        125978 ack packet headers correctly predicted
        129051 data packet headers correctly predicted
53 connection requests
38 connection accepts
60 connections established (including accepts)
987 connections closed (including 18 drops)
0 connections with ECN capability
0 times responded to ECN
0 embryonic connections dropped
80458 segments updated rtt (of 80541 attempts)
0 segments with congestion window reduced bit set
0 segments with congestion experienced bit set
0 resends due to path MTU discovery
3 path MTU discovery terminations due to retransmits
88 retransmit timeouts
        1 connection dropped by rexmit timeout
27 fast retransmits
        0 when congestion window less than 4 segments
7 newreno retransmits
0 times avoided false fast retransmits
0 persist timeouts
        0 connections dropped due to persist timeout
86 keepalive timeouts
        83 keepalive probes sent
        3 connections 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
[p630n04][/home/hennie]>
```

The statistics of interest are:

- Packets Sent and Data Packets
- Data Packets Retransmitted
- Packets Received
- Completely Duplicate Packets
- Retransmit Timeouts

For the TCP statistics, compare the number of packets sent to the number of data packets retransmitted. If the number of packets retransmitted is over 10-15 percent of the total packets sent, TCP is experiencing timeouts indicating that network traffic may be too high for acknowledgments (ACKs) to return before a timeout. A bottleneck on the receiving node or general network problems can also cause TCP retransmissions, which will increase network traffic, further adding to any network performance problems.

Also, compare the number of packets received with the number of completely duplicate packets. If TCP on a sending node times out before an ACK is received from the receiving node, it will retransmit the packet. Duplicate packets occur when the receiving node eventually receives all the retransmitted packets. If the number of duplicate packets exceeds 10-15 percent, the problem may again be too much network traffic or a bottleneck at the receiving node. Duplicate packets increase network traffic.

The value for retransmit timeouts occurs when TCP sends a packet but does not receive an ACK in time. It then re-sends the packet. This value is incremented for any subsequent retransmittals. These continuous retransmittals drive CPU utilization higher, and if the receiving node does not receive the packet, it eventually will be dropped.

The **netstat -D** command shows the number of packets received, transmitted, and dropped in the communications subsystem as shown in Example 6-34.

Example 6-34 netstat -D command

```
[p630n04][/]> netstat -D

Source                          Ipkts               Opkts      Idrops      Odrops
-------------------------------------------------------------------------------
ent_dev0                       502053              166238           0           0
ent_dev2                            0                   0           0           0
ent_dev1                        29505               31669           0           0
```

```
                ----------------------------------------------------------------
Devices Total                531558               197907           0           0
--------------------------------------------------------------------------------
ent_dd0                      502053               166238           0           0
ent_dd2                           0                    0           0           0
ent_dd1                       29505                31669           0           0
                ----------------------------------------------------------------
Drivers Total                531558               197907           0           0
--------------------------------------------------------------------------------
fcs_dmx0                          0                  N/A           0         N/A
fcs_dmx1                          0                  N/A           0         N/A
ent_dmx0                     501997                  N/A          60         N/A
ent_dmx2                          0                  N/A           0         N/A
ent_dmx1                      29505                  N/A           0         N/A
                ----------------------------------------------------------------
Demuxer Total                531502                  N/A          60         N/A
--------------------------------------------------------------------------------
IP                           563528               247255       85498       21217
IPv6                             16                   16           0           0
TCP                          335550               148439          65           0
UDP                           35488                10960        3632           0
                ----------------------------------------------------------------
Protocols Total              934566               406654       89195       21217
--------------------------------------------------------------------------------
en_if0                       501997               166253           0          61
en_if1                        29505                31787           0        6548
lo_if0                        73308                81766        8459           0
                ----------------------------------------------------------------
Net IF Total                 604810               279806        8459        6609
--------------------------------------------------------------------------------
                ----------------------------------------------------------------
NFS/RPC Total                   N/A                65878           0           0
--------------------------------------------------------------------------------
(Note:  N/A -> Not Applicable)
[p630n04][/]>
```

> **Note:** In the statistics output, a N/A displayed in a field indicates the count is not applicable. For the NFS/RPC statistics, the number of incoming packets that pass through RPC is the same as the number of packets that pass through NFS, so these numbers are not summed in the NFS/RPC Total field, thus the N/A displayed. NFS has no outgoing packet or outgoing packet drop counters specific to NFS and RPC. Therefore, individual counts have a field value of N/A, and the cumulative count is stored in the NFS/RPC Total field.

The *Devices* layer shows number of packets coming into the adapter, going out of the adapter, and number of packets dropped on input and output. There are

various causes of adapter errors, and the **netstat -v** command can be examined for more details.

The *Drivers* layer shows packet counts handled by the device driver for each adapter. Output of the **netstat -v** command is useful here to determine which errors are counted.

The *Demuxer* values show packet counts at the demux layer, and *Idrops* here usually indicate that filtering has caused packets to be rejected (for example, NetWare or DecNet packets being rejected because these are not handled by the system under examination). Details for the Protocols layer can be seen in the output of the **netstat -s** or **netstat -p** commands.

The **netstat -c** command provides statistics about the network buffer cache (NBC) usage as in Example 6-35.

Example 6-35 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 command shows the statistics of the Network Buffer Cache. The Network Buffer Cache is a list of network buffers that contain data that can be transmitted to networks. The Network Buffer Cache 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

The Example 6-35 on page 368 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. This may also signal that 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).

Network options to control the NBC

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.

6.4.3 The pmtu command

The **pmtu** command manages pmtu information. It is used to displays and deletes Path MTU discovery related information.

The pmtu command is provided to manage the Path MTU information. The command can be used to display the Path MTU table. By default the IPV4 (IP Version 4) pmtu entries are displayed. IPV6 pmtu entries can be displayed using the –inet6 flag. This command also enables a root user to delete a pmtu entry (using the pmtu delete command). The delete can be based on destination, gateway, or both.

A pmtu entry gets added into the PMTU table when a route add occurs with an MTU value.

Another network option, pmtu_expire, is provided to expire unused pmtu entries. The default value of pmtu_expire is 10 minutes.

Syntax

pmtu [-inet6] display/[delete [-dst destination] [-gw gateway]]

In AIX 5.2 an later, the **netstat -rn** command does not display information about Path MTU in its output. The pmtu command should be used to display or delete any information about Path MTU.

Example 6-36 shows the output of **pmtu display** command.

Example 6-36 The pmtu display command output

```
"[p630n04][/home/hennie/tcpdump]> pmtu display

   dst           gw           If    pmtu     refcnt   redisc_t    exp

 --------------------------------------------------------------------------

9.12.6.143    192.168.100.    en0    1500        2          7       0

192.168.100.  127.0.0.1       lo0   16896        2          7       0

192.168.100.  192.168.100.    en0    1500        1         22       0

127.0.0.1     127.0.0.1       lo0   16896        3          7       0
[p630n04][/home/hennie/tcpdump]>
```

The field in the previous Example 6-36 have the following description:

dst Displays the destination network of the path.

gw	Display the gateway used to connect to the network
If	Displays the interface used for the connection
pmtu	Displays the Path MTU size used for the connection
refcnt	Displays the number of current TCP and UDP applications using this pmtu entry
redisc_t	Displays the amount of time that is elapsed since the last Path MTU discovery attempt. The PMTU is rediscovered after every **pmtu_rediscover_interval** minutes. Its default value is 30 minutes and can be changed using the **no** command.
exp	Displays the pmtu expiry time. The expiry time is controlled by the network option **pmtu_expire**. Its default value is 10 minutes. This value can be changed using the **no** command. A value of 0 does not expire any entries. The exp entry signifies the expiry time. PMTU entries having more than zero *refcnt* have *exp* of 0. When the *refcnt* becomes zero, the *exp* time increases every minute and the entry gets deleted when the exp variable becomes equal to **pmtu_expire**.

To delete a particular Path MTU entry use the **pmtu delete** command as in Example 6-37.

Example 6-37 Delete pmtu entry

```
[p630n04][/]> pmtu delete -dst 9.12.6.143
```

6.5 Network packet tracing tools

This section describes the network packet tracing commands and other packet monitoring tools.

6.5.1 The iptrace command

The **iptrace** command provides interface-level packet tracing for Internet protocols.

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.

Measurement and sampling

The **iptrace** command can monitor more than one network interface at the same time, and not only one as with the **tcpdump** command. 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 to format the binary network trace data into a readable format (unlike **tcpdump** which can do both).

Note: 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 5.3 Technical Reference: Communications, Volume 2. For more information about the net_xmit_trace kernel service, see AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems, Volume 1.

Syntax:

```
/usr/sbin/iptrace  [ -a ] [ -b ][ -e ] [ -u ] [ -PProtocol_list ] [ -i
                   Interface ] [ -p Port_list ] [ -s Host [ -b ] ] [
                   -dHost ] [ -L Log_size ] [ -B ] [ -T ] [ -S
                   snap_length]  LogFile
```

The **iptrace** command is located in /usr/sbin/iptrace, and it is part of the bos.net.tcp.server fileset.

Example of iptrace command

As mentioned in the previous paragraph, the **iptrace** command can be run in two ways, from the command line or using the SRC service. If starting iptrace with the **iptrace** command you have to stop it using the **kill -15** PID command. The kernel extension loaded by the iptrace daemon remains active in memory if **iptrace** is stopped any other way.

Example 6-38 Starting iptrace with startsrc

```
[p630n04][/]> startsrc -s iptrace -a "-i en0 iptrc.out" &
[1]     26402
[p630n04][/]>
```

The command in example Example 6-38 shows how to manually start iptrace and monitor any packets passing through interface en0. Use with care, as when using any type of tracing tool, since large amounts of information gets collected in a very short period of time.

We ran iptrace for a period of 20 seconds over a very busy network and this created a trace file of 184 Mb. To stop tracing, use the **stopsrc** command as in Example 6-39.

Example 6-39 Stop iptrace with the iptrace command

```
[p630n04][/]> stopsrc -s iptrace
```

6.5.2 The ipreport command

After the trace file has been created you must use the **ipreport** command to generate a human readable you can analyze.

Syntax:

```
ipreport [-CenrsSvx1NT] [-c count] [-j pktnum] [-X bytes] tracefile

    -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 packets
    -N: dont do name resolution
    -r: know about rpc
    -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 AIX3.1
```

The ipreport command is located in /**usr/sbin/ipreport** and is part of the **bos.net.tcp.server** fileset.

When using the **ipreport** command you must specify the existing trace file that was generated by the **iptrace** command. The **ipreport** command writes information generated to standard output, so you can use output redirection to a file as in Example 6-40. After the report file has been created, use the viewer of your choice to see the contents of the file.

Example 6-40 Using ipreport to generate a report file

```
[p630n04][/home/hennie/iptrc]> ipreport -r -s iptrc.out > ipreport
```

Example 6-41 shows a sample output form a report generated by the **ipreport** command. Observe the lines related to the **ping** command.

Example 6-41 Caption from output of ipreport

```
........lines omitted...........

ETH: ====( 98 bytes transmitted on interface en0 )====10:46:50.449898352
ETH:    [ 00:02:55:4f:d6:74 -> 00:02:55:4f:c4:ab ]  type 800  (IP)
IP:     < SRC =  192.168.100.34 >  (p630n04)
IP:     < DST =  192.168.100.31 >  (p630n01)
```

```
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=45286, ip_off=0
IP:     ip_ttl=255, ip_sum=c12f, ip_p = 1 (ICMP)
ICMP:   icmp_type=8 (ECHO_REQUEST)  icmp_id=26464  icmp_seq=1

ETH: ====( 100 bytes received on interface en0 )====10:46:50.450025717
ETH:    [ 00:02:55:4f:c4:ab -> 00:02:55:4f:d6:74 ]  type 800  (IP)
IP:     < SRC =  192.168.100.31 >  (p630n01)
IP:     < DST =  192.168.100.34 >  (p630n04)
IP:     ip_v=4, ip_hl=20, ip_tos=0, ip_len=84, ip_id=56741, ip_off=0
IP:     ip_ttl=255, ip_sum=9470, ip_p = 1 (ICMP)
ICMP:   icmp_type=0 (ECHO_REPLY)  icmp_id=26464  icmp_seq=1

.............lines omited.................
```

The fields of interest (for the **ping** command) are:

- The source (SRC) and destination (DST) host address, both in dotted decimal and in ASCII
- The IP packet length (ip_len)
- The indication of the higher-level protocol in use (ip_p)

Example 6-42 shows the captured information about FTP packets. Observe the IP packet size, ip_len information.

Example 6-42 Observing ftp packets

```
.........lines omitted.............

ETH: ====( 4434 bytes transmitted on interface en0 )====11:25:49.84682843
ETH:    [ 00:02:55:4f:d6:74 -> 00:02:55:4f:c4:ab ]  type 800  (IP)
IP:     < SRC =  192.168.100.34 >  (p630n04)
IP:     < DST =  192.168.100.31 >  (p630n01)
IP:     ip_v=4, ip_hl=20, ip_tos=8, ip_len=4420, ip_id=49936, ip_off=0 DF
IP:     ip_ttl=60, ip_sum=2109, ip_p = 6 (TCP)
TCP:    <source port=32836, destination port=20(ftp-data) >
TCP:    th_seq=8f233e5c, th_ack=67842c8d
TCP:    th_off=5, flags<ACK>
TCP:    th_win=17520, th_sum=5b4, th_urp=0
TCP: 00000000     74686973 20697320 61206269 6766696c     |this is a bigfil|
TCP: 00000010     650a7468 69732069 73206120 62696766     |e.this is a bigf|
TCP: 00000020     696c650a 74686973 20697320 61206269     |ile.this is a bi|
TCP: 00000030     6766696c 650a7468 69732069 73206120     |gfile.this is a |
TCP: 00000040     62696766 696c650a 74686973 20697320     |bigfile.this is |

............lines omited.............
```

6.5.3 The ipfilter command

The **ipfilter** command extracts different operation headers from an ipreport output file and displays them in a table. Some customized NFS information regarding requests and replies is also provided.

syntax

```
ipfilter [ -f [ u n t x c a ] ] [ -s [ u n t x c a ] ] [ -n [ -d
   milliseconds ] ] ipreport_output_file
```

The **ipfilter** command is located in /usr/bin/ipfilter and is part of the **bos.perf.tools** fileset.

The **ipfilter** command reads 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* (see Example 6-43).

Example 6-43 Output of the ipfilter command

```
Operation Headers:  ICMP IPX NFS TCP UDP ATM


Ports

-------------------------------------------------------------------------------------------
pkt. Source          Dest. Length Seq #     Ack #    Source Dest      Net_Interf Operation
-------------------------------------------------------------------------------------------
101 9.12.6.143 192.168.100.34 41, 5d6975ec, 2c4d7c53 1172, 23(telnet) en0 TCP ACK PUSH
102 192.168.100.34 9.12.6.143 41, 2c4d7c53, 5d6975ed 23(telnet), 1172 en0 TCP ACK PUSH
103 9.12.6.143 192.168.100.34 41, 5d6975ed, 2c4d7c54 1172, 23(telnet) en0 TCP ACK PUSH
104 192.168.100.34 9.12.6.143 41, 2c4d7c54, 5d6975ee 23(telnet), 1172 en0 TCP ACK PUSH
```

6.5.4 The netpmon command

The **netpmon** command monitors activity and reports statistics on network I/O and network-related CPU usage.

Syntax:

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

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*, *cstokdd*, 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.

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 Usage
: The **netpmon** command monitors CPU usage by all threads and interrupt handlers. It estimates how much of this usage is due to network-related activities.

Network Device-Driver I/O
: The **netpmon** command monitors I/O operations through Micro-Channel Ethernet, token- ring, and Fiber-Distributed Data Interface (FDDI) network device drivers. In the case of transmission I/O, the command also monitors utilizations, queue lengths, and destination hosts. For receive ID, the command also monitors time in the *demux* layer.

Internet Socket Calls
: The **netpmon** command monitors all send, recv, sendto, recvfrom, read, and write subroutines on Internet sockets. It reports statistics on a per-process basis, for each of the following protocol types:
- 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 Network File System (NFS) files, client NFS remote procedure call (RPC) requests, and NFS server read or write requests. The command reports subroutine statistics on a per-process or optional per-thread basis and on a per-file basis for each server. The **netpmon** command reports client RPC statistics for

each server, and server read and write statistics for each client.

Any combination of the preceding report types can be specified with the command line flags. By default, all the reports are produced.

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:

```
"TRACEBUFFER 8 WRAPAROUND, 10249 missed entries"
```

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.

Example 6-44 Starting netpmon tracing

```
[p630n04][/nfs] > netpmon -T 10000000 -o /netpmon/netpmon.out
```

In Example 6-44 we are starting **netpmon** specifying that it should use a trace buffer size of 1,000,000 bytes and the output be written to a file called /netpmon/netpmon.out.

Once **netpmon** has been started start running commands to generate network activity. After all the commands has completed run **trcstop** to stop the tracing (see Example 6-45).

Example 6-45 Stop netpmon tracing

```
[p630n04][/nfs]> trcstop
[netpmon: Reporting started]
[p630n04][/nfs]>
[netpmon: Reporting completed]

[netpmon: 166.387 secs in measured interval]

[p630n04][/nfs]>
```

After you have stopped tracing you can view the contents of the netpmon output file with the **more** command, as in Example 6-46 on page 379.

Example 6-46 netpmon output file

```
[p630n04][/home/hennie/netpmon]> more /netpmon/netpmon.ou
Fri Oct 22 17:05:49 2004
System: AIX p630n04 Node: 5 Machine: 000685CF4C00

TRACEBUFFER 134 WRAPAROUND, 30491 missed entries

========================================================================

Process CPU Usage Statistics:
-----------------------------
                                                    Network
Process (top 20)             PID  CPU Time   CPU %   CPU %
-----------------------------------------------------------
cp                         25724   11.2790   2.943   0.000
cat                        26240    4.3129   1.125   0.000
netpmon                    27224    2.6688   0.696   0.000
rm                         26248    1.1326   0.296   0.000
IBM.CSMAgentRMd            23870    0.9378   0.245   0.000
ftp                        31610    0.4678   0.122   0.080
ftp                        28436    0.4410   0.115   0.075
ftp                        24774    0.4271   0.111   0.074
ftp                        25418    0.4144   0.108   0.073
lrud                        1806    0.4079   0.106   0.000
ftp                        24866    0.3999   0.104   0.072
ftp                        17390    0.3973   0.104   0.072
ftp                        26486    0.3949   0.103   0.072
UNKNOWN                    19022    0.3580   0.093   0.000
hats_nim                   23734    0.3461   0.090   0.000
ksh                        26238    0.1186   0.031   0.000
ksh                        25726    0.1153   0.030   0.000
ksh                        26484    0.1146   0.030   0.000
hats_nim                   26844    0.1095   0.029   0.000
pilegc                      2580    0.0965   0.025   0.000
-----------------------------------------------------------
Total (all processes)               25.4537   6.642   0.518
Idle time                          347.9617  90.802

========================================================================

First Level Interrupt Handler CPU Usage Statistics:
---------------------------------------------------
                                                    Network
FLIH                               CPU Time   CPU %   CPU %
-----------------------------------------------------------
external device                      3.1065   0.811   0.160
PPC decrementer                      0.3027   0.079   0.000
data page fault                      0.0994   0.026   0.000
```

```
queued interrupt                    0.0892   0.023   0.000
-------------------------------------------------------------
Total (all FLIHs)                   3.5979   0.939   0.160

================================================================================

Second Level Interrupt Handler CPU Usage Statistics:
-----------------------------------------------------
                                                    Network
SLIH                              CPU Time   CPU %   CPU %
-------------------------------------------------------------
scentdd32                           6.5237   1.702   0.990
s_scsiddpin32                       0.1059   0.028   0.000
goentdd32                           0.0009   0.000   0.000
efcddpin32                          0.0004   0.000   0.000
/unix                               0.0001   0.000   0.000
-------------------------------------------------------------
Total (all SLIHs)                   6.6311   1.730   0.990

================================================================================

TCP Socket Call Statistics (by Process):
----------------------------------------
                                  ------ Read -----   ----- Write -----
Process (top 20)          PID   Calls/s   Bytes/s   Calls/s   Bytes/s
--------------------------------------------------------------------------------
ftp                     31610      0.08       342     19.73   1289477
ftp                     28436      0.08       342     18.57   1213545
ftp                     24774      0.08       342     18.12   1184130
ftp                     25418      0.08       342     17.82   1164292
ftp                     24866      0.08       342     17.70   1156767
ftp                     17390      0.08       342     17.56   1147190
ftp                     26486      0.08       342     17.44   1139666
sshd                    20794      5.10     83628      3.68       257
--------------------------------------------------------------------------------
Total (all processes)               5.69     86022    130.62   8295325

================================================================================

ICMP Socket Call Statistics (by Process):
-----------------------------------------
                                  ------ Read -----   ----- Write -----
Process (top 20)          PID   Calls/s   Bytes/s   Calls/s   Bytes/s
--------------------------------------------------------------------------------
hats_nim                26844      0.33       342      0.33        27
hats_nim                23734      0.02        21      0.02         2
--------------------------------------------------------------------------------
Total (all processes)               0.35       363      0.35        29
```

```
========================================================================

NFS Client RPC Statistics (by Server):
--------------------------------------

Server                     Calls/s
---------------------------------
p630n01                       5.62
------------------------------------------------------------------------
Total (all servers)           5.62

========================================================================

Detailed Second Level Interrupt Handler CPU Usage Statistics:
-------------------------------------------------------------

SLIH: scentdd32
count:                  267892
  cpu time (msec):      avg 0.024   min 0.010   max 12.402  sdev 0.098

SLIH: s_scsiddpin32
count:                  9444
  cpu time (msec):      avg 0.011   min 0.006   max 0.026   sdev 0.002

SLIH: goentdd32
count:                  50
  cpu time (msec):      avg 0.018   min 0.008   max 0.028   sdev 0.005

SLIH: efcddpin32
count:                  29
  cpu time (msec):      avg 0.015   min 0.008   max 0.027   sdev 0.006

SLIH: /unix
count:                  39
  cpu time (msec):      avg 0.004   min 0.002   max 0.006   sdev 0.001

COMBINED (All SLIHs)
count:                  277454
  cpu time (msec):      avg 0.024   min 0.002   max 12.402  sdev 0.097

========================================================================

Detailed TCP Socket Call Statistics (by Process):
-------------------------------------------------

PROCESS: ftp   PID: 31610
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 3.061   min 0.006   max 9.761   sdev 3.551
```

```
writes:                 1890
  write sizes (bytes):  avg 65362.7 min 8       max 65536   sdev 3365.5
  write times (msec):   avg 36.975  min 0.014   max 403.457 sdev 18.378

PROCESS: ftp   PID: 28436
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 4.996   min 0.007   max 19.014  sdev 6.031
writes:                 1779
  write sizes (bytes):  avg 65351.9 min 8       max 65536   sdev 3468.7
  write times (msec):   avg 38.907  min 0.014   max 403.273 sdev 17.478

PROCESS: ftp   PID: 24774
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 6.182   min 0.006   max 21.363  sdev 6.547
writes:                 1736
  write sizes (bytes):  avg 65347.3 min 8       max 65536   sdev 3511.2
  write times (msec):   avg 39.630  min 0.013   max 403.241 sdev 16.834

PROCESS: ftp   PID: 25418
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 8.733   min 0.006   max 23.426  sdev 7.990
writes:                 1707
  write sizes (bytes):  avg 65344.1 min 8       max 65536   sdev 3540.8
  write times (msec):   avg 40.072  min 0.014   max 403.229 sdev 17.128

PROCESS: ftp   PID: 24866
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 10.733  min 0.006   max 46.869  sdev 14.107
writes:                 1696
  write sizes (bytes):  avg 65342.8 min 8       max 65536   sdev 3552.3
  write times (msec):   avg 40.102  min 0.013   max 403.016 sdev 16.784

PROCESS: ftp   PID: 17390
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 9.895   min 0.006   max 29.064  sdev 8.615
writes:                 1682
  write sizes (bytes):  avg 65341.2 min 8       max 65536   sdev 3567.0
  write times (msec):   avg 40.227  min 0.013   max 403.264 sdev 16.952

PROCESS: ftp   PID: 26486
reads:                  8
  read sizes (bytes):   avg 4096.0  min 4096    max 4096    sdev 0.0
  read times (msec):    avg 11.040  min 0.005   max 31.570  sdev 9.328
writes:                 1671
```

```
  write sizes (bytes):  avg 65339.9 min 8       max 65536   sdev 3578.7
  write times (msec):   avg 40.270  min 0.013   max 403.002 sdev 17.022

PROCESS: sshd   PID: 20794
reads:                  489
  read sizes (bytes):   avg 16384.0 min 16384   max 16384   sdev 0.0
  read times (msec):    avg 0.007   min 0.005   max 0.054   sdev 0.004
writes:                 353
  write sizes (bytes):  avg 69.7    min 52      max 388     sdev 28.9
  write times (msec):   avg 0.017   min 0.013   max 0.063   sdev 0.005

PROTOCOL: TCP (All Processes)
reads:                  545
  read sizes (bytes):   avg 15121.4 min 4096    max 16384   sdev 3731.1
  read times (msec):    avg 0.809   min 0.005   max 46.869  sdev 3.744
writes:                 12514
  write sizes (bytes):  avg 63506.0 min 8       max 65536   sdev 11348.2
  write times (msec):   avg 38.299  min 0.013   max 403.457 sdev 18.250

======================================================================

Detailed ICMP Socket Call Statistics (by Process):
--------------------------------------------------

PROCESS: hats_nim   PID: 26844
reads:                  32
  read sizes (bytes):   avg 1024.0  min 1024    max 1024    sdev 0.0
  read times (msec):    avg 0.009   min 0.006   max 0.015   sdev 0.002
writes:                 32
  write sizes (bytes):  avg 81.0    min 81      max 81      sdev 0.0
  write times (msec):   avg 0.054   min 0.037   max 0.067   sdev 0.008

PROCESS: hats_nim   PID: 23734
reads:                  2
  read sizes (bytes):   avg 1024.0  min 1024    max 1024    sdev 0.0
  read times (msec):    avg 0.006   min 0.006   max 0.006   sdev 0.000
writes:                 2
  write sizes (bytes):  avg 81.0    min 81      max 81      sdev 0.0
  write times (msec):   avg 0.050   min 0.039   max 0.060   sdev 0.010

PROTOCOL: ICMP (All Processes)
reads:                  34
  read sizes (bytes):   avg 1024.0  min 1024    max 1024    sdev 0.0
  read times (msec):    avg 0.009   min 0.006   max 0.015   sdev 0.002
writes:                 34
  write sizes (bytes):  avg 81.0    min 81      max 81      sdev 0.0
  write times (msec):   avg 0.054   min 0.037   max 0.067   sdev 0.008

======================================================================
```

```
Detailed NFS Client RPC Statistics (by Server):
-----------------------------------------------

SERVER: p630n01
calls:                  538
  call times (msec):    avg 10.680  min 6.720   max 32.442  sdev 1.177

COMBINED (All Servers)
calls:                  538
  call times (msec):    avg 10.680  min 6.720   max 32.442  sdev 1.177
[p630n04][/home/hennie/netpmon]>
```

Example 6-46 on page 379 is a full listing of all the data collected by netpmon.

The data collected by the **netpmon** command in this example is:

- Process CPU Usage Statistics (top 20 processes)
- First Level Interrupt Handler CPU Usage Statistics
- Second Level Interrupt Handler CPU Usage Statistics
- TCP Socket Call Statistics (by Process)
- ICMP Socket Call Statistics (by Process)
- NFS Client RPC Statistics (by Server)
- Detailed Second Level Interrupt Handler CPU Usage Statistics
- Detailed TCP Socket Call Statistics (by Process)
- Detailed ICMP Socket Call Statistics (by Process)
- Detailed NFS Client RPC Statistics (by Server)

The global reports are shown at the beginning of the **netpmon** output, and are the occurrences during the measured interval. The detailed reports provide additional information for the global reports. By default, the reports are limited to the 20 most active statistics measured. All information in the reports is listed from top to bottom as most active to least active.

The reports generated by the **netpmon** command begin with a header, which identifies the date, the machine ID, and the length of the monitoring period in seconds. The header is followed by a set of global and detailed reports for all specified report types.

6.5.5 The trpt command

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

trpt [-a] [-f] [-j] [-pAddress]... [-s] [-t]

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 SO_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 SO_DEBUG flag will not be turned off for sockets that have this set even when the sodebug option is set to zero.

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, see *AIX 5L Version 5.3 Commands Reference, Volume 5*, SC23-4892.

To list the PCB addresses for which trace records exist, use the **-j** parameter with the **trpt** command as in Example 6-47.

Example 6-47 Using trpt -j

```
# trpt -j
7064fbe8
```

You can check the PCB record with the **netstat** command as in Example 6-48.

Example 6-48 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 fields description:

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

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 6-49 shows the output during initialization.

Example 6-49 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 6-50 shows the result of the **trpt** command after the **telnet** session is closed.

Example 6-50 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
```

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 6-51 on page 387. The following example contains the same information as the two examples in Example 6-49 and Example 6-50, 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 6-51 Using trpt -a

```
# trpt -a

7064fbe8:
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=1.3.1.164,23, 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
```

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 6-52. The following example contains the same information as Example 6-49 on page 386 and Example 6-50 on page 386, but with additional details.

Example 6-52 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 [fcbaf1a5..fcbaf1a9)@0(win=4000)<SYN> -> SYN_SENT
   rcv_nxt 0 rcv_wnd 0 snd_una fcbaf1a5 snd_nxt fcbaf1a6 snd_max fcbaf1a6
   snd_wl1 0 snd_wl2 0 snd_wnd 0
365 CLOSED:user CONNECT -> SYN_SENT
   rcv_nxt 0 rcv_wnd 0 snd_una fcbaf1a5 snd_nxt fcbaf1a6 snd_max fcbaf1a6
   snd_wl1 0 snd_wl2 0 snd_wnd 0
365 SYN_SENT:input 4b96e888@fcbaf1a6(win=4410)<SYN,ACK> -> ESTABLISHED
   rcv_nxt 4b96e889 rcv_wnd 4410 snd_una fcbaf1a6 snd_nxt fcbaf1a6 snd_max fcbaf1a6
   snd_wl1 4b96e889 snd_wl2 fcbaf1a6 snd_wnd 4410
...(lines omitted)...
591 LAST_ACK:output fcbaf1d3@4b96e914(win=4410)<ACK,FIN> -> LAST_ACK
   rcv_nxt 4b96e914 rcv_wnd 4410 snd_una fcbaf1d3 snd_nxt fcbaf1d4 snd_max fcbaf1d4
   snd_wl1 4b96e913 snd_wl2 fcbaf1d3 snd_wnd 4410
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
   rcv_nxt 4b96e914 rcv_wnd 4410 snd_una fcbaf1d3 snd_nxt fcbaf1d4 snd_max fcbaf1d4
```

```
   snd_wl1 4b96e913 snd_wl2 fcbaf1d3 snd_wnd 4410
```

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 6-53. The following example contains the same information as Example 6-49 on page 386 and Example 6-50 on page 386, but with additional details.

Example 6-53 Using trpt -t

```
# trpt -t

7064fbe8:
365 CLOSED:user ATTACH -> CLOSED
365 SYN_SENT:output [fcbaf1a5..fcbaf1a9)@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@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
   REXMT=3 (t_rxtshft=0)
365 ESTABLISHED:user SEND -> ESTABLISHED
   REXMT=3 (t_rxtshft=0)
...(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
   REXMT=3 (t_rxtshft=0), 2MSL=1200
591 CLOSE_WAIT:user SHUTDOWN -> LAST_ACK
   REXMT=3 (t_rxtshft=0), 2MSL=1200
```

Printing trace records for a single protocol control block

Example 6-54 shows the trace record for a single protocol control block.

Example 6-54 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).....
```

6.6 NFS related performance commands

The NFS subsystem involves multiple performance monitoring and tuning commands. NFS performance is determined not only by the network subsystem, but also by the Virtual Memory Manager, CPU, and Disk I/O subsystems. In this section we present the NFS related performance monitoring and tuning commands.

6.6.1 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/nfsstat and is part of the bos.net.nfs.client fileset, which is installable from the AIX base installation media.

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

/usr/sbin/nfsstat [-c] [-s] [-n] [-r] [-m] [-v4] [-z] [-x] [-t] [-b][-g]

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, **nfsstat -z** command can only be executed by root.

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.

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

NFS server RPC statistics

The output in Example 6-55 shows the server RPC statistics created using the **nfsstat -sr** command:

Example 6-55 Output of nfsstat -sr

```
[p630n04][/nfs2]> nfsstat -sr

Server rpc:
Connection oriented
calls     badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
234949    0          0          0          0          119817     0
Connectionless
calls     badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
0         0          0          0          0          0          0
[p630n04][/nfs2]>
```

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.

`xdrcall` 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. 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 “NFS client RPC statistics” on page 393 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.

See “The nfso command” on page 416 for an explanation on how to change the nfs option listed above.

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 server 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.

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 6-56 shows the output of the **nfsstat -sn** command.

Example 6-56 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.

NFS client RPC statistics

The output in Example 6-57 shows the client RPC statistics created using the command **`nfsstat -cr`**.

Example 6-57 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. See also the related commands in 5.1.5, “The vmstat command” on page 310, 7.2.1, “The iostat

command" on page 433, and 6.4.2, "The netstat command" on page 356. Use the **netstat -in**, **netstat -s**, **netstat -v**, and **netstat -m** commands on the server and client to check for dropped packets.

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 **nfso** command in case of socket buffer overflows is required. Refer to 6.7.3, "The nfso command" on page 416 for more information about the **nfso** command.

`badxid` 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 `retrans` field.

`timeouts` Number of times a call timed-out while waiting for a reply from the server. The same as for the `retrans` value applies. Refer to the description in for the `retrans` field.

Increasing the NFS mount option timeo by using the **smitty chnfsmnt** 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 timeo value is only used for the first retransmission timeout. Refer to "Statistics on mounted file systems" on page 395 for more details.

`newcreds` Number of times authentication information had to be refreshed.

`badverfs` Number of times a call failed due to a bad verifier in the response.

`timers` Number of times the calculated time-out value was greater than or equal to the minimum specified time-out value for a call.

`nomem` Number of times a call failed due to a failure to allocate memory.

`cantconn` Number of times a call failed due to a failure to make a connection to the server.

`interrupts` Number of times a call was interrupted by a signal before completing.

`cantsend` Number of times a send failed due to a failure to make a connection to the client.

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 6-58 was taken on the NFS client at the same time the NFS Server Example 6-56 on page 392 was produced.

Example 6-58 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 “NFS server NFS statistics” on page 392 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 6-56 on page 392. 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.

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 6-59 on page 396 is an example of the **nfsstat -m** output.

Example 6-59 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,wsize=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 *AIX 5L Version 5.3 Commands Reference, Volume 5*, SC23-4892, 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 respond time values are shown for this function.

The dynamic retransmission can be turned off using the NFS option nfs_dynamic_retrans. Refer to 6.7.3, “The nfso command” on page 416 for more information. The default in AIX is that dynamic retransmission is used.

6.7 Network tuning commands

Beside network monitoring, tuning is a very important component to consider for obtaining optimal system performance. This section presents the network-related tuning commands, mentioning also other tuning commands, not directly involved in network parameters tuning.

6.7.1 The no command

The **no** (network options) command is used to set network tuning parameters.

Use the no command to configure network tuning parameters. The no command sets or displays current or next boot values for network 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. When the no command is used to modify a network option it logs a message to the syslog using the LOG_KERN facility.

no Syntax

```
no [ -p | -r ] { -o Tunable[=NewValue] }
no [ -p | -r ] {-d Tunable }
no [ -p | -r ] { -D }
no [ -p | -r ] -a
no -?
no -h [ Tunable ]
no -L [ Tunable ]
no -x [ Tunable ] Note: Multiple flags -o, -d, -x, and -L are allowed.
```

The **no** command in located in **/usr/sbin/no** and is part of the bos.net.tcp.client fileset. This fileset is installed by default at installation time.

Be careful when you use this command. If used incorrectly, the **no** command can cause your system to become inoperable.

Before modifying any tunable parameter, you should first carefully read about all its characteristics of a tunable. For more information about tunable parameters, see Network Tunable Parameters in the **man** pages

You must then make sure that the Diagnosis and Tuning sections for this parameter truly apply to your situation and that changing the value of this parameter could help improve the performance of your system.

The no examples

A list of all the available tunables can be displayed with the **no -a** command as in Example 6-60.

Example 6-60 The no -a example

```
[p630n04][/home/hennie]> 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 = 1
                   fasttimo = 200
           icmp6_errmsg_rate = 10
             icmpaddressmask = 0
ie5_old_multicast_mapping = 0
                     ifsize = 256
          inet_stack_size = 16
               ip6_defttl = 64
                ip6_prune = 1
            ip6forwarding = 0
         ip6srcrouteforward = 1
              ipforwarding = 1
                 ipfragttl = 60
           ipignoreredirects = 0
                 ipqmaxlen = 100
            ipsendredirects = 1
           ipsrcrouteforward = 1
              ipsrcrouterecv = 1
              ipsrcroutesend = 1
            llsleep_timeout = 3
                  lowthresh = 90
                   main_if6 = 0
                 main_site6 = 0
                  maxnip6q = 20
                    maxttl = 255
                 medthresh = 95
                mpr_policy = 1
               multi_homed = 1
                 nbc_limit = 891289
             nbc_max_cache = 131072
             nbc_min_cache = 1
          nbc_ofile_hashsz = 12841
                  nbc_pseg = 0
             nbc_pseg_limit = 4194304
             ndd_event_name = {all}
           ndd_event_tracing = 0
              ndp_mmaxtries = 3
              ndp_umaxtries = 3
                   ndpqsize = 50
                  ndpt_down = 3
                  ndpt_keep = 120
                 ndpt_probe = 5
            ndpt_reachable = 30
```

```
            ndpt_retrans = 1
            net_buf_size = {all}
            net_buf_type = {all}
       net_malloc_police = 0
          nonlocsrcroute = 1
                nstrpush = 8
             passive_dgd = 0
        pmtu_default_age = 10
             pmtu_expire = 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 = 75
                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_mssdflt = 1460
```

```
          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 = 1048576
        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
             use_sndbufpool = 1
[p630n04][/home/hennie]>
```

The no -o command is used to display or set a specific tunable.In Example 6-61 the **no -o** command is used to display the tunable value of **tcp_recvspace.**

Example 6-61 The no -o example to display a tunable

```
[p630n04][/home/hennie]> no -o tcp_recvspace
tcp_recvspace = 16384
[p630n04][/home/hennie]>
```

When changing the value of a tunable make sure you understand the characteristics of the tunable.

The **no -L** command can be used to display the values associated with the tunables. All the tunables can be listed with its attributes or a particular tunable can be displayed.

To display all the attributes associated with the no command use **no -L** with no arguments as in Example 6-62.

Example 6-62 The no -L command

```
[p630n04][/home/hennie]> no -L

General Network Parameters
-------------------------------------------------------------------------------------------
```

```
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT          TYPE
DEPENDENCIES
------------------------------------------------------------------------------------------
extendednetstats          1      0      0      0      1      boolean       R
------------------------------------------------------------------------------------------
fasttimo                  200    200    200    50     200    millisecond   D
------------------------------------------------------------------------------------------
inet_stack_size           16     16     16     1      32K-1  kbyte         R
------------------------------------------------------------------------------------------
nbc_limit                 891289 891289 891289 0      2G-1   kbyte         D
     thewall
------------------------------------------------------------------------------------------
nbc_max_cache             128K   128K   128K   1      2G-1   byte          D
     nbc_min_cache
------------------------------------------------------------------------------------------
nbc_min_cache             1      1      1      1      128K   byte          D
     nbc_max_cache
------------------------------------------------------------------------------------------
nbc_ofile_hashsz          12841  12841  12841  1      999999 segment       D
------------------------------------------------------------------------------------------
nbc_pseg                  0      0      0      0      2G-1   segment       D
------------------------------------------------------------------------------------------
nbc_pseg_limit            4M     4M     4M     0      2G-1   kbyte         D
------------------------------------------------------------------------------------------
ndd_event_name            {all}  {all}  {all}  0      128    string        D
------------------------------------------------------------------------------------------
ndd_event_tracing         0      0      0      0      2G-1   numeric       D
------------------------------------------------------------------------------------------
net_buf_size              {all}  {all}  {all}  0      128    string        D
------------------------------------------------------------------------------------------
net_buf_type              {all}  {all}  {all}  0      128    string        D
------------------------------------------------------------------------------------------
net_malloc_police         0      0      0      0      2G-1   numeric       D
------------------------------------------------------------------------------------------
sb_max                    1M     1M     1M     1      2G-1   byte          D
------------------------------------------------------------------------------------------
send_file_duration        300    300    300    0      2G-1   second        D
------------------------------------------------------------------------------------------
sockthresh                75     85     85     0      100    %_of_thewall  D
------------------------------------------------------------------------------------------
sodebug                   0      0      0      0      1      boolean       C
------------------------------------------------------------------------------------------
somaxconn                 1K     1K     1K     0      32K-1  numeric       C
------------------------------------------------------------------------------------------
tcp_inpcb_hashtab_siz     24499  24499  24499  1      999999 numeric       R
------------------------------------------------------------------------------------------
thewall                   1M     1M     1M     0      1M     kbyte         S
------------------------------------------------------------------------------------------
udp_inpcb_hashtab_siz     24499  24499  24499  1      83000  numeric       R
```

```
-------------------------------------------------------------------------------------------------
use_isno                  1      1      1      0      1      boolean           D
-------------------------------------------------------------------------------------------------
use_sndbufpool            1      1      1      0      1      boolean           R
-------------------------------------------------------------------------------------------------
clean_partial_conns       0      0      0      0      1      boolean           D
-------------------------------------------------------------------------------------------------
delayack                  0      0      0      0      3      boolean           D
-------------------------------------------------------------------------------------------------
delayackports             {}     {}     {}     0      10     ports_list        D
-------------------------------------------------------------------------------------------------
rfc1323                   0      0      0      0      1      boolean           C
-------------------------------------------------------------------------------------------------
rfc2414                   0      0      0      0      1      boolean           C
-------------------------------------------------------------------------------------------------
rto_high                  64     64     64     2      2G-1   roundtriptime R
     rto_low
-------------------------------------------------------------------------------------------------
rto_length                13     13     13     1      64     roundtriptime R
-------------------------------------------------------------------------------------------------
rto_limit                 7      7      7      1      64     roundtriptime R
     rto_high
     rto_low
-------------------------------------------------------------------------------------------------
rto_low                   1      1      1      1      63     roundtriptime     R
     rto_high
-------------------------------------------------------------------------------------------------
sack                      0      0      0      0      1      boolean           C
-------------------------------------------------------------------------------------------------
tcp_bad_port_limit        0      0      0      0      2G-1   numeric           D
-------------------------------------------------------------------------------------------------
tcp_ecn                   0      0      0      0      1      boolean           C
-------------------------------------------------------------------------------------------------
tcp_ephemeral_high        64K-1  64K-1  64K-1  32K+1  64K-1  numeric           D
     tcp_ephemeral_low
-------------------------------------------------------------------------------------------------
tcp_ephemeral_low         32K    32K    32K    1K     65534  numeric           D
     tcp_ephemeral_high
-------------------------------------------------------------------------------------------------
tcp_finwait2              1200   1200   1200   0      64K-1  halfsecond        D
-------------------------------------------------------------------------------------------------
tcp_init_window           0      0      0      0      32K-1  byte              C
-------------------------------------------------------------------------------------------------
tcp_keepcnt               8      8      8      0      2G-1   numeric           D
-------------------------------------------------------------------------------------------------
tcp_keepidle              14400  14400  14400  1      2G-1   halfsecond        C
-------------------------------------------------------------------------------------------------
tcp_keepinit              150    150    150    1      2G-1   halfsecond        D
-------------------------------------------------------------------------------------------------
```

```
tcp_keepintvl             150    150    150    1      32K-1  halfsecond        C
--------------------------------------------------------------------------------
tcp_limited_transmit      1      1      1      0      1      boolean           D
--------------------------------------------------------------------------------
tcp_maxburst              0      0      0      0      32K-1  numeric           D
--------------------------------------------------------------------------------
tcp_mssdflt               1460   1460   1460   1      64K-1  byte              C
--------------------------------------------------------------------------------
tcp_nagle_limit           64K-1  64K-1  64K-1  0      64K-1  byte              D
--------------------------------------------------------------------------------
tcp_ndebug                100    100    100    0      32K-1  numeric           D
--------------------------------------------------------------------------------
tcp_newreno               1      1      1      0      1      boolean           D
--------------------------------------------------------------------------------
tcp_nodelayack            0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
tcp_recvspace             16K    16K    16K    4K     2G-1   byte              C
     sb_max
--------------------------------------------------------------------------------
tcp_sendspace             16K    16K    16K    4K     2G-1   byte              C
     sb_max
--------------------------------------------------------------------------------
tcp_timewait              1      1      1      1      5      15_second         D
--------------------------------------------------------------------------------
tcp_ttl                   60     60     60     1      255    0.6_second        C
--------------------------------------------------------------------------------
udp_bad_port_limit        0      0      0      0      2G-1   numeric           D
--------------------------------------------------------------------------------
udp_ephemeral_high        64K-1  64K-1  64K-1  32K+1  64K-1  numeric           D
     udp_ephemeral_low
--------------------------------------------------------------------------------
udp_ephemeral_low         32K    32K    32K    1K     65534  numeric           D
     udp_ephemeral_high
--------------------------------------------------------------------------------
udp_recvspace             42080  42080  42080  4K     2G-1   byte              C
     sb_max
--------------------------------------------------------------------------------
udp_sendspace             9K     9K     9K     4K     2G-1   byte              C
     sb_max
--------------------------------------------------------------------------------
udp_ttl                   30     30     30     1      255    second            C
--------------------------------------------------------------------------------
udpcksum                  1      1      1      0      1      boolean           D
--------------------------------------------------------------------------------
directed_broadcast        0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
ie5_old_multicast_mapping 0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
ip6_defttl                64     64     64     1      255    numeric           D
```

```
------------------------------------------------------------------------------------------------
ip6_prune                 1      1      1      1      2G-1   second          D
------------------------------------------------------------------------------------------------
ip6forwarding             0      0      0      0      1      boolean         D
------------------------------------------------------------------------------------------------
ip6srcrouteforward        1      1      1      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipforwarding              1      0      0      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipfragttl                 60     60     60     1      255    halfsecond      D
------------------------------------------------------------------------------------------------
ipignoreredirects         0      0      0      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipqmaxlen                 100    100    100    100    2G-1   numeric         R
------------------------------------------------------------------------------------------------
ipsendredirects           1      1      1      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipsrcrouteforward         1      1      1      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipsrcrouterecv            1      0      0      0      1      boolean         D
------------------------------------------------------------------------------------------------
ipsrcroutesend            1      1      1      0      1      boolean         D
------------------------------------------------------------------------------------------------
maxnip6q                  20     20     20     1      32K-1  numeric         D
------------------------------------------------------------------------------------------------
multi_homed               1      1      1      0      3      boolean         D
------------------------------------------------------------------------------------------------
nonlocsrcroute            1      0      0      0      1      boolean         D
------------------------------------------------------------------------------------------------
subnetsarelocal           1      1      1      0      1      boolean         D
------------------------------------------------------------------------------------------------
arpqsize                  12     12     12     1      32K-1  numeric         D
     tcp_pmtu_discover
     udp_pmtu_discover
------------------------------------------------------------------------------------------------
arpt_killc                20     20     20     0      32K-1  minute          D
------------------------------------------------------------------------------------------------
arptab_bsiz               7      7      7      1      32K-1  bucket_size     R
------------------------------------------------------------------------------------------------
arptab_nb                 73     73     73     1      32K-1  buckets         R
------------------------------------------------------------------------------------------------
dgd_packets_lost          3      3      3      1      32K-1  numeric         D
------------------------------------------------------------------------------------------------
dgd_ping_time             5      5      5      1      2G-1   second          D
------------------------------------------------------------------------------------------------
dgd_retry_time            5      5      5      1      32K-1  numeric         D
------------------------------------------------------------------------------------------------
ndp_mmaxtries             3      3      3      0      2G-1   numeric         D
------------------------------------------------------------------------------------------------
```

```
ndp_umaxtries              3      3      3      0      2G-1   numeric           D
--------------------------------------------------------------------------------
ndpqsize                   50     50     50     1      32K-1  numeric           D
--------------------------------------------------------------------------------
ndpt_down                  3      3      3      1      2G-1   halfsecond        D
--------------------------------------------------------------------------------
ndpt_keep                  120    120    120    1      2G-1   halfsecond        D
--------------------------------------------------------------------------------
ndpt_probe                 5      5      5      1      2G-1   halfsecond        D
--------------------------------------------------------------------------------
ndpt_reachable             30     30     30     1      2G-1   halfsecond        D
--------------------------------------------------------------------------------
ndpt_retrans               1      1      1      1      2G-1   halfsecond        D
--------------------------------------------------------------------------------
passive_dgd                0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
rfc1122addrchk             0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
lowthresh                  90     90     90     0      100    %_of_thewall      D
--------------------------------------------------------------------------------
medthresh                  95     95     95     0      100    %_of_thewall      D
--------------------------------------------------------------------------------
nstrpush                   8      8      8      8      32K-1  numeric           R
--------------------------------------------------------------------------------
psebufcalls                20     20     20     20     2G-1   numeric           I
--------------------------------------------------------------------------------
psecache                   1      1      1      0      1      boolean           D
--------------------------------------------------------------------------------
pseintrstack               12K    12K    12K    12K    2G-1   byte              R
--------------------------------------------------------------------------------
psetimers                  20     20     20     20     2G-1   numeric           I
--------------------------------------------------------------------------------
strctlsz                   1K     1K     1K     0      32K-1  byte              D
--------------------------------------------------------------------------------
strmsgsz                   0      0      0      0      32K-1  byte              D
--------------------------------------------------------------------------------
strthresh                  85     85     85     0      100    %_of_thewall      D
--------------------------------------------------------------------------------
strturncnt                 15     15     15     1      2G-1   numeric           D
--------------------------------------------------------------------------------
bcastping                  0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
icmp6_errmsg_rate          10     10     10     1      255    msg/second        D
--------------------------------------------------------------------------------
icmpaddressmask            0      0      0      0      1      boolean           D
--------------------------------------------------------------------------------
ifsize                     256    256    256    8      1K     numeric           R
--------------------------------------------------------------------------------
llsleep_timeout            3      3      3      1      2G-1   second            D
```

```
------------------------------------------------------------------------------------------------
main_if6                  0      0      0      0      1      boolean           D
------------------------------------------------------------------------------------------------
main_site6                0      0      0      0      1      boolean           D
------------------------------------------------------------------------------------------------
maxttl                    255    255    255    1      255    second            D
------------------------------------------------------------------------------------------------
mpr_policy                1      1      1      1      5      numeric           D
------------------------------------------------------------------------------------------------
pmtu_default_age          10     10     10     0      32K-1  minute            D
------------------------------------------------------------------------------------------------
pmtu_expire               10     10     10     0      32K-1  minute            D
------------------------------------------------------------------------------------------------
pmtu_rediscover_interval  30     30     30     0      32K-1  minute            D
------------------------------------------------------------------------------------------------
route_expire              1      1      1      0      1      boolean           D
------------------------------------------------------------------------------------------------
routerevalidate           0      0      0      0      1      boolean           D
------------------------------------------------------------------------------------------------
site6_index               0      0      0      0      32K-1  numeric           D
------------------------------------------------------------------------------------------------
tcp_pmtu_discover         1      1      1      0      1      boolean           D
------------------------------------------------------------------------------------------------
udp_pmtu_discover         1      1      1      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
[p630n04][/home/hennie]>
```

As can be seen in Example 6-62 on page 400 the **no -L** command displays a list of all the tunables and detail about the value of each tunable.

The fields displayed by the no -L command are:

NAME This displays the name of the tunable

CUR	This displays the current value of the tunable
DEF	This displays the default value of the tunable
BOOT	This displays the value of the tunable after a reboot.
MIN	This displays the minimum value of the tunable
MAX	This displays the maximum value of the tunable.
UNIT	This displays the tunables unit of measurement
TYPE	This displays the parameter type. The parameter type specifies how a particular tunable change will take effect. D - Dynamic, the tunable value is a dynamic value and a change to the tunable will take effect immediately. S - Static, the tunable is a static value and the value of the tunable cannot be changed. R - Reboot, the tunable value is a reboot value and the tunable change will only take effect after a reboot. B - Bosboot, the tunable value is a bosboot value and the user needs to run the **bosboot** command for the BLV (Boot logical volume) to be updated. Changes will only take effect after a reboot. M - Mount, the value of the tunable is a mount value and the tunable will only take effect after the file system is remounted or new mounts occur on a file system. I - Incremental, the value of the tunable is incremental and can only be incremented, except at boot time. C - Connect, the value of the tunable is connection orientated, the tunable will only take effect for new socket connections.
DEPENDENCIES	This displays a list of dependable tunables, it will display one dependency per line.

To display the attributes associated with particular tunable see Example 6-63 on page 408. This example displays the output of the **no -L** command to display the value attributes associated with the **tcp_recvspace** tunable.

Example 6-63 The no -L tcp_recvspace

```
[p630n04][/home/hennie]> no -L tcp_recvspace
--------------------------------------------------------------------------------------------------
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------------------------
tcp_recvspace             32K    16K    16K    4K     2G-1   byte             C
     sb_max
--------------------------------------------------------------------------------------------------
[p630n04][/home/hennie]>
```

The **no -x** command gives the same information as the **no -L** command, it just displays the information of each tunable in a comma separated list. See Example 6-64.

Example 6-64 The no -x command

```
[p630n04][/]> no -x
arpqsize,12,12,12,1,32767,numeric,D,tcp_pmtu_discover,udp_pmtu_discover,
arpt_killc,20,20,20,0,32767,minute,D,
arptab_bsiz,7,7,7,1,32767,bucket_size,R,
arptab_nb,73,73,73,1,32767,buckets,R,
bcastping,0,0,0,0,1,boolean,D,
clean_partial_conns,0,0,0,0,1,boolean,D,
delayack,0,0,0,0,3,boolean,D,
delayackports,{},{},{},0,10,ports_list,D,
dgd_packets_lost,3,3,3,1,32767,numeric,D,
dgd_ping_time,5,5,5,1,2147483647,second,D,
dgd_retry_time,5,5,5,1,32767,numeric,D,
directed_broadcast,0,0,0,0,1,boolean,D,
extendednetstats,1,0,0,0,1,boolean,R,
fasttimo,200,200,200,50,200,millisecond,D,
icmp6_errmsg_rate,10,10,10,1,255,msg/second,D,
icmpaddressmask,0,0,0,0,1,boolean,D,
ie5_old_multicast_mapping,0,0,0,0,1,boolean,D,
ifsize,256,256,256,8,1024,numeric,R,
inet_stack_size,16,16,16,1,32767,kbyte,R,
ip6_defttl,64,64,64,1,255,numeric,D,
ip6_prune,1,1,1,1,2147483647,second,D,
ip6forwarding,0,0,0,0,1,boolean,D,
ip6srcrouteforward,1,1,1,0,1,boolean,D,
ipforwarding,1,0,0,0,1,boolean,D,
ipfragttl,60,60,60,1,255,halfsecond,D,
ipignoreredirects,0,0,0,0,1,boolean,D,
ipqmaxlen,100,100,100,100,2147483647,numeric,R,
ipsendredirects,1,1,1,0,1,boolean,D,
ipsrcrouteforward,1,1,1,0,1,boolean,D,
ipsrcrouterecv,1,0,0,0,1,boolean,D,
```

```
ipsrcroutesend,1,1,1,0,1,boolean,D,
llsleep_timeout,3,3,3,1,2147483647,second,D,
lowthresh,90,90,90,0,100,%_of_thewall,D,
main_if6,0,0,0,0,1,boolean,D,
main_site6,0,0,0,0,1,boolean,D,
maxnip6q,20,20,20,1,32767,numeric,D,
maxttl,255,255,255,1,255,second,D,
medthresh,95,95,95,0,100,%_of_thewall,D,
mpr_policy,1,1,1,1,5,numeric,D,
multi_homed,1,1,1,0,3,boolean,D,
nbc_limit,891289,891289,891289,0,2147483647,kbyte,D,thewall,
nbc_max_cache,131072,131072,131072,1,2147483647,byte,D,nbc_min_cache,
nbc_min_cache,1,1,1,1,131072,byte,D,nbc_max_cache,
nbc_ofile_hashsz,12841,12841,12841,1,999999,segment,D,
nbc_pseg,0,0,0,0,2147483647,segment,D,
nbc_pseg_limit,4194304,4194304,4194304,0,2147483647,kbyte,D,
ndd_event_name,{all},{all},{all},0,128,string,D,
ndd_event_tracing,0,0,0,0,2147483647,numeric,D,
ndp_mmaxtries,3,3,3,0,2147483647,numeric,D,
ndp_umaxtries,3,3,3,0,2147483647,numeric,D,
ndpqsize,50,50,50,1,32767,numeric,D,
ndpt_down,3,3,3,1,2147483647,halfsecond,D,
ndpt_keep,120,120,120,1,2147483647,halfsecond,D,
ndpt_probe,5,5,5,1,2147483647,halfsecond,D,
ndpt_reachable,30,30,30,1,2147483647,halfsecond,D,
ndpt_retrans,1,1,1,1,2147483647,halfsecond,D,
net_buf_size,{all},{all},{all},0,128,string,D,
net_buf_type,{all},{all},{all},0,128,string,D,
net_malloc_police,0,0,0,0,2147483647,numeric,D,
nonlocsrcroute,1,0,0,0,1,boolean,D,
nstrpush,8,8,8,8,32767,numeric,R,
passive_dgd,0,0,0,0,1,boolean,D,
pmtu_default_age,10,10,10,0,32767,minute,D,
pmtu_expire,10,10,10,0,32767,minute,D,
pmtu_rediscover_interval,30,30,30,0,32767,minute,D,
psebufcalls,20,20,20,20,2147483647,numeric,I,
psecache,1,1,1,0,1,boolean,D,
pseintrstack,12288,12288,12288,12288,2147483647,byte,R,
psetimers,20,20,20,20,2147483647,numeric,I,
rfc1122addrchk,0,0,0,0,1,boolean,D,
rfc1323,0,0,0,0,1,boolean,C,
rfc2414,0,0,0,0,1,boolean,C,
route_expire,1,1,1,0,1,boolean,D,
routerevalidate,0,0,0,0,1,boolean,D,
rto_high,64,64,64,2,2147483647,roundtriptime,R,rto_low,
rto_length,13,13,13,1,64,roundtriptime,R,
rto_limit,7,7,7,1,64,roundtriptime,R,rto_high,rto_low,
rto_low,1,1,1,1,63,roundtriptime,R,rto_high,
sack,0,0,0,0,1,boolean,C,
```

```
sb_max,1048576,1048576,1048576,1,2147483647,byte,D,
send_file_duration,300,300,300,0,2147483647,second,D,
site6_index,0,0,0,0,32767,numeric,D,
sockthresh,75,85,85,0,100,%_of_thewall,D,
sodebug,0,0,0,0,1,boolean,C,
somaxconn,1024,1024,1024,0,32767,numeric,C,
strctlsz,1024,1024,1024,0,32767,byte,D,
strmsgsz,0,0,0,0,32767,byte,D,
strthresh,85,85,85,0,100,%_of_thewall,D,
strturncnt,15,15,15,1,2147483647,numeric,D,
subnetsarelocal,1,1,1,0,1,boolean,D,
tcp_bad_port_limit,0,0,0,0,2147483647,numeric,D,
tcp_ecn,0,0,0,0,1,boolean,C,
tcp_ephemeral_high,65535,65535,65535,32769,65535,numeric,D,tcp_ephemeral_low,
tcp_ephemeral_low,32768,32768,32768,1024,65534,numeric,D,tcp_ephemeral_high,
tcp_finwait2,1200,1200,1200,0,65535,halfsecond,D,
tcp_init_window,0,0,0,0,32767,byte,C,
tcp_inpcb_hashtab_siz,24499,24499,24499,1,999999,numeric,R,
tcp_keepcnt,8,8,8,0,2147483647,numeric,D,
tcp_keepidle,14400,14400,14400,1,2147483647,halfsecond,C,
tcp_keepinit,150,150,150,1,2147483647,halfsecond,D,
tcp_keepintvl,150,150,150,1,32767,halfsecond,C,
tcp_limited_transmit,1,1,1,0,1,boolean,D,
tcp_maxburst,0,0,0,0,32767,numeric,D,
tcp_mssdflt,1460,1460,1460,1,65535,byte,C,
tcp_nagle_limit,65535,65535,65535,0,65535,byte,D,
tcp_ndebug,100,100,100,0,32767,numeric,D,
tcp_newreno,1,1,1,0,1,boolean,D,
tcp_nodelayack,0,0,0,0,1,boolean,D,
tcp_pmtu_discover,1,1,1,0,1,boolean,D,
tcp_recvspace,16384,16384,16k,4096,2147483647,byte,C,sb_max,
tcp_sendspace,16384,16384,16384,4096,2147483647,byte,C,sb_max,
tcp_timewait,1,1,1,1,5,15_second,D,
tcp_ttl,60,60,60,1,255,0.6_second,C,
thewall,1048576,1048576,1048576,0,1048576,kbyte,S,
udp_bad_port_limit,0,0,0,0,2147483647,numeric,D,
udp_ephemeral_high,65535,65535,65535,32769,65535,numeric,D,udp_ephemeral_low,
udp_ephemeral_low,32768,32768,32768,1024,65534,numeric,D,udp_ephemeral_high,
udp_inpcb_hashtab_siz,24499,24499,24499,1,83000,numeric,R,
udp_pmtu_discover,1,1,1,0,1,boolean,D,
udp_recvspace,42080,42080,42080,4096,2147483647,byte,C,sb_max,
udp_sendspace,9216,9216,9216,4096,2147483647,byte,C,sb_max,
udp_ttl,30,30,30,1,255,second,C,
udpcksum,1,1,1,0,1,boolean,D,
use_isno,1,1,1,0,1,boolean,D,
use_sndbufpool,1,1,1,0,1,boolean,R,
[p630n04][/]>
```

To display a specific tunable using the **no -x** command see Example 6-65.

Example 6-65 The no -x tcp_recvspace

```
[p630n04][/]> no -x tcp_recvspace
tcp_recvspace,16384,16384,16k,4096,2147483647,byte,C,sb_max,
[p630n04][/]>
```

The output of the **no -x** command lists the following attributes. “tunable, current, default, reboot, min, max, unit, type” in a comma separated list.

As can be seen from the output the command the current value of the tunable is 32K, the default value is 16K, the minimum value is 4K, the maximum value is 2G-1. The unit used for this tunable is bytes, and the tunable type is C (Connect), which means if the tunable is changed the changes will only take effect for new connections. This tunable is also dependent on the **sb_max** tunable.

To better understand what a specific tunable is used for you can use the **no -h** command to display a description of the tunable. As can be seen in Example 6-66 a very detailed explanation is given about the **tcp_recvspace** tunable, when using the **no -h** option.

Example 6-66 The no -h example

```
[p630n04][/home/hennie]> no -h tcp_recvspace

Help for tunable tcp_recvspace:

Specifies the system default socket buffer size for receiving data. This
affects the window size used by TCP. Setting the socket buffer size to 16KB
(16,384) improves performance over Standard Ethernet and token-ring networks.
The default is a value of 4096; however, a value of 16,384 is set automatically
by the rc.net file or the rc.bsdnet file (if Berkeley-style configuration is
issued). Lower bandwidth networks, such as Serial Line Internet Protocol
(SLIP), or higher bandwidth networks, such as Serial Optical Link, should have
different optimum buffer sizes. The optimum buffer size is the product of the
media bandwidth and the average round-trip time of a packet. In AIX 4.3.3 and
later versions, the tcp_recvspace network option can also be set on a per
interface basis via the ifconfig command. The tcp_recvspace attribute must
specify a socket buffer size less than or equal to the setting of the sb_max
attribute. tcp_recvspace is a Connect attribute, but for daemons started by
inetd, the following command needs to be executed: 'stopsrc -s inetd ; startsrc
-s inetd'
```

To change the value of a tunable with **no -o** see Example 6-67 on page 412. The **no -o** command is used to change the value of the **tcp_recvspace** to 32768.

Example 6-67 The no -o

```
[p630n04][/home/hennie]> no -o tcp_recvspace=32768
Setting tcp_recvspace to 32768
Change to tunable tcp_recvspace, will only be effective for future connections
[p630n04][/home/hennie]>
```

All tunables set by the no -o command is only valid for the duration that the system is up. If the system is rebooted it automatically uses the default values of the tunables.

AIX 5.2 introduced a more flexible and centralized mode for setting most of the AIX kernel tuning parameters. It is now possible to make permanent changes without editing any **rc** files. This is achieved by placing the reboot values for all tunable parameters in a new **/etc/tunables/nextboot** stanza file. When the machine is rebooted, the values in that file are automatically applied.

The **/etc/tunables/lastboot** stanza file is automatically generated with all the values that were set immediately after the reboot. This provides the ability to return to those values at any time. The **/etc/tunables/lastboot.log** log file records any changes made or that could not be made during reboot. There are sets of SMIT panels and a Web-based System Manager plug-in also available to manipulate current and reboot values for all tuning parameters, as well as the files in the **/etc/tunables** directory.

Pre 5.2 compatibility mode considerations

Pre 5.2 compatibility mode is controlled by the pre520tune attribute of sys0. 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 rc 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.

The following commands were introduced in AIX 5.2 to modify the tunables files (see Table 6-2 on page 413).

Table 6-2 AIX 5.2 Tunables commands

Command	Purpose
tunsave	Saves values to a stanza file
tunrestore	Applies applicable parameter values that are specified in a file
tuncheck	Validates files that are created manually
tundefault	Resets tunable parameters to their default values

To make any changes to no tunables be effective after a reboot the -r or -p commands can be used.

When using the -r option with the no command will have the tunable change only take effect after a reboot.

In Example 6-68 we are changing the value of the **tcp_recvspace** to 16k which is the default but we only want changes to take effect after the reboot.

Example 6-68 The no -r -o tcp_recvspace

```
[p630n04][/etc/tunables]> no -r -o tcp_recvspace=16k
Setting tcp_recvspace to 16k in nextboot file
Warning: changes will take effect only at next reboot
[p630n04][/etc/tunables]>
```

As explained earlier, the /etc/tunables/nextboot file is used to set values after a reboot (see Example 6-69).

Example 6-69 Contents of the /etc/tunables/nextboot file

```
[p630n04][/etc/tunables]> more /etc/tunables/nextboot
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog. #
# bos530 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:
```

```
no:
        tcp_recvspace = "16k"
[p630n04][/etc/tunables]>
```

The **vmo, schedo, ioo, no** and **nfso** commands all make use of this file to store their tunable values that will be set at next reboot.

After we executed the **no -r -o tcp_recvspace** command an entry gets made in the **/etc/tunables/nextboot** file. In Example 6-69 on page 413 you will notice that the **tcp_recvspace** value is set to 16k this will be set when the system is rebooted.

Also if you query the current value of the **tcp_recvspace** tunable you will note that the tunable value has not changed. See Example 6-70.

Example 6-70 The no -o tcp_recvspace to display the current value

```
[p630n04][/etc/tunables]> no -o tcp_recvspace
tcp_recvspace = 32768
[p630n04][/etc/tunables]>
```

To have no tunable values take effect immediately and after a reboot use the **no -p** command. See Example 6-71. This will have the current **no** tunable change to the specified value as well as an entry be made in the **/etc/tunables/nextboot** file.

Example 6-71 The no -p -o tcp_recvspace command

```
[p630n04][/etc/tunables]> no -p -o tcp_recvspace=16k
Setting tcp_recvspace to 16k
Setting tcp_recvspace to 16k in nextboot file
Change to tunable tcp_recvspace, will only be effective for future connections
[p630n04][/etc/tunables]>
```

If you want to change the value of a tunable to its default value make use of the **no -d** command to change a specific value.

In Example 6-72 we are using the **no -d** command to change the value of the **tcp_recvspace** tunable to its default value which is 16384 bytes.

Example 6-72 The no -d tcp_recvspace command

```
[p630n04][/]> no -d tcp_recvspace
Setting tcp_recvspace to 16384
[p630n04][/]>
```

Note: If you use the no -d command to chance a tunable to its default value, the /etc/tunables/nextboot file is not updated. Use the no -p -d combination to have a command set to its default and update the next reboot value.

6.7.2 The Interface Specific Network Options (ISNO)

In AIX 5L V5.2 and later Interface Specific Network Options (ISNO) made it possible to define certain no option on a specific interface.

In Example 6-73 we are using the **lsattr** to display information about a particular interface you will note at the end of the report that it lists attributes that would normally be set with the no command.

When using no to set certain tunables they are defined system wide, if tunables are defined on a particular interface using the **chdev** command, they will be defined for the particular interface giving you better manageability.

Example 6-73 The lsattr -El en0 command

```
[p630n04][/home/hennie/nfso]> lsattr -El en0
alias4                      IPv4 Alias including Subnet Mask           True
alias6                      IPv6 Alias including Prefix Length         True
arp           on            Address Resolution Protocol (ARP)          True
authority                   Authorized Users                           True
broadcast                   Broadcast Address                          True
mtu           1500          Maximum IP Packet Size for This Device     True
netaddr       192.168.100.34 Internet Address                          True
netaddr6                    IPv6 Internet Address                      True
netmask       255.255.255.0 Subnet Mask                                True
prefixlen                   Prefix Length for IPv6 Internet Address    True
remmtu        576           Maximum IP Packet Size for REMOTE Networks True
rfc1323                     Enable/Disable TCP RFC 1323 Window Scaling True
security      none          Security Level                             True
state         up            Current Interface Status                   True
tcp_mssdflt                 Set TCP Maximum Segment Size               True
tcp_nodelay                 Enable/Disable TCP_NODELAY Option          True
tcp_recvspace               Set Socket Buffer Space for Receiving      True
tcp_sendspace               Set Socket Buffer Space for Sending        True
[p630n04][/home/hennie/nfso]>
```

Note: If no value is displayed next to the ISNO fields, the no values for that tunable is used by the interface.

In Example 6-74 on page 416 we use the **chdev** command to change an ISNO attribute of an interface.

Example 6-74 The chdev command to change ISNO value

```
[p630n04][/home/hennie]> chdev -l en0 -a tcp_recvspace=32768
en0 changed
[p630n04][/home/hennie]> lsattr -El en0
```

In Example 6-75 you will note that after changing the tcp_recvspace attribute of the interface, the value gets displayed next to the output of the command.

Example 6-75 The lsattr to display ISNO values

```
[p630n04][/home/hennie]> lsattr -El en0
alias4                      IPv4 Alias including Subnet Mask          True
alias6                      IPv6 Alias including Prefix Length        True
arp           on            Address Resolution Protocol (ARP)         True
authority                   Authorized Users                          True
broadcast                   Broadcast Address                         True
mtu           1500          Maximum IP Packet Size for This Device    True
netaddr       192.168.100.34 Internet Address                         True
netaddr6                    IPv6 Internet Address                     True
netmask       255.255.255.0 Subnet Mask                               True
prefixlen                   Prefix Length for IPv6 Internet Address   True
remmtu        576           Maximum IP Packet Size for REMOTE Networks True
rfc1323                     Enable/Disable TCP RFC 1323 Window Scaling True
security      none          Security Level                            True
state         up            Current Interface Status                  True
tcp_mssdflt                 Set TCP Maximum Segment Size              True
tcp_nodelay                 Enable/Disable TCP_NODELAY Option         True
tcp_recvspace 32768         Set Socket Buffer Space for Receiving     True
tcp_sendspace               Set Socket Buffer Space for Sending       True
[p630n04][/home/hennie]>
```

6.7.3 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 is located in /usr/sbin/nfso and is part of the bos.net.nfs.client fileset, which is installable from the AIX base installation media.

nfso syntax

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

```
nfso [ -p | -r ] [ -c ] { -o Tunable[ =Newvalue ] }
nfso [ -p | -r ] { -d Tunable }
```

```
nfso [ -p | -r ] -D
nfso [ -p | -r ] -a [ -c ]
nfso -?
nfso -h Tunable
nfso -l Hostname
nfso [ -c ]
nfso -L [ Tunable ]
nfso -x [ Tunable ]
```

Multiple flags -o, -d, and -L are allowed.

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.

Examples for nfso

This section shows some examples of the **nfso** command.

Listing all of the tunables and their current values

Example 6-76 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 6-76 Display and store in a file the current NFS network variables

```
[p630n04][/home/hennie/nfso]> nfso -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
           lockd_debug_level = 0
           statd_debug_level = 0
           statd_max_threads = 50
    nfs_v4_fail_over_timeout = 0
             utf8_validation = 1
                 nfs_v4_pdts = 1
              nfs_v4_vm_bufs = 1000
[p630n04][/home/hennie/nfso]>
```

Displaying characteristics of all tunables

Example 6-77 displays the output when using the **nfso** command with the -L flag to display all of the variables and their characteristics.

Example 6-77 Listing of nfso tunables

```
[p630n04][/home/hennie/nfso]> nfso -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    8K+1   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      64K-1  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_pdts               1      1      1      1      8      PDTs            M
--------------------------------------------------------------------------------
nfs_v3_pdts               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         0      0      0      0      10     Level           D
--------------------------------------------------------------------------------
statd_debug_level         0      0      0      0      10     Level           D
--------------------------------------------------------------------------------
statd_max_threads         50     50     50     1      1000   Threads         D
--------------------------------------------------------------------------------
nfs_v4_fail_over_timeout  0      0      0      0      3600   Seconds         D
--------------------------------------------------------------------------------
utf8_validation           1      1      1      0      1      On/Off          D
--------------------------------------------------------------------------------
nfs_v4_pdts               1      1      1      1      8      PDTs            M
--------------------------------------------------------------------------------
nfs_v4_vm_bufs            1000   1000   1000   512    5000   Bufs            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: 2^10       G = Giga: 2^30       P = Peta: 2^50
    M = Mega: 2^20       T = Tera: 2^40       E = Exa: 2^60
[p630n04][/home/hennie/nfso]>
```

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.

Displaying and changing a tunable with the nfso command

Example 6-78 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 6-78 Displaying and changing a tunable

```
# nfso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 1
# nfso -o nfs_dynamic_retrans=0
Setting nfs_dynamic_retrans to 0
# nfso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0
```

Resetting a tunable value to its default

Example 6-79 shows that a value was changed in Example 6-78 can be reset to the default by using the -d flag.

Example 6-79 Restoring default tunable value

```
# nfso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0
# nfso -d nfs_dynamic_retrans
Setting nfs_dynamic_retrans to 1
# nfso -o nfs_dynamic_retrans
```

```
nfs_dynamic_retrans= 1
```

Displaying help information about a tunable

Using the -h flag with the **nfso** command displays information about that specific variable, as shown in Example 6-80.

Example 6-80 Getting information about a tunable

```
[p630n04][/home/hennie/nfso]> nfso -h nfs_dynamic_retrans
Help for tunable 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 timeo 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.
[p630n04][/home/hennie/nfso]>
```

Permanently changing an nfso 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 6-81 displays the contents of the /etc/tunables/nextboot file with no information about the nfs_dynamic_retrans variable. Then by executing **nfso -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 6-81 Permanently changing the nextboot file

```
# nfso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 1
# cat /etc/tunables/nextboot
ioo:
nfso:
        nfs_v3_vm_bufs = "5000"
        nfs_v2_vm_bufs = "5000"
vmo:
        maxperm% = "50"
        maxclient% = "50"
        spec_dataseg_int = "0"

# nfso -p -o nfs_dynamic_retrans=0
# cat /etc/tunables/nextboot
```

```
ioo:
nfso:
        nfs_dynamic_retrans = "0"
        nfs_v3_vm_bufs = "5000"
        nfs_v2_vm_bufs = "5000"
vmo:
        maxperm% = "50"
        maxclient% = "50"
        spec_dataseg_int = "0"
# nfso -o nfs_dynamic_rtrans
nfs_dynamic_retrans= 0
```

Changing a tunable after reboot

By using the -r flag the change to a variable will only take effect after a reboot. In Example 6-82 we used the **nfso** 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 **nfso** 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 6-82 Changing a parameter after next reboot

```
# nfso -o nfs_dynamic_retrans
nfs_dynamic_retrans= 0

# cat /etc/tunables/nextboot
ioo:
nfso:
        nfs_dynamic_retrans = "0"
        nfs_v3_vm_bufs = "5000"
        nfs_v2_vm_bufs = "5000"
vmo:
        maxperm% = "50"
        maxclient% = "50"
        spec_dataseg_int = "0"

# nfso -ro nfs_dynamic_rtrans=1

# nfso -o nfs_dynamic_rtrans
nfs_dynamic_rtrans=0

# 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"
```

7

Storage analysis and tuning

In this chapter we discuss how to monitor and tune disk I/O. The disk storage is a key component which determines the performance of several other subsystems.

The physical aspect (adapters, disks, etc.) is as critical in system performance as the device drivers, LVM and file system layers. Thus, we present performance monitoring at all levels together with some tuning techniques.

These commands are covered in this chapter:

- For monitoring:
  - iostat, filemon, fileplace, lslv, lspv, lsvg, lvmstat, sar -d
- For tuning:
  - lsdev, rmdev, mkdev, lscfg, lsattr, chdev, ioo, lvmo, vmo

7.1 Data placement and design

There is a vast gap between disk metrics and system metrics. In fact it's entirely possible to use the same hardware and application and get vastly different results in system performance by varying the data layout. For optimal performance, the data access patterns of the application and the subsequent workload need to match the data layout.

Data layout goals

- Minimize I/O service times
- Balance I/O across all disks
- Keep I/O localized and sequential

7.1.1 AIX I/O stack

The AIX logical volume manager (LVM) provides flexibility in specifying the data layout. A basic understanding of the AIX I/O stack is important in order to effectively monitor and tune an AIX system.

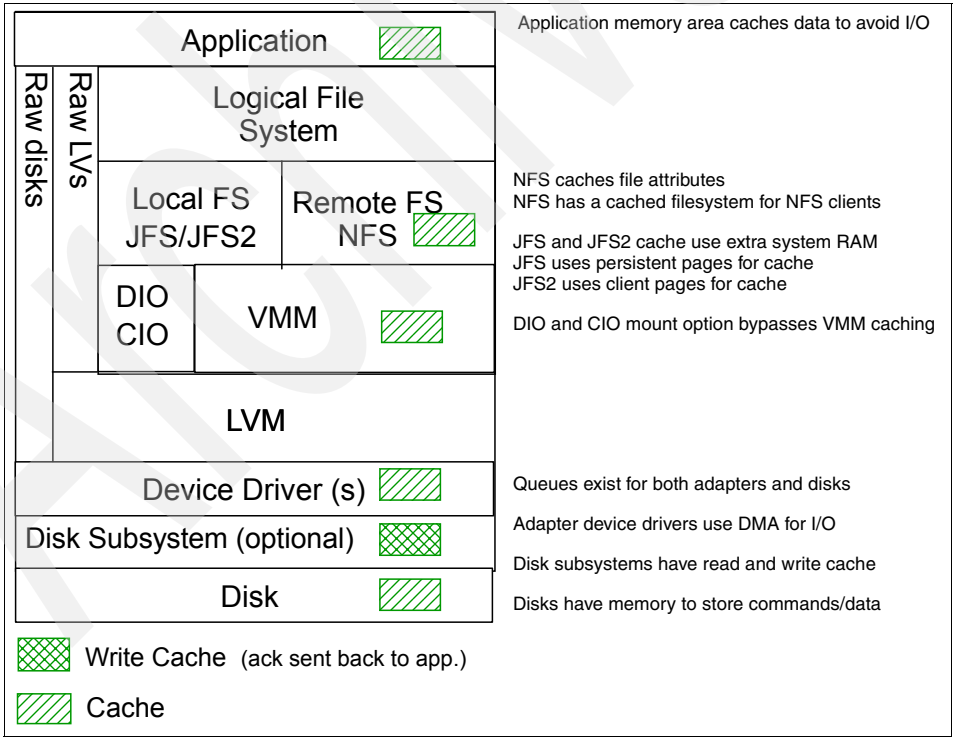

Figure 7-1 AIX I/O Stack

Figure 7-1 on page 426 shows the path a data request moves through from the initiating application down to the physical disk. Some applications manage their own data buffering, and system performance can be improved by bypassing file system caching to avoid the condition known as double buffering. Double buffering, where data resides in main memory more than once, increases CPU load and reduces available main memory.

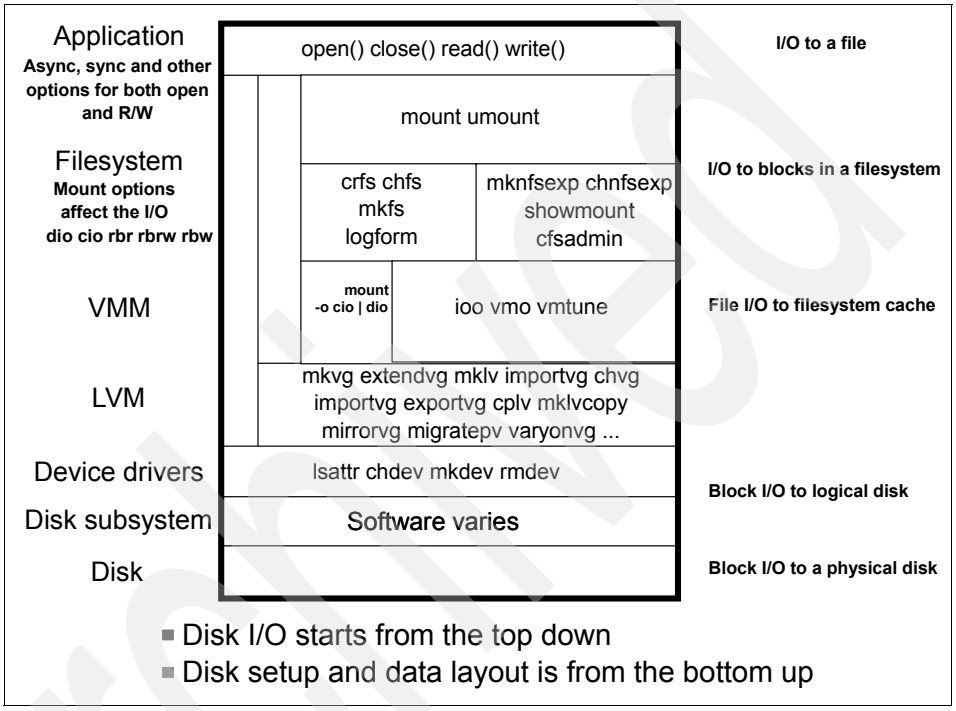

Figure 7-2 AIX I/O Management

Figure 7-2 shows the I/O stack for AIX. When tuning, we have to be aware of all the layers, as each layer impacts performance, and there are knobs to turn at each layer.

I/O operations can be coalesced into fewer I/Os, or broken up into more I/Os as they go up and down through the I/O layers. Generally, one gets better performance, in MB/s, with fewer but larger I/Os. With fewer I/Os, there's less CPU overhead to handle the requests.

Note that system setup, from a data layout viewpoint, is generally done from the bottom up. First the disk subsystem is configured, then the device layer (hdisks, vpaths, etc.), then the LVM layer (VGs then LVs) then the filesystems, and finally the files.

The disk interconnection technology exists below the device driver level, sometimes prior to the disk subsystem and within the disk subsystem if it exists. The advent of SANs, NAS and iSCSI have additional latencies for getting the I/O across the disk network.

Direct I/O bypasses the use of JFS cache, and is beneficial in some circumstances, e.g., when updating log files. Direct I/O can be specified either by a mount option mount -o dio or via a program opening a file with the O_DIRECT open flag. Another mount option exists for multi indirect support (useful when copying many file >32 KB) to allow more memory segments to be used for inode caching: mount -o mind, which applies to AIX 4.3.3 only.

Synchronous and asynchronous I/O refer to whether or not the application is coded to wait for the I/O to complete (synchronous-wait, asynchronous-don't wait). Default write I/Os to JFS or JFS2 are asynchronous unless specifically coded to be synchronous.

Most database applications use the character device (the r device, e.g. /dev/r<lvname>) for I/O though it's also possible to use the block device.

NFS file attribute caching is specified via the actimeo, acregmin, acregmax, acdirmin and acdirmax attributes in /etc/filesystems. It also allows a cached filesystem on NFS clients via the cfsadmin command. When caching is specified, files from the NFS server will be cached to local disk.

Figure 7-2 on page 427 is a companion figure to Figure 7-1 on page 426, and shows the basic commands used to manage each layer. These commands will be discussed further in 7.2, "Monitoring" on page 433 and in 7.3, "Tuning" on page 480.

We will now discuss each of the layers of the AIX I/O stack and what to keep in mind when setting up each layer.

7.1.2 Physical disk and disk subsystem

Correctly sizing and purchasing the appropriate storage is critical in obtaining desired performance. When purchasing disk, the size, speed (rpm), number, and attachment type all play a role in the overall performance of the system. Capacity of physical disks is increasing much faster than throughput (both I/Os per second and MBs per second). This trend leads to buying enough capacity, but ending up short on throughput.

Most physical disk is purchased and installed within a disk subsystem like the IBM 2105 Enterprise Storage System (ESS) or the IBM DS4000 (FAStT) storage systems. The disk in these storage systems is then configured into RAID groups. The RAID groups are then sliced or carved into logical disks commonly referred

to as LUNs (logical unit numbers) and then assigned to servers. A RAID LUN whether it is comprised of two or more disks, appears to the AIX system as a single physical disk.

For the purposes of this document, we will focus on disk residing in a disk subsystem. The AIX documentation provides details on direct attached physical disk.

RAID 5 versus Mirroring

The two main choices to protect against data loss due to disk failure are a RAID 5 approach or some type of mirroring (RAID 1, RAID 10, LVM mirroring). Neither is better than the other in all situations.

7.1.3 Device drivers and adapters

At this time, several high-speed adapters for connecting disk drives are available (SCSI, SSA, FC); however, if you build a configuration with a lot of these adapters on one system bus, you may have fast adapters, but the bus becomes the performance bottleneck. Therefore, it is always better to spread these fast adapters across several busses.

Attaching forty fast disks or LUNs to one disk adapter may result in low disk performance. Although the disks or the RAID-based LUNs are fast, all the data must go through that one adapter. This adapter gets overloaded, resulting in low performance. Adding more disk adapters is a good thing to do when a lot of disks are used, especially if those disks are used intensively.

Beside using multiple adapters, the device drivers for the adapters should support load balancing (Multi-Path I/O - MPIO). MPIO operation is dependant on the adapter (AIX) device driver, and also on the storage subsystem used.

MPIO is short for multi-path I/O. MPIO is the ability to uniquely detect, configure and manage a device on multiple physical paths. MPIO in AIX consists of enhancements to the configuration subsystem and device drivers. MPIO also includes a new module called a path control module or PCM for short. The PCM provides the ability for a device driver to be tailored to the capabilities of the device being managed. This is an important change in allowing a device vendor to provide code to modify the behavior of the AIX base device driver. The MPIO components are (see also Figure 7-3 on page 430):

- Device Configuration Database (ODM)
- Device Configuration Commands
- Device Configuration Methods
- Device Drivers
- Path Control Module (PCM)

– User Interfaces (SMIT and WEBSM)

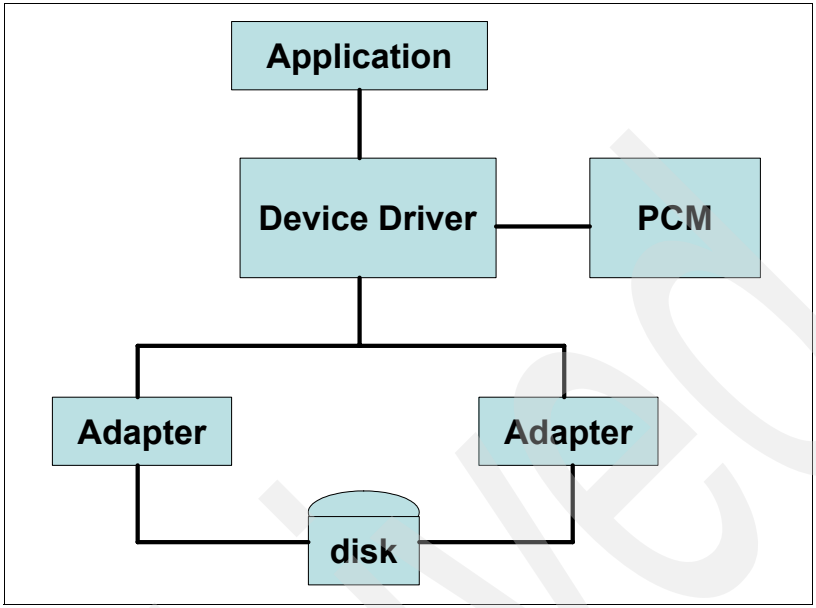

Figure 7-3 MPIO device driver structure

7.1.4 Volume groups and logical volumes

Many options of the LVM were designed to deal with direct attached storage (DAS). LVM options like specifying the Inter-Physical Volume Allocation Policy do not have any effect on RAID LUNs. Data availability options like Logical Partition copies may not be necessary when the storage subsystem is handling redundancy.

Figure 7-4 on page 431 presents the logical diagram for the LVM. For details, refer to the redbook *AIX Logical Volume Manager from A to Z, Introduction and Concepts*, SG24-5432.

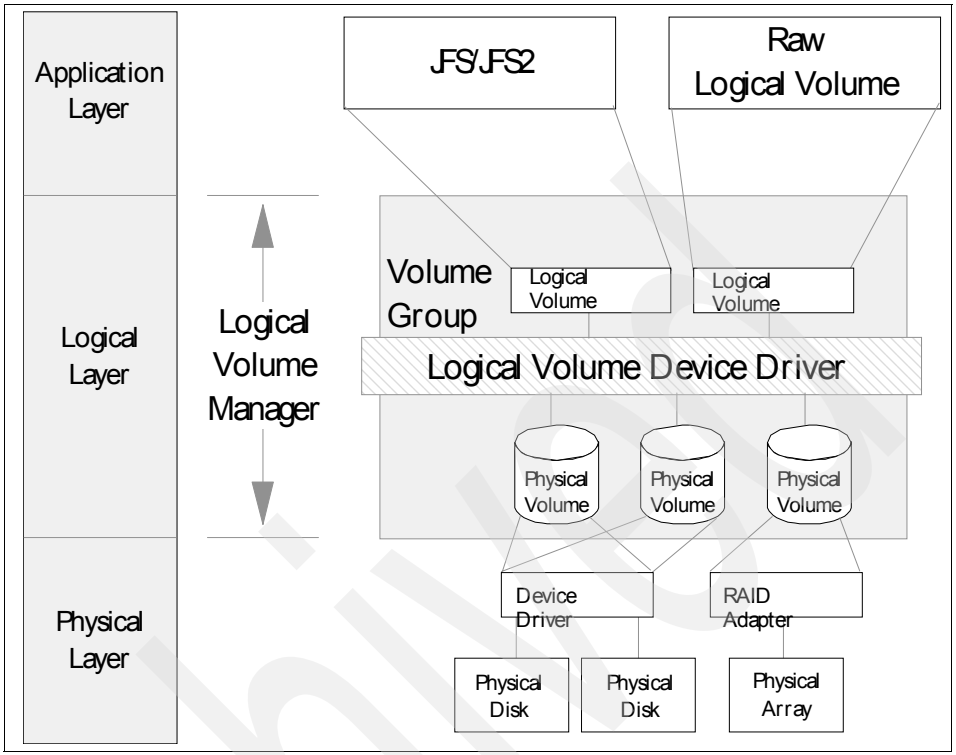

Figure 7-4 Logical Volume Manager diagram

7.1.5 VMM and direct I/O

When you are processing normal I/O to JFS or JFS2 files, the I/O goes from the application buffer to the VMM and from there to the JFS. The contents of the buffer could get cached in RAM through the VMM's use of real memory as a file buffer cache. If the file cache hit rate is high, then this type of cached I/O is very effective by improving performance of JFS I/O. But applications that have poor cache hit rates or applications that do very large I/Os may not get much benefit from the use of normal cached I/O. In operating system Version 4.3, direct I/O was introduced as an alternative I/O method for JFS files.

Direct I/O is only supported for program working storage (local persistent files). The main benefit of direct I/O is to reduce CPU utilization for file reads and writes by eliminating the copy from the VMM file cache to the user buffer. If the cache hit rate is low, then most read requests have to go to the disk. Writes are faster with normal cached I/O in most cases. But if a file is opened with O_SYNC or O_DSYNC (see Using sync/fsync Calls), then the writes have to go to disk. In

these cases, direct I/O can benefit applications because the data copy is eliminated.

Even though the use of direct I/O can reduce CPU usage, it typically results in longer elapsed times, especially for small I/O requests, because the requests would not be cached in memory.

Direct I/O reads cause synchronous reads from the disk, whereas with normal cached policy, the reads may be satisfied from the cache. This can result in poor performance if the data was likely to be in memory under the normal caching policy. Direct I/O also bypasses the VMM read-ahead algorithm because the I/Os do not go through the VMM. The read-ahead algorithm is very useful for sequential access to files because the VMM can initiate disk requests and have the pages already be resident in memory before the application has requested the pages. Applications can compensate for the loss of this read-ahead by using one of the following methods:

- Issuing larger read requests (minimum of 128 K)
- Issuing asynchronous direct I/O read-ahead by the use of multiple threads
- Using the asynchronous I/O facilities such as aio_read() or lio_listio()

Direct I/O writes bypass the VMM and go directly to the disk, so that there can be a significant performance penalty; in normal cached I/O, the writes can go to memory and then be flushed to disk later (write-behind). Because direct I/O writes do not get copied into memory, when a sync operation is performed, it will not have to flush these pages to disk, thus reducing the amount of work the syncd daemon has to perform.

7.1.6 JFS/JFS2 file systems

With the introduction of AIX 5L IBM introduced a new file system referred to as Enhanced JFS (JFS2) that provides greater scalability than the previous file system JFS. Enhanced JFS is designed and optimized for a 64-bit kernel environment taking full advantage of 64-bit functionality. JFS2 is the default file system for a 64-bit kernel installation.

JFS is the default file system for a 32-bit kernel installation. JFS2 is supported in a 32-bit kernel environment. It is recommended to use a 64-bit kernel to achieve maximum performance for JFS2.

Table 7-1 shows a comparison between JFS and JFS2.

Table 7-1 JFS/JFS2 comparison

Function	JFS	JFS2
Fragments/Block size	512- 4096 Frags	512- 4096 Frags

Function	JFS	JFS2
File size	64 GBytes	4 Petabytes
Filesystem size	1 TBytes	4 Petabytes
Number of inodes	Limited at file system creation or expansion	Dynamic, limited by disk space
Directory organization	Standard	Faster file lookups compared to JFS
Online defragmentation	Yes	Yes
Compression	Yes	No
Quotas	Yes	Yes
Fsck on large filesystems	Slow	Fast
Deferred update	Yes	No

For more information refer to *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905.

7.2 Monitoring

Multi-resource system monitoring tools such as **vmstat** provide the first indicators that a disk related bottleneck may exist. This section provides details on disk specific tools that are used to further investigate disk performance issues.

The goal of monitoring is to ensure that you, the system's administrator, are warning of impending problems and slow-downs *before* your customers tell you about them.

Monitoring should be proactive and exception-based. When something is out of spec or out of norm, an alert should be sent. You should not rely only on reviews of logs or reports after the fact.

7.2.1 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

Beginning with AIX 5.3, the **iostat** command reports the number of physical processors consumed (physc) and the percentage of entitlement consumed (% entc) in micro-partitioning and simultaneous multi-threading environments. These metrics will only be displayed on micro- partitioning/simultaneous multi-threading environments.

AIX 5.3 also introduces enhancements to the **iostat** command to allow the user to obtain asynchronous I/O (AIO) statistics.

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

iostat syntax

The syntax of the **iostat** command is

```
iostat [ -a ] [ -s ] [-t  ] [ -T ] [ -d [ -m ] [ -A ] [ -P ] [ -q | -Q ] [ -l ]
[ Drives ... ] [ Interval] [ Count ]
```

Useful combinations

- **iostat -d hdisk4** — **collect disk stats for hdisk4**
- **iostat -a 5** — **adapter stats every 5 seconds**
- **iostat 5 60** — **stats every 5 seconds for 5 minutes**

Information about measurement and sampling

The iostat command generates different types of reports:

- tty and CPU utilization
- Disk utilization
- System throughput
- Adapter throughput
- Asynchronous I/O statistics

Examples

Tip: The average I/O size can be calculated by dividing the value for Kbps by tps (avg I/O = Kbps/tps). 2144.0/67.0=32Kb

Tip: The interval used for an iostat report can be calculated by summing the Kb_read and Kb_wrtn values and dividing by the data rate Kbps ((Kb_read + Kb_wrtn)/Kbps). (2016+128)/2144=1 second.

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 7-1 shows the disk utilization report.

Example 7-1 Disk utilization report

```
[p630n06][/home/guest/2105]> iostat -d

System configuration: lcpu=4 drives=9

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk3            0.0      0.0      0.0       9492       444
hdisk2            0.6    126.4      1.4   45393397  27468855
hdisk1            0.4     52.2      0.8   27393610   2730728
hdisk0            0.4     12.0      0.8     969337   5921412
cd0               0.0      0.0      0.0          0         0
dac0              0.0      0.0      0.0          0         0
dac0-utm          0.0      0.0      0.0          0         0
hdisk5            0.0      1.6      0.0      29753    874124
hdisk4            0.0      0.0      0.0      21223        16
```

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 reports will reflect the amount of I/O on the system over the last interval.

Disk utilization report for MPIO

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.

If you use **iostat -m**, you can see input/output statistics on MPIO as shown in Example 7-2. In this example hdisk0 has a single path, and hdisk4 has two paths.

Example 7-2 Output of iostat -m

```
[p630n06][/home/guest/2105]> iostat -m hdisk0
...system information omitted...

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.4      11.9       0.8     972949   5921780

Paths:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
Path0            0.4      11.9       0.8     972949   5921780

[p630n06][/home/guest/2105]> iostat -m hdisk4
...system information omitted...

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk4           0.0       0.0       0.0      21223        24

Paths:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
Path1            0.0       0.0       0.0           0         0
Path0            0.0       0.0       0.0      21223        24
```

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. Disable disk history if not needed.

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 7-3 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 7-3 Adapter throughput report

```
[p630n06][/]> iostat -a

System configuration: lcpu=4 drives=7

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.1       2748.5               3.0       1.0       95.6       0.4

Adapter:                  Kbps      tps    Kb_read   Kb_wrtn
scsi0                   192.8       3.1   73769489  37524044

Paths/Disk:       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk3_Path0           0.0      0.0       0.0       9492       444
hdisk2_Path0           0.7    128.6       1.4   45393397  28866430
hdisk1_Path0           0.4     52.2       0.8   27393610   2735094
hdisk0_Path0           0.4     11.9       0.8     972990   5922076

Adapter:                  Kbps      tps    Kb_read   Kb_wrtn
ide0                      0.0       0.0          0         0

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

Adapter:                  Kbps      tps    Kb_read   Kb_wrtn
fcs0                      4.5       0.0      50976   2550116

Paths/Disk:       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk5_Path1           0.0      0.0       0.0          0         0
hdisk5_Path0           0.0      4.5       0.0      29753   2550092
hdisk4_Path1           0.0      0.0       0.0          0         0
hdisk4_Path0           0.0      0.0       0.0      21223        24
```

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 reports reflect the amount of I/O on the system over the last interval (current activity).

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.

Asynchronous I/O statistics

AIX 5L Version 5.3 introduces enhancements to the **iostat** command, these allow the user to obtain AIO statistics. In previous versions of AIX there were no tools available to monitor AIO. From Version 5.3 the performance kernel libraries are modified to obtain AIO statistics and the **iostat** command is enhanced to monitor also the AIO.

The **iostat** command reports CPU and I/O statistics for the system, adapters, TTY devices, disks and CD-ROMs. This command is enhanced by new monitoring features and flags for getting the AIO and the POSIX AIO statistics.

The following new flags are added to the **iostat** command:

-A Reports AIO statistics along with the existing output.

-P Reports AIO statistics using the POSIX AIO calls. If not specified then the Legacy AIO statistics are returned.

-q Reports each AIO queue's request count.

-Q Reports AIO queues associated with each mounted file system and the queue request count.

-l Displays the data in a 132 column width. This flag is simply a formatting flag.

When using the -A option, the output of iostat gives additional statistics of the AIO. The following information is added:

avgc Average global non-fastpath AIO request count per second for the specified interval.

avfc Average global AIO fastpath request count per second for the specified interval

maxg Maximum global non-fastpath AIO request count since the last time this value was fetched

maxf Maximum fastpath request count since the last time this value was fetched

maxr Maximum AIO requests allowed. This is the AIO device maxreqs attribute.

When the AIO device driver is not configured in the kernel, the **iostat -A** command gives an error message that the AIO is not loaded like in the following example.

Example 7-4 The iostat with AIO not configured

```
[p630n06][/]> iostat -Aq

System configuration: lcpu=4

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow
iostat: 0551-157 Asynchronous I/O not configured on the system.
```

In order to use the AIO drivers we have to enable the AIO device driver using the **mkdev -l aio0** command or through **smit aio**. In order to enable the POSIX AIO device drivers, you have to use the **mkdev -l posix_aio0** or **smit posixaio** commands.

Once enabled, iostat will report AIO statistics as in Example 7-5.

Example 7-5 iostat -A displays basic AIO statistics

```
[p630n06][/]> iostat -A

System configuration: lcpu=4 drives=7

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow
       0    0    0    0 4096            3.0  1.0  95.6  0.4
...lines omitted...
```

Check the avgc as it shows the average number of AIO requests in the queues along with the maxg as it shows the maximum number of AIO requests in the

queues for the last measuring period. If the avgc or maxg is getting close to the maxr then tuning of the maxreqs and maxservers attributes is required.

If we use raw devices, then avfc is of interest as it reflects the use of the average AIO fastpath calls along with the maxf as it shows the maximum value of fastpath count value. As the fast path calls bypass the AIO queues, these statistics give only information on how fastpath AIO used.

If we are interested in the allocation of the AIO queues and their use, then the **iostat -Aq** command is useful (see Example 7-6).

Example 7-6 Output of the iostat -Aq command with AIO queue statistics

```
[p630n06][/]> iostat -Aq | head

System configuration: lcpu=4

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow
        0    0    0    0 4096            3.0  1.0  95.6  0.4

q[  0]=0       q[  1]=0       q[  2]=0       q[  3]=0       q[  4]=0
q[  5]=0       q[  6]=0       q[  7]=0       q[  8]=0       q[  9]=0
q[ 10]=0       q[ 11]=0       q[ 12]=0       q[ 13]=0       q[ 14]=0
q[ 15]=0       q[ 16]=0       q[ 17]=0       q[ 18]=0       q[ 19]=0
```

If statistics of any single queue are significantly higher than the others, it points to the application using one file system significantly more than other file systems. The queues are usually allocated one per file system.

If a specific AIO queue is filling up unusually and we want to know to which file system the queue is related, then the **iostat -AQ** command is useful. See Example 7-7 that shows the distribution of the AIO queues to specific file systems.

Example 7-7 The iostat -AQ command output

```
[p630n06][/]> iostat -AQ

System configuration: lcpu=4

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow
        0    0    0    0 4096            3.0  1.0  95.6  0.4

Queue#         Count          Filesystems
129            0              /
130            0              /usr
132            0              /var
133            0              /tmp
```

```
136             0               /home
137             0               /proc
138             0               /opt
139             0               /itso_files
143             0               /mnt
156             0               /localfs
158             0               /ljfs
177             0               /essfs
```

The `iostat -P` *option and related options for POSIX AIO*

In order to get the basic POSIX AIO statistics we have to use the iostat -P command. The output is in the same format as for the iostat -A legacy AIO command as well as the meaning of the flags is the same but related to the POSIX AIO calls.

The output format of the **`iostat -Pq`** command corresponds with the **`iostat -Aq`** and the **`iostat -PQ`** corresponds with the **`iostat -AQ`** command.

Using SMIT and going through menus Devices → Asynchronous I/O → Asynchronous I/O (Legacy) → Change/Show Characteristics of Asynchronous I/O, you can set the characteristics of the AIO, like minservers, maxservers, maxreqs, kprocprio, autoconfig, or fastpath.

7.2.2 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.

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. Since **`filemon`** is based on the trace utility, it is normally only used for reporting over a period of a few minutes.

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

filemon syntax

```
filemon [ -d ] [ -i Trace_File -n Gennames_File] [ -o File] [ -O Levels] [ -P ]
[ -T n] [ -u ] [ -v ]
```

Useful combinations

- **`filemon -o fm.out;sleep 15;trcstop`** filemon trace for 15 seconds, output saved to fm.out
- **`filemon; {disk I/O command}; trcstop`** filemon trace for of 'some command', output to stdout

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.

Tip: To facilitate using **filemon**, a simple shell script can be created. We created a script called fmon.sh.

```
[p630n06][/home/guest]> cat fmon.sh
#!/usr/bin/ksh
Usage="Usage: fmon.sh [command]"
if [ $# -eq 0 ] ; then
        echo $Usage
        exit
fi
filemon -o fm.out -O all
$@
#next line optional
#sync;sleep 5
trcstop
```

Examples

The **dd** command provides an easy way to generate I/O to disk which can be traced with the **filemon** command.

Writing to a file

The most active resources are reported with filemon. If the item you are interested in is not one of the most active, the report may not contain information on that item.

In this example we use dd to write a 100 MB file to disk using a 4MB blocksize. In Example 7-8 the output is saved to the fm.out.

Example 7-8 The **filemon** *trace while writing to a file with* **dd**

```
[p630n06][/home/guest]> filemon -o fm.out -O all

Enter the "trcstop" command to complete filemon processing

[p630n06][/home/guest]> dd if=/dev/zero of=/localfs/100mbfile bs=4096K count=25
25+0 records in.
25+0 records out.
[p630n06][/home/guest]> trcstop
[p630n06][/home/guest]> [filemon: Reporting started]
[filemon: Reporting completed]

[filemon: 5.664 secs in measured interval]
```

Because the '-O all' flag was specified, all the filemon reports were saved in fm.out. If the -O flag is not specified, the default is the vm, lv, and pv levels (virtual memory, logical volume, physical volume). The only level not included by default

is the logical file level (lf). All reports, regardless of which report flag(s) are specified contains header information similar to what is show in Example 7-9.

Example 7-9 Header information for filemon reports

```
Fri Oct  8 14:06:37 2004
System: AIX p630n06 Node: 5 Machine: 000684FF4C00

Cpu utilization:  31.9%
```

The header shows when and where the report was created and the CPU utilization during the monitoring period.

We will now cover each report in detail.

Logical volumes report

The logical volume report has three parts; the header, the logical volume summary, and the detailed logical volume report. To create only a logical volume report, issue the **filemon** command as follows:

```
filemon -uo filemon.lv -O lv;{command or sleep statement};trcstop
```

Example 7-10 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 7-10 Logical volume report information

```
Fri Oct 15 12:24:22 2004
System: AIX p630n06 Node: 5 Machine: 000684FF4C00

Cpu utilization:  32.2%

Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume                  description
------------------------------------------------------------------------
  0.45     56 204800 23128.4  /dev/fslv01              /localfs
  0.41   1952      0  220.4  /dev/hd6                 paging
  0.39   1936      0  218.6  /dev/paging00            paging
  0.14      0    280   31.6  /dev/hd8                 jfs2log
  0.01      0     16    1.8  /dev/hd4                 /
  0.00      0      8    0.9  /dev/hd1                 /home
  0.00      0     16    1.8  /dev/loglv02             jfs2log

------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/fslv01  description: /localfs
```

```
reads:                  7       (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.247 min   0.231 max   0.298 sdev   0.021
  read sequences:       4
  read seq. lengths:    avg    14.0 min       8 max      32 sdev    10.4
writes:                 400     (0 errs)
  write sizes (blks):   avg   512.0 min     512 max     512 sdev     0.0
  write times (msec):   avg 287.052 min  19.938 max 867.239 sdev 244.275
  write sequences:      1
  write seq. lengths:   avg 204800.0 min  204800 max  204800 sdev     0.0
seeks:                  5       (1.2%)
  seek dist (blks):     init   1536,
                        avg  2366.0 min      40 max    8032 sdev  3313.3
time to next req(msec): avg   8.674 min   0.000 max 975.635 sdev  74.663
throughput:             23128.4 KB/sec
utilization:            0.45
...lines omitted...
```

The 100 MB of data written to `fslv01` with the **dd** command is visible in the report. In the `Most Active Logical Volumes` section, the `wblk` column is the number of 512 byte write blocks, so 204800 * 512 bytes = 104857600 bytes = 100*1024*1024 bytes = 100 MB. In the `Detailed Logical Volume Stats` section, a similar calculation is possible: 400 writes * 512 write size * 512 byte block size = 400 * 512 * 512 bytes = 104857600 bytes = 100*1024*1024 bytes = 100 MB.

Virtual memory system report

The virtual memory report has three parts: the header, the segment summary, and the detailed segment report. To create only a virtual memory report, issue the **filemon** command as follows:

```
filemon -uo filemon.vm -O vm;{command or sleep statement};trcstop
```

Example 7-11 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 7-11 Virtual memory system report information

```
Fri Oct 15 12:24:19 2004
System: AIX p630n06 Node: 5 Machine: 000684FF4C00

Cpu utilization:  33.2%

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype                  volume:inode
------------------------------------------------------------------------
 100.0      0  25600   f3a1f  page table
   0.6    154      0   e001  page table
```

```
   0.0      0      1   f7e1  page table
   0.0      0      1   870d0  page table
   0.0      0      1   5f0cb  page table

------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: f3a1f  segtype: page table
segment flags:          pgtbl
writes:                 25600   (0 errs)
  write times (msec):   avg 801.602 min  30.606 max 1559.925 sdev 444.427
  write sequences:      1
  write seq. lengths:   avg 25600.0 min   25600 max   25600 sdev     0.0

SEGMENT: e001  segtype: page table
segment flags:          log pgtbl
reads:                  154     (0 errs)
  read times (msec):    avg   3.824 min   0.198 max  19.220 sdev   4.905
  read sequences:       1
  read seq. lengths:    avg   154.0 min     154 max     154 sdev     0.0
...(lines omitted)...
```

The 100 MB of data written with the **dd** command passed through the VMM and is visible in the report. In the `Most Active Segments` section, the `100MB` is visible in the first line. The `segid` of that line `f3alf` can be matched with the `Detailed Logical Volume Stats` section. For that segid, 25600 writes of blocksize 4096 bytes are shown. 25600 * 4096 bytes = 104857600 bytes = 100*1024*1024 bytes = 100 MB.

Logical file system report

The logical file system report has three parts: the header, most active files, and the detailed file stats. To create only a logical file system report, issue the **filemon** command as follows:

```
filemon -uo filemon.lf -O lf;{command or sleep statement};trcstop
```

Example 7-11 on page 445 shows the full logical file system report. In the report, the file with the highest utilization is in the beginning.

Example 7-12 Logical file system report information

```
Fri Oct 15 12:24:15 2004
System: AIX p630n06 Node: 5 Machine: 000684FF4C00

Cpu utilization:  30.2%

Most Active Files
```

```
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                      volume:inode
------------------------------------------------------------------------
 100.0      1      0     25  100mbfile                 <major=0,minor=2>:9
 100.0      1     25      0  zero
   0.3      1     70      0  unix                      <major=0,minor=5>:4433
   0.0      3      6      2  ksh.cat                   <major=0,minor=5>:15753
   0.0      1      2      0  cmdtrace.cat              <major=0,minor=5>:15469
   0.0      1      1      0  dd.cat                    <major=0,minor=5>:15500
   0.0      7      2      0  SWservAt                  <major=0,minor=4>:53
   0.0      7      2      0  SWservAt.vc               <major=0,minor=4>:54

------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------

FILE: /localfs/100mbfile  volume: <major=0,minor=2>  inode: 9
opens:                  1
total bytes xfrd:       104857600
writes:                 25      (0 errs)
  write sizes (bytes):  avg 4194304.0 min 4194304 max 4194304 sdev     0.0
  write times (msec):   avg  14.492 min  13.079 max  16.161 sdev   0.673
lseeks:                 2

FILE: /dev/zero
opens:                  1
total bytes xfrd:       104857600
reads:                  25      (0 errs)
  read sizes (bytes):   avg 4194304.0 min 4194304 max 4194304 sdev     0.0
  read times (msec):    avg   6.010 min   4.630 max  10.667 sdev   1.688
...(lines omitted)...
```

The 100 MB file created with the **dd** command, using /dev/zero as the source is visible throughout this report. The `Most Active Files` section shows both the amount of data written as well as the number of write commands issued. The count=25 parameter of the **dd** command matches with the number of writes. In the `Detailed File Stats` section, we see the 4096 KB blocksize specified in the dd command matching the write sizes in bytes (4194304).

Physical volumes report

The physical volume report is divided into three parts; the header, the physical volume summary, and the detailed physical volume report. To create only a physical volume report, issue the **filemon** command as follows:

```
filemon -uo filemon.pv -O pv;{command or sleep statement};trcstop
```

Example 7-13 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 7-13 Physical volumes report information

```
Fri Oct 15 12:24:27 2004
System: AIX p630n06 Node: 5 Machine: 000684FF4C00

Cpu utilization:  31.7%

Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume                   description
------------------------------------------------------------------------
  0.66     56 204815 34935.7  /dev/hdisk2               N/A
  0.57   1976    424  409.3  /dev/hdisk0              N/A
  0.02      0     31    5.3  /dev/hdisk1              N/A

------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hdisk2  description: N/A
reads:                  7       (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.229 min   0.217 max   0.246 sdev   0.008
  read sequences:       4
  read seq. lengths:    avg    14.0 min       8 max      32 sdev    10.4
writes:                 416     (0 errs)
  write sizes (blks):   avg   492.3 min       1 max     512 sdev    96.7
  write times (msec):   avg   4.534 min   1.960 max  11.742 sdev   1.161
  write sequences:      23
  write seq. lengths:   avg  8905.0 min       1 max   31744 sdev 13550.0
seeks:                  27      (6.4%)
  seek dist (blks):     init 57677568,
                        avg 35549677.2 min       1 max 57875454 sdev 28104423.6
  seek dist (%tot blks):init 20.11437,
                        avg 12.39753 min 0.00000 max 20.18339 sdev 9.80109
time to next req(msec): avg  75.037 min   0.010 max 958.979 sdev 176.740
throughput:             34935.7 KB/sec
utilization:            0.66
...(lines omitted)...
```

The 100 MB file created with the **dd** command that was written to hdisk2 is visible throughout this report. The `Most Active Physical Volumes` section shows the number of 512 byte blocks written to the volume. 204815 * 512 = 100 MB. The `Detailed Physical Volume Stats section` shows 416 writes with an average write

size of 492 blocks. Besides the write commands caused by the **dd** command, some other minor activity occurred during the filemon tracing.

7.2.3 The fileplace command

The **fileplace** command displays the placement of a file's logical or physical blocks within a Journaled File System (JFS) or Enhanced Journaled File System (JFS2), but not Network File System (NFS). 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 fileset, which is installable from the AIX base installation media.

fileplace syntax

```
fileplace [ { -l | -p } [ -i ] [ -v ] ] File | [-m LogicalVolumeName]
```

Useful combinations

- **fileplace -lv somefile** Logical layout for file 'somefile'
- **fileplace -pv otherfile** Physical layout for file 'otherfile'

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 non-contiguous 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 file systems 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).

Examples for fileplace

In Example 7-14, 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 7-14 Using fileplace

```
[p630n06][/]> fileplace /smit.log

File: /smit.log  Size: 55920 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 14

  Logical Extent
  --------------
  00000782                      1 frags         4096 Bytes,   7.1%
  00000787-00000788             2 frags         8192 Bytes,  14.3%
  00000856-00000860             5 frags        20480 Bytes,  35.7%
  00000875                      1 frags         4096 Bytes,   7.1%
  00000880-00000881             2 frags         8192 Bytes,  14.3%
  00000893-00000895             3 frags        12288 Bytes,  21.4%
```

The report shows that the majority of the file occupies a consecutive range of blocks starting from 856 and ending at 860(35.7%).

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 7-15.

Example 7-15 Using fileplace -l

```
[p630n06][/]> fileplace -l /smit.log

File: /smit.log  Size: 55920 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 14

  Logical Extent
  --------------
  00000782                    1 frags         4096 Bytes,   7.1%
  00000787-00000788           2 frags         8192 Bytes,  14.3%
  00000856-00000860           5 frags        20480 Bytes,  35.7%
  00000875                    1 frags         4096 Bytes,   7.1%
  00000880-00000881           2 frags         8192 Bytes,  14.3%
  00000893-00000895           3 frags        12288 Bytes,  21.4%
```

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.

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 7-16.

Example 7-16 Using fileplace -p

```
[p630n06][/]> fileplace -p /smit.log

File: /smit.log  Size: 55920 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 14

  Physical Addresses (mirror copy 1)                                  Logical Extent
  ---------------------------------- ----------------
  14419246           hdisk0            1 frags         4096 Bytes,   7.1%    00000782
  14419251-14419252  hdisk0            2 frags         8192 Bytes,  14.3%    00000787-00000788
  14419320-14419324  hdisk0            5 frags        20480 Bytes,  35.7%    00000856-00000860
  14419339           hdisk0            1 frags         4096 Bytes,   7.1%    00000875
  14419344-14419345  hdisk0            2 frags         8192 Bytes,  14.3%    00000880-00000881
  14419357-14419359  hdisk0            3 frags        12288 Bytes,  21.4%    00000893-00000895
```

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 from left to right as:

`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.

Analyzing the physical address

The Logical Volume Device Driver (LVDD) requires that all disks are partitioned in 512 byte 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 5.3 Kernel Extensions and Device Support Programming Concepts*, SC23-4900, 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 5.3 Files Reference*, SC23-4895, 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 5.3 Performance Management Guide*, SC23-4905.

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 7-17 illustrates how the 512-byte physical disk block is reported by the **fileplace** command.

Example 7-17 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 5.3 System Management Concepts: Operating System and Devices*, SC23-4908.

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. The -i flag is not support with JFS2 filesystems. For more detail on the use of the indirect block see *AIX 5L Version 5.3 System User's Guide: Operating System and Devices,* SC23-4910. For examples of the use of the -i flag, see *AIX 5L Performance Tools Handbook*, SG24-6039.

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 7-18 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 7-18 Using fileplace -vl

```
[p630n06][/]> fileplace -vl /smit.log

File: /smit.log  Size: 62087 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 16
Inode: 6  Mode: -rw-r--r--  Owner: root  Group: system

  Logical Extent
  --------------
  00000782                      1 frags         4096 Bytes,   6.2%
  00000787-00000788             2 frags         8192 Bytes,  12.5%
  00000856-00000860             5 frags        20480 Bytes,  31.2%
  00000875                      1 frags         4096 Bytes,   6.2%
  00000880-00000881             2 frags         8192 Bytes,  12.5%
  00000893-00000895             3 frags        12288 Bytes,  18.8%
  00000940-00000941             2 frags         8192 Bytes,  12.5%

  16 frags over space of 160 frags:   space efficiency = 10.0%
  7 extents out of 16 possible:   sequentiality = 60.0%
```

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. 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 10 percent space efficiency for the logical fragmentation.

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

Physical report

In Example 7-19 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 7-19 The fileplace -vp

```
[p630n06][/]> fileplace -vp /smit.log

File: /smit.log  Size: 67909 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 17
Inode: 6  Mode: -rw-r--r--  Owner: root  Group: system
```

```
Physical Addresses (mirror copy 1)                                   Logical Extent
----------------------------------                                   ----------------
14419246           hdisk0            1 frags         4096 Bytes,   5.9%    00000782
14419251-14419252  hdisk0            2 frags         8192 Bytes,  11.8%    00000787-00000788
14419320-14419324  hdisk0            5 frags        20480 Bytes,  29.4%    00000856-00000860
14419339           hdisk0            1 frags         4096 Bytes,   5.9%    00000875
14419344-14419345  hdisk0            2 frags         8192 Bytes,  11.8%    00000880-00000881
14419357-14419359  hdisk0            3 frags        12288 Bytes,  17.6%    00000893-00000895
14419404-14419405  hdisk0            2 frags         8192 Bytes,  11.8%    00000940-00000941
14419416           hdisk0            1 frags         4096 Bytes,   5.9%    00000952

17 frags over space of 171 frags:   space efficiency = 9.9%
8 extents out of 17 possible:   sequentiality = 56.2%
```

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 56.2 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

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. 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 7-20 shows how to use the **ls**, **du**, and **fileplace** commands to identify that a file is not sparse.

Example 7-20 Checking a file that is not sparse

```
[p630n06][/localfs]> ls -l happy.file
-rw-r--r--   1 root     system         1536 Oct 12 14:38 happy.file
[p630n06][/localfs]> du -k happy.file
4       happy.file
[p630n06][/localfs]> fileplace happy.file

File: happy.file  Size: 1536 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1

  Logical Extent
  --------------
  00000194                      1 frags         4096 Bytes,  100.0%
```

The example output above shows that the size of the file happy.file is 1536 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 7-21 shows how the same type of report could look if the file was sparse.

Example 7-21 Checking a sparse file

```
[p630n06][/localfs]> ls -l unhappy.file
-rw-r--r--   1 root     system       256001 Oct 12 14:35 unhappy.file
[p630n06][/localfs]> du -k unhappy.file
4       unhappy.file
[p630n06][/localfs]> fileplace unhappy.file

File: unhappy.file  Size: 256001 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1

  Logical Extent
  --------------
  00000193                      1 frags         4096 Bytes,  100.0%
        unallocated            62 frags       253952 Bytes      0.0%
```

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 (256001). 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 62 unallocated blocks and one allocated at logical address 00000193, 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 (12) from the **df** output in Example 7-22.

Example 7-22 Using df

```
[p630n06][/localfs]> df $PWD
Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
/dev/fslv01     10485760   1978128   82%       12     1% /localfs
```

Then we use the **dd** command to create a 1 gigabyte sparse file as shown in the Example 7-23. The input was just a new line character (\n) from the echo command.

Example 7-23 Creating a sparse file

```
[p630n06][/localfs]> echo | dd of=ugly.file seek=1024 bs=1024k
0+1 records in.
0+1 records out.
```

Example 7-24 shows the examination of the file's space utilization with the **ls**. The example 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 7-24 Using ls on the sparse file

```
[p630n06][/localfs]> ls -sl ugly.file
   4 -rw-r--r--   1 root     system   1073741825 Oct 12 15:24 ugly.file
```

According to the **ls** output in the previous example, the file size is 1073741825 bytes but only 4 (1 KB) blocks. Now we know that this is a sparse file. In Example 7-25 on page 460 we use the **fileplace -l** command to look at the allocation in detail, first from a logical view.

Example 7-25 Using fileplace -l on the sparse file

```
[p630n06][/localfs]> fileplace -l ugly.file

File: ugly.file  Size: 1073741825 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1

  Logical Extent
  --------------
  00000195                      1 frags         4096 Bytes,  100.0%
        unallocated        262144 frags   1073741824 Bytes      0.0%
```

The logical report above shows that logical block 195 is allocated for the file occupying 4 KB, and the rest is unallocated.

Example 7-26 shows the physical view of the file using the **fileplace -p** command.

Example 7-26 Using fileplace -p on the sparse file

```
[[p630n06][/localfs]> fileplace -p ugly.file

File: ugly.file  Size: 1073741825 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1

  Physical Addresses (mirror copy 1)                                  Logical Extent
  ----------------------------------                                  ----------------
   07209699         hdisk2              1 frags         4096 Bytes,  100.0%    00000195
   unallocated                     262144 frags   1073741824 Bytes     0.0%
```

The physical report shows that physical block 07209699 is allocated for the logical block 195 and it resides on hdisk2.

Example 7-27 shows the volume report (logical view) for the file using the **fileplace -v** command.

Example 7-27 Using fileplace -lv on the sparse file

```
[p630n06][/localfs]> fileplace -lv ugly.file

File: ugly.file  Size: 1073741825 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1
Inode: 12  Mode: -rw-r--r--  Owner: root  Group: system

  Logical Extent
  --------------
  00000195                      1 frags         4096 Bytes,  100.0%
        unallocated        262144 frags   1073741824 Bytes      0.0%
```

```
  1 frags over space of 1 frags:   space efficiency = 100.0%
  1 extent 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 7-28) shows the volume report for the physical view of the file.

Example 7-28 Using fileplace -pv on the sparse file

```
#[p630n06][/localfs]> fileplace -pv ugly.file

File: ugly.file  Size: 1073741825 bytes  Vol: /dev/fslv01
Blk Size: 4096  Frag Size: 4096  Nfrags: 1
Inode: 12  Mode: -rw-r--r--  Owner: root  Group: system

  Physical Addresses (mirror copy 1)                                  Logical Extent
  ----------------------------------                                  ----------------
  07209699           hdisk2             1 frags         4096 Bytes,  100.0%    00000195
        unallocated                 262144 frags   1073741824 Bytes    0.0%

  1 frags over space of 1 frags:   space efficiency = 100.0%
  1 extent 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.

Searching for sparse files

To find sparse files in file systems we can use the **find** command with the -ls flag. Example 7-29 shows how this can be done.

Example 7-29 Using find to find sparse files

```
[p630n06][/localfs]> find /localfs -type f -xdev -ls
    9 102400 -rw-r--r--  1 root     system    104857600 Oct 11 15:14 /localfs/100mbfile
   11      4 -rw-r--r--  1 root     system        1536 Oct 12 14:38 /localfs/happy.file
   12      4 -rw-r--r--  1 root     system     1073741825 Oct 12 15:24 /localfs/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 7-30 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 7-30 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 7-31 is sample output from running the script above.

Example 7-31 Sample output from sparse file search script

```
/localfs/ugly.file 4 1073741825
/ljfs/ug3 16 1073741825
```

To find out how many sparse files the script found, just pipe the output to the **wc** command with -l flag.

Displaying logical to physical map for a logical volume

Example 7-32 shows the use of -m flag to display logical to physical map of a logical volume.

Example 7-32 Using -m flag

```
[p630n06][/]> fileplace -m hd2
Device: /dev/hd2   Partition Size: 256 MB      Block Size = 4096
Number of Partitions: 6   Number of Copies: 1

  Physical Addresses (mirror copy 1)                                 Logical Fragment
  ----------------------------------                                 ----------------
  14484000-14549535  hdisk0        65536 blocks 268435456 Bytes,  16.7%    0000000-0065535
  14811680-15139359  hdisk0       327680 blocks 1342177280 Bytes,  83.3%    0065536-0393215
```

7.2.4 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 resides in /usr/sbin and are part of the bos.rte.lvm fileset, which is installed by default from the AIX base installation media.

lslv

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

```
lslv [ -L ] [ -l| -m ] [ -n DescriptorPV ] LVname
lslv [ -L ] [ -n DescriptorPV ] -p PVname [ LVname ]
```

Useful combinations

- **lslv LVname** — List detailed logical volume attributes for logical volume
- **lslv -l LVname** — Lists physical volumes and distributions for logical volume

lspv

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

```
lspv [-L] [-M | -l | -p]  [-n DescriptorPV] [-v VGid] PVname
```

Useful combinations

- **lspv** — List all physical volumes and the associated volume groups
- **lspv -l PVname** — Lists logical volumes on the specified physical volume

lsvg

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

```
lsvg [-o] [[-L] -n PVname] | -p ] volume group ...
lsvg [-L] [-i] [-M | -l | -p] VGname...
```

Useful combinations

- **lsvg** List all volume groups
- **lsvg VGname** List detailed volume group attributes
- **lsvg -l VGname** Lists logical volumes for a volume group
- **lsvg -p VGname** Lists physical volumes for a volume group

Examples for lslv, 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, which shows the load on different disks in great detail. The output in Example 7-33 is the summary since boot time (if the **iostat** attribute has been enabled for the sys0 logical device driver).

Example 7-33 Starting point with iostat

```
[p630n06][/]> iostat -ad

System configuration: lcpu=4 drives=7

Adapter:                 Kbps      tps    Kb_read   Kb_wrtn
scsi0                   22.8       1.2    1950889   7790812

Paths/Disk:      % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk3_Path0          0.0      0.0      0.0       3157         0
hdisk2_Path0          0.3     18.5      0.5    1745814   6147340
hdisk1_Path0          0.2      3.3      0.5      55979   1348248
hdisk0_Path0          0.1      1.0      0.2     145939    295224

Adapter:                 Kbps      tps    Kb_read   Kb_wrtn
ide0                     0.0       0.0          0         0
```

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

Adapter:                   Kbps      tps    Kb_read   Kb_wrtn
fcs0                      76.7      5.9      22728  32770316

Paths/Disk:       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk5_Path1         0.0      0.0      0.0          0         0
hdisk5_Path0         0.2     64.6      2.9        598  27608970
hdisk4_Path1         0.0      0.0      0.0          0         0
hdisk4_Path0         0.1     12.1      3.0      22130   5161346
```

This system has three adapters (SCSI, IDE, Fibre). There are four local disks on the SCSI adapter. The IDE adapter is controlling the CD-ROM. The fibre channel adapter has two paths to hdisk4 and hdisk5. 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 7-34.

Example 7-34 Using uptime

```
[p630n06][/]> uptime
  04:31PM   up 4 days,  22:49,  10 users,  load average: 0.00, 0.03, 0.04
```

The example tells us that the statistics have been collected over four 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 7-35.

Example 7-35 Using iostat

```
# iostat -ad 1 2
...(lines omitted)...
Adapter:                   Kbps      tps    Kb_read   Kb_wrtn
scsi0                    6764.0     278.0          0      6764

Paths/Disk:       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk3_Path0         0.0      0.0      0.0          0         0
hdisk2_Path0        51.0   4236.0     57.0          0      4236
hdisk1_Path0        97.0   2528.0    221.0          0      2528
hdisk0_Path0         0.0      0.0      0.0          0         0

Adapter:                   Kbps      tps    Kb_read   Kb_wrtn
fcs0                     8832.0     69.0          0      8832

Paths/Disk:       % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk5_Path1         0.0      0.0      0.0          0         0
hdisk5_Path0        18.0   8832.0     69.0          0      8832
```

```
hdisk4_Path1          0.0       0.0       0.0          0         0
hdisk4_Path0          0.0       0.0       0.0          0         0
```

And now we see that the system performs quite a bit of I/O on `hdisk1 and hdisk2`, so we should check how the layout is for these disks. First let's find out what volume groups the disk belong to as seen in Example 7-36.

Example 7-36 Using lspv to examine the disk versus volume group mapping

```
[p630n06][/]> lspv
hdisk0          000684ff60a73b09                    rootvg          active
hdisk1          000684ff753af8b0                    localvg         active
hdisk2          000684fff5a4045d                    localvg         active
hdisk3          000684ff702c2c01                    jikkvg          active
hdisk4          0000331209edf8b2                    essvg           active
hdisk5          0000331209edf958                    essvg           active
```

The disks we are examining (`hdisk1 and hdisk2`) belong to the `localvg` 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 7-37.

Example 7-37 Using lsvg to check the distribution

```
[p630n06][/]> lsvg -p localvg
localvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk1            active            546         420         110..00..92..109..109
hdisk2            active            546         426         110..00..98..109..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 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 7-38.

Example 7-38 Using lsvg to get all logical volumes within the volume group

```
[p630n06][/]> lsvg -l localvg
localvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv02             jfs2log    1     1     1    open/syncd    N/A
fslv01              jfs2       20    40    2    open/stale    /localfs
loglv03             jfslog     1     1     1    open/syncd    N/A
lv00                jfs        4     4     1    open/syncd    /ljfs
testlv              jfs2       200   200   2    closed/syncd  N/A
```

This tells us that there are both JFS and JFS2 filesystems, a logical volume without an entry in /etc/filesystems (testlv mount point show up as N/A), and that one logical volume is mirrored (fslv01) and one logical volume is spread over two disks (testlv). The output above also shows that we have two external log logical volumes; loglv02 that is used by JFS2 file systems and loglv03 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. Opened/stale indicates that the logical volume is open but contains partitions that are not current. Opened/syncd indicates that the logical volume is open and synchronized. Closed 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** or **lvmstat** commands. For instance, Example 7-39 with **lvmstat** shows the five busiest logical volumes.

Example 7-39 Checking busy logical volumes with lvmstat

```
[p630n06][/]> lvmstat -v localvg -c 5

Logical Volume       iocnt   Kb_read   Kb_wrtn      Kbps
  fslv01             19036   3380224    709512     19.07
  lv00               13574   1048644     95280      5.33
```

```
  testlv              6188   1584128          0         7.39
  loglv03             3422         0      13688         0.06
  loglv02             1690         0       6760         0.03
```

We can clearly see that both `fslv01`and `lv00` 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 7-40.

Example 7-40 Using lslv -p

```
[p630n06][/]> lslv -p hdisk2 fslv01
hdisk2:fslv01:/localfs
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE      1-10
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     11-20
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     21-30
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     31-40
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     41-50
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     51-60
FREE   FREE   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    101-110
0001   0002   0003   0004   0005   0006   0007   0008   0009   0010    111-120
0011   0012   0013   0014   0015   0016   0017   0018   0019   0020    121-130
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    131-140
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    141-150
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    151-160
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    161-170
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    171-180
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    181-190
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    191-200
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    201-210
USED   USED   USED   USED   USED   USED   USED   USED   USED           211-219
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    220-229
USED   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    230-239
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    240-249
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    250-259

...(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 0001-0020 indicate that this belongs to the logical volume we wanted to check, in our case fslv01. A STALE partition (not shown in the example above) is a physical partition that contains data you cannot use.

Example 7-41 shows a similar output from **lspv** to find out the intra disk layout of logical volumes on hdisk1 and hdisk2.

Example 7-41 Using lspv to check the intra disk policy

```
[p630n06][/]> lspv -l hdisk1;lspv -l hdisk2
hdisk1:
LV NAME               LPs   PPs   DISTRIBUTION          MOUNT POINT
testlv                100   100   55..45..00..00..00    N/A
lv00                  4     4     04..00..00..00..00    /ljfs
loglv03               1     1     01..00..00..00..00    N/A
fslv01                20    20    20..00..00..00..00    /localfs
loglv02               1     1     01..00..00..00..00    N/A
hdisk2:
LV NAME               LPs   PPs   DISTRIBUTION          MOUNT POINT
testlv                100   100   55..45..00..00..00    N/A
fslv01                20    20    20..00..00..00..00    /localfs
```

Filesystems lv00 and fslv01 are both on hdisk1. Additionally fslv01 is mirrored on hdisk2. The filesystems are on the same part of hdisk1, and is contiguously allocated there. Example 7-42 shows the intra disk layout in another, more readable, way with the **lspv** command.

Example 7-42 Using lspv to check the intra disk layout

```
[p630n06][/]> lspv -p hdisk1;lspv -p hdisk2
hdisk1:
PP RANGE  STATE   REGION        LV ID              TYPE       MOUNT POINT
  1-110   free    outer edge
111-111   used    outer middle  loglv02            jfs2log    N/A
112-112   used    outer middle  loglv03            jfslog     N/A
113-116   used    outer middle  lv00               jfs        /ljfs
117-136   stale   outer middle  fslv01             jfs2       /localfs
137-219   used    outer middle  testlv             jfs2       N/A
220-236   used    center        testlv             jfs2       N/A
237-328   free    center
329-437   free    inner middle
438-546   free    inner edge
hdisk2:
PP RANGE  STATE   REGION        LV ID              TYPE       MOUNT POINT
  1-110   free    outer edge
111-130   used    outer middle  fslv01             jfs2       /localfs
```

```
131-219   used    outer middle  testlv              jfs2       N/A
220-230   used    center        testlv              jfs2       N/A
231-328   free    center
329-437   free    inner middle
438-546   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 7-43 shows a quick look at the two log logical volumes.

Example 7-43 Using lslv to check the logical volume disk layout

```
[p630n06][/]> lslv -m fslv01;lslv -m lv00
fslv01:/localfs
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0111 hdisk2            0117 hdisk1
0002  0112 hdisk2            0118 hdisk1
...(lines omitted)...
0019  0129 hdisk2            0135 hdisk1
0020  0130 hdisk2            0136 hdisk1
lv00:/ljfs
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0113 hdisk1
0002  0114 hdisk1
0003  0115 hdisk1
0004  0116 hdisk1
```

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.

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.

Using lslv

The **lslv** command displays the characteristics and status of the logical volume, as Example 7-44 shows.

Example 7-44 Logical volume fragmentation with lslv

```
[p630n06][/]> lslv -l hd6
hd6:N/A
PV                COPIES        IN BAND       DISTRIBUTION
hdisk0            010:000:000   100%          000:010:000:000:000
```

The **lslv** command also shows that it has 10 LPs but no additional copies. It also says that the intra-policy of center is 100% in band.

The **lslv -l** report has the following format:

PV Physical volume name.

COPIES These three fields are displayed:

- 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.

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 7-45 shows.

Example 7-45 Logical volume fragmentation with lspv -l

```
[p630n06][/]> lspv -l hdisk0
hdisk0:
LV NAME               LPs   PPs   DISTRIBUTION          MOUNT POINT
hd6                   10    10 00..10..00..00..00    N/A
...(lines omitted)...
```

This example shows that the hd6 logical volume is at the outer middle part of the disk, with all physical partitions located their.

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 7-46.

Example 7-46 Logical volume fragmentation with lspv -p

```
[p630n06][/]> lspv -p hdisk0
hdisk0:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-1     used    outer edge    hd5                 boot       N/A
  2-110   free    outer edge
111-111   used    outer middle  hd6                 paging     N/A
112-115   used    outer middle  lg_dumplv           sysdump    N/A
116-124   used    outer middle  hd6                 paging     N/A
125-219   free    outer middle
220-220   used    center        hd8                 jfs2log    N/A
221-221   used    center        hd4                 jfs2       /
222-222   used    center        hd2                 jfs2       /usr
223-223   used    center        hd9var              jfs2       /var
224-224   used    center        hd3                 jfs2       /tmp
225-225   used    center        hd1                 jfs2       /home
```

```
226-226   used    center        hd10opt            jfs2       /opt
227-231   used    center        hd2                jfs2       /usr
232-241   used    center        paging00           paging     N/A
242-328   free    center
329-437   free    inner middle
438-546   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.

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 7-47 shows how to obtain this basic information about a volume group.

Example 7-47 Using lsvg to obtain volume group basics

```
[p630n06][/]> lsvg -L essvg
VOLUME GROUP:       essvg                   VG IDENTIFIER:  000684ff00004c00000000ff78cdb1e1
VG STATE:           active                  PP SIZE:        16 megabyte(s)
VG PERMISSION:      read/write              TOTAL PPs:      1190 (19040 megabytes)
MAX LVs:            256                     FREE PPs:       869 (13904 megabytes)
LVs:                2                       USED PPs:       321 (5136 megabytes)
OPEN LVs:           2                       QUORUM:         2
TOTAL PVs:          2                       VG DESCRIPTORS: 3
STALE PVs:          0                       STALE PPs:      0
```

```
ACTIVE PVs:         2                       AUTO ON:        yes
MAX PPs per VG:     32512
MAX PPs per PV:     1016                    MAX PVs:        32
LTG size (Dynamic): 256 kilobyte(s)         AUTO SYNC:      no
HOT SPARE:          no                      BB POLICY:      relocatable
```

The volume group shown in the example has two logical volumes and two disks with a physical partition size of 16 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 7-48. 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 7-48 Using lsvg to check the logical volume state

```
[p630n06][/]> lsvg -l essvg
essvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv01             jfs2log    1     1     1    open/syncd    N/A
fslv02              jfs2       320   320   1    open/syncd    /essfs
```

As the example above shows, there are one logical volume with a file system and one jfslog2. We can have two types of jfs: a journal file system or an Enhanced Journaled File System (JFS2).

Remember that the physical partition size was 16 MB, so even though the logs logical volume only has one (1) logical partition it is a 16 MB partition. Example 7-49 shows the disks that are allocated for this volume group.

Example 7-49 Using lsvg to determine disks allocated to the volume group

```
[p630n06][/]> lsvg -p essvg
essvg:
PV_NAME           PV STATE          TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk4            active            595         594         119..118..119..119..119
hdisk5            active            595         275         119..00..00..37..119
```

So there are two disks 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.

Acquiring more disk information

Example 7-50 uses the **lsdev** command to obtain information about the types of disks in the volume group.

Example 7-50 Using lsdev to examine a disk device

```
[p630n06][/]> lsdev -Cl hdisk4
hdisk4 Available 1n-08-02 MPIO Other FC SCSI Disk Drive
```

The output tells us that it is an MPIO FC SCSI disk drive.

7.2.5 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.

lvmstat syntax

```
lvmstat {-l|-v} <name> [-e|-d] [-F] [-C] [-c count] [-s] [interval [iterations]]
```

Useful combinations

- **lvmstat -v rootvg -e** Enable stat collection for volume group rootvg
- **lvmstat -v rootvg** Report stats for volume group rootvg
- **lvmstat -v rootvg -d** Disable stat collection for volume group rootvg

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 (LVMDD) 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.

Examples for lvmstat

If the statistics collection has not been enabled for the volume group or logical volume you want to monitor, the output from **lvmstat** will look like Example 7-51.

Example 7-51 Using lvmstat without enabling statistics collection

```
[p630n06][/home/guest]> lvmstat -v localvg
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 localvg -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 localvg -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 7-52 there was no activity at all in the vg0 volume group:

Example 7-52 No activity lvmstat

```
[p630n06][/home/guest]> date;lvmstat -v localvg 1 10;print;date
Wed Oct 13 14:20:59 CDT 2004
..........
Wed Oct 13 14:21:08 CDT 2004
```

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 7-53.

Example 7-53 Using lvmstat with a volume group

```
[p630n06][/home/guest]> lvmstat -v localvg

Logical Volume       iocnt   Kb_read   Kb_wrtn      Kbps
  fslv01            307444  17413752   7924436     88.83
  testlv            204800  52428800         0    183.80
  loglv02            17592         0     70368      0.25
  lv00               13728   1048648     95892      4.01
  loglv03             3555         0     14220      0.05
```

This output shows that the most-utilized logical volumes since we turned on the statistical collection are `fslv01` and `testlv`. Example 7-54 shows the use of the -l <logical volume> flag to look at the logical partition statistics for logical volume `fslv01` and `testlv`.

Example 7-54 Using lvmstat with a single logical volume

```
[p630n06][/home/guest/2105]> lvmstat -l fslv01 | head

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       9       1   44401   1045184   1913536     10.37
       4       1   28318    860864   1823168      9.41
       6       1   23639   1048576         0      3.68
      14       1   23202    791104   1632580      8.49
       5       1   19741    946176         0      3.32
      15       1   17131    788544    934272      6.04
      13       1   16265    786432    290516      3.77
      16       1   11407    786432    262116      3.68

[p630n06][/home/guest]> lvmstat -l testlv | head

Log_part  mirror#  iocnt   Kb_read   Kb_wrtn      Kbps
       1       1    1024    262144         0      0.92
       2       1    1024    262144         0      0.92
       3       1    1024    262144         0      0.92
       4       1    1024    262144         0      0.92
       5       1    1024    262144         0      0.92
       6       1    1024    262144         0      0.92
       7       1    1024    262144         0      0.92
       8       1    1024    262144         0      0.92
```

From the output we see that the most-utilized logical partition for the `fslv01` logical volume is logical partition number 9, and that each partition was used equally for the `testlv` logical volume.

`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

7.2.6 The sar -d 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

This section will focus on the -d option that directly relates to storage.

sar syntax

/usr/sbin/**sar** [{ -A | [-a] [-b] [-c] [**-d**][-k] [-m] [-q] [-r] [-u] [-v] [-w] [-y] }] [-P ProcessorIdentifier, ... | ALL] [-ehh [:mm [:ss]]] [-XFile] [-fFile] [-iSeconds] [-oFile] [-shh [:mm [:ss]]] **[Interval [Number]]**

Useful combinations

- **`sar -d 5 60`** `Disk report at 5 second intervals for 60 iterations.`

Information about measurement and sampling

The sar command itself can generate a considerable number of reads and writes depending on the interval at which it is run. Run the sar statistics without the workload to understand the sar command's contribution to your total statistics. Reports activity for each block device with the exception of tape drives.

The activity data reported is:

%busy	Reports the portion of time the device was busy servicing a transfer request.
avque	Before AIX 5.3: Reports the instantaneous number of requests sent to disk but not completed yet. AIX 5.3: Reports the average number of requests waiting to be sent to disk.
read/s, write/s, blk/s	Reports the number of read-write transfers from or to a device. The number of bytes is transferred in 512-byte units.
avwait, avserv	Average wait time and service time per request in milliseconds.

Examples for sar -d

One of the nice features of the sar command is that it summarizes and provides an average when an Interval and Number is specified. Additionally it provides information on the disk queue and service times.

In Example 7-55 we see an example of the sar command with an interval of 5 seconds and the number of intervals being 3.

Example 7-55 sar -d 5 3

```
# sar -d 5 3

AIX p690_LPAR1 3 5 0022BE2A4C00    10/15/04

System configuration: lcpu=2 drives=5

15:48:35     device    %busy    avque    r+w/s    Kbs/s   avwait   avserv

15:48:40     hdisk0      0      0.0        0        0      0.0      0.0
             hdisk1    100    235.9      387     1660     11.8      7.7
               cd0       0      0.0        0        0      0.0      0.0
             hdisk2      0      0.0        0        0      0.0      0.0
             hdisk3      0      0.0        0        0      0.0      0.0


15:48:45     hdisk0      0      0.0        0        0      0.0      0.0
             hdisk1     99    235.7      396     1696      5.9      7.5
                cd0      0      0.0        0        0      0.0      0.0
             hdisk2      0      0.0        0        0      0.0      0.0
             hdisk3      0      0.0        0        0      0.0      0.0


15:48:50     hdisk0      0      0.0        0        0      0.0      0.0
             hdisk1     99    236.9      389     1679      8.3      7.7
```

```
                cd0      0      0.0       0       0      0.0      0.0
              hdisk2     0      0.0       0       0      0.0      0.0
              hdisk3     0      0.0       0       0      0.0      0.0


Average       hdisk0     0      0.0       0       0      0.0      0.0
              hdisk1    99    236.2     390    1678      8.7      7.6
                cd0      0      0.0       0       0      0.0      0.0
              hdisk2     0      0.0       0       0      0.0      0.0
              hdisk3     0      0.0       0       0      0.0      0.0
```

The report raises some red flags that indicate a number of performance issues. One hdisk is 100% busy and the other three are idle. There is a large queue of requests waiting for hdisk1. The data rate is 1.7 MB/second and each request is waiting on average 8.7 milliseconds and is taking 7.6 milliseconds to complete.

7.3 Tuning

In order to effectively tune the storage layer, it is important to understand the workload generated by the application. Without a good understand of the workload, and the subsequent load that is placed on the storage, tuning will likely be ineffective. Worse than that, improper tuning can degrade performance. Tuning cannot make up for bad data placement and design.

This section covers the commands that can be used to tune the I/O layer. For the discussion of data placement and design see.

7.3.1 The lsdev, rmdev and mkdev commands

When tuning disk storage you often need to work with the adapters and disks. Some tuning requires that the device is made unavailable (changed from available to defined). A major change like installing a new device driver may require the device to be completely removed from the system and then re-installed. The basic commands to work with adapters and disks at this level are **lsdev**, **rmdev**, and **mkdev**:

- The **lsdev** command displays devices in the system and their characteristics.
- The **rmdev** command unconfigures or both unconfigures and undefines devices (removes/deletes).
- The **mkdev** command makes available a previously defined device.

These commands reside in /usr/sbin and are part of the bos.rte.methods fileset, which is installed by default from the AIX base installation media.

lsdev

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

```
lsdev [ -C ][ -c Class ] [ -s Subclass ] [ -t Type ] [ -f File ] [ -F Format | -r ColumnName ] [ -h ] [ -H ] [ -l  { Name | - } ] [ -p Parent ] [ -S State ]

lsdev -P [ -c Class ] [ -s Subclass ] [ -t Type ] [ -f File ] [ -F Format | -r ColumnName ] [ -h ] [ -H ]
```

Useful combinations

- **lsdev -Cc tape** list all attached tape devices
- **lsdev -Cc disk** list all attached disk drives
- **lsdev -Cc adapter** list all adapters

rmdev

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

```
rmdev { -l | -p }Name [ -d | -S ] [ -f File ] [ -h ] [ -q ] [ -R ]
```

Useful combinations

- **rmdev -l hdisk4** Change status of disk to defined
- **rmdev -dl hdisk4** Undefine and remove device
- **rmdev -dl fcs0 -R** Unconfigure device and all children devices

mkdev

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

```
mkdev -l Name [ -h ] [ -q ] [ -S ]
```

Useful combinations

- **mkdev -l hdisk4** Change status of disk to available

Examples for lsdev, rmdev, mkdev

This paragraph presents usage examples for the **lsdev**, **rmdev**, and **mkdev** commands.

Using lsdev

The **lsdev** command can be used to list customized or predefined devices in the Device Configuration database. Customized devices are those which are defined to the operating system. Pre-defined devices are those which the operating system has information on how to configure if the device is attached to the system. If you want to list devices that are configured to the system, you use the -C flag. The command **lsdev -C** would list all devices of all types that are defined to the system. It is often helpful to restrict the output to a specific device class.

A common way to look at the disk is to use the command **lsdev -Cc disk**. A variation of that command which uses the -H flag to include header information is shown in Example 7-56.

Example 7-56 lsdev -CH -c disk

```
# lsdev -CH -c disk
name   status    location     description

hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
```

The output shows that there are three types of disk attached to the system: LVD SCSI, MPIO Other FC SCSI, and 1742-900. The MPIO Other FC SCSI is a generic device driver used by AIX for multi path disk drivers, for which AIX does not have information available in the Pre-defined Device Configuration database. The LVD SCSI disk drive is the basic locally attached device type. The 1742-900 is the DS4500 (FAStT 900) disk device. AIX 5.3 has built-in support for DS4500 disk devices.

The first column is the customized name of the device. Many commands take the customized name as an argument (**lsdev, mkdev, rmdev, lsattr, chdev, lscfg**). When specifying a specific device with one of those commands, you use the -l flag.

By using the same command, but changing the class to adapter, we can see all the device adapters defined to the system.

Example 7-57 lsdev -CH -c adapter

```
# lsdev -CH -c adapter
name  status    location description

ent0  Available 44-08    Gigabit Ethernet-SX PCI-X Adapter (14106802)
ent1  Available 47-08    10/100 Mbps Ethernet PCI Adapter II (1410ff01)
ent2  Available 4s-08    10/100 Mbps Ethernet PCI Adapter II (1410ff01)
ent3  Available 54-08    10/100 Mbps Ethernet PCI Adapter II (1410ff01)
ent4  Available 5F-08    Gigabit Ethernet-SX PCI-X Adapter (14106802)
fcs0  Available 41-08    FC Adapter
fcs1  Available 4Q-08    FC Adapter
sa0   Available          LPAR Virtual Serial Adapter
scsi0 Available 3s-08    Wide/Ultra-3 SCSI I/O Controller
scsi1 Available 5M-08    Wide/Ultra-3 SCSI I/O Controller
scsi2 Available 3A-08    Wide/Fast-20 SCSI I/O Controller
```

```
vsa0  Available          LPAR Virtual Serial Adapter
```

Using the location column from Example 7-56 on page 482 and Example 7-57 on page 482, it is easy to identify which adapter each hdisk is attached to. The disk hdisk5 at location `41-08-02` is attached through adapter fcs0 at location `41-08`.

In the case of MPIO disk drives this is misleading. It is necessary to use the **`lspath`** command with MPIO disk drives to see if the disk is available through other adapters.

IBM Enterprise Storage Server® considerations

ESS disk devices are configured as MPIO-capable or non-MPIO-capable depending on which ESS host attachment script is installed. To configure ESS devices as non-MPIO-capable devices, install the ibm2105.rte package with a version of 32.6.100.x. To configure ESS devices as MPIO-capable devices, install the devices.fcp.disk.ibm2105.mpio.rte package with a version of 33.6.100.y.

A visible difference between MPIO-capable and non-MPIO-capable is the number of disk devices reported by commands like **`lspv`** or **`lsdev -Cc disk`**. For non-MPIO-capable host attachment script (32.6.100.x ibm2105.rte), an hdisk device will show up for each path.

For each of the two types of ESS host attachment scripts, there is a corresponding device driver to facilitate path management. The two types of attachment scripts and two types of subsystem device drivers cannot be intermixed.

For the non-MPIO-capable 32.6.100.x ibm2105.rte attachment script, you install the Subsystem Device Driver (SDD or devices.**sdd**.52.rte). In addition to an hdisk for every path, after installing SDD you also get a logical disk device with the name vpathX (X is a unique number for each hdisk).

Example 7-58 non-MPIO-capable ESS devices

```
[p630n05][/]> lsdev -Cc disk | egrep "vpath|2105"
hdisk4  Available 1n-08-01      IBM FC 2105
hdisk5  Available 1n-08-01      IBM FC 2105
hdisk6  Available 1n-08-01      IBM FC 2105
hdisk7  Available 1n-08-01      IBM FC 2105
hdisk8  Available 11-08-01      IBM FC 2105
hdisk9  Available 11-08-01      IBM FC 2105
hdisk10 Available 11-08-01      IBM FC 2105
hdisk11 Available 11-08-01      IBM FC 2105
vpath0  Available               Data Path Optimizer Pseudo Device Driver
vpath1  Available               Data Path Optimizer Pseudo Device Driver
```

For the MPIO-capable 33.6.100.y devices.fcp.disk.ibm2105.pio.rte attachment script, you install the Subsystem Device Driver Path Control Module (SDDPCM or devices.**sddpcm**.52f.rte). When the ESS devices are configured as MPIO-capable devices, SDDPCM is loaded during the ESS device configuration and becomes part of the AIX MPIO SCSI/FCP (Fibre Channel Protocol) device driver. At the time of this publication, SDDPCM does not support HACMP, GPFS, SVC. With MPIO-capable devices, each hdisk shows up once regardless of home many paths. To see paths, you can use the **lspath** command as in Example 7-59.

Example 7-59 MPIO-capable ESS devices

```
#lsdev -Cc disk
hdisk0 Available 1S-08-00-8,0  16 Bit LVD SCSI Disk Drive
hdisk1 Available 1S-08-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk2 Available 1S-08-00-10,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 1S-08-00-11,0 16 Bit LVD SCSI Disk Drive
hdisk4 Available 1n-08-02      IBM MPIO FC 2105
hdisk5 Available 1n-08-02      IBM MPIO FC 2105
#lspath -l hdisk4
Enabled hdisk4 fscsi0
Enabled hdisk4 fscsi0
Enabled hdisk4 fscsi1
Enabled hdisk4 fscsi1
```

Using rmdev

The **rmdev** command can be used to unconfigure a device or to unconfigure and undefine a device. The command requires that any child devices be in a state where they can be undefined as well. For disks that belong to a volume group, the respective volume group must be varied off. Likewise for adapters.

To unconfigure a device, you issue the command **rmdev -l {name}**. The name can be identified by using the **lsdev** command as in Example 7-56 on page 482. When a device is unconfigured, the device status changes from `available` to *defined*.

Example 7-60 shows how to unconfigure a device with **rmdev**. The **lsdev** command is used before and after to show the status change.

Example 7-60 Unconfigure a device with rmdev

```
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
```

```
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
# rmdev -l hdisk3
hdisk3 Defined
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Defined   41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
```

When a device is in a defined state, you can modify attributes that cannot be modified when the device is an available state.

If a disk is in a volume group that is varied on, the **rmdev** command will fail as in Example 7-61

Example 7-61 rmdev with busy device

```
# rmdev -l hdisk3
Method error (/usr/lib/methods/ucfgdevice):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.
```

To completely remove a device from the system, use the **rmdev** command with the -d flag. Example 7-62 shows how to unconfigure and undefine a device with **rmdev**. The **lsdev** command is used before and after to show the status change.

Example 7-62 Unconfigure and undefine a device with rmdev

```
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Defined   41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
# rmdev -dl hdisk2
hdisk2 deleted
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk3 Defined   41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
```

This disk, hdisk2, has been completely removed from the system. To bring it back you need to run **cfgmgr**.

Using mkdev

The **mkdev** command makes available the previously defined device specified by the given device logical name (-l Name flag). At times you may need to unconfigure a device in order to make changes to the device attributes. Once the changes are made, the **mkdev** command is used to make the device available.

Example 7-63 shows how to make a device available with **mkdev**. The **lsdev** command is used before and after to show the status change.

Example 7-63 mkdev example

```
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk3 Defined   41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
# mkdev -l hdisk3
hdisk3 Available
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
```

The hdisk, hdisk2 is still undefined from the rmdev command from Example 7-62 on page 485. To bring back the device, we can run **cfgmgr** which detects and defines devices that are attached to the system (see Example 7-64).

Example 7-64 cfgmgr to bring back removed devices

```
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
# cfgmgr
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 4Q-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 41-08-02     1742-900 (900) Disk Array Device
```

The disk, hdisk2, is now available for use on the system.

7.3.2 The lscfg, lsattr, and chdev commands

In 7.3.1, “The lsdev, rmdev and mkdev commands” on page 480 we discussed how to list, configure, and change the status of devices. The focus will be on disk devices and disk adapters. The commands to list and change device attributes are:

- The **lscfg** command displays configuration, diagnostic, and vital product data (VPD)
- The **lsattr** command displays attribute characteristics and possible values of attributes for devices in the system.
- The **chdev** command changes the characteristics of a device.

These commands reside in /usr/sbin, **lsattr** and **chdev** are part of the bos.rte.methods fileset, lscfg is part of the bos.rte.diag fileset, both of which are installed by default from the AIX base installation media

lscfg

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

lscfg [-v] [-p] [-s] [-l Name]

Useful combinations

- **lscfg -vl hdisk4** List detailed about a specific device

lsattr

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

```
lsattr { -D [ -O ] | -E [ -O ] | -F Format [ -Z Character ] } -l Name [-a
Attribute ] ...[ -f File ] [ -h ] [ -H ]

lsattr -R { -l Name | [ -c Class ] [ -s Subclass ] [ -t Type ] } -a Attribute [
-f File ] [ -h ] [ -H ]
```

Useful combinations

- **lsattr -El fcs0** List attributes of a specific device. Attributes labeled true can be changed.

chdev

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

```
chdev -l Name [ -a Attribute=Value ... ] [ -f File ] [ -h ] [ -p ParentName ] [
-P | -T ] [ -q ] [ -w ConnectionLocation ]
```

Useful combinations

- **chdev -l ent0 -a jumbo_frames=yes** Change the value of an attribute for a specific device.

Examples for lscfg, lsattr, chdev

Using lscfg

The lscfg can be used to display configuration, diagnostic, and vital product data (VPD). Each device and each device type has different characteristics that are displayed. To understand what the fields mean, you need to refer to the product documentation for the specific device.

In Example 7-65 we take a look at the lscfg output for a fibre channel adapter. We use the -v flag for verbose output, otherwise only the first line shown would be reported. We also use th -l flag to specify the name of the device that we are interested in.

Example 7-65 lscfg for fibre channel adapter

```
# lscfg -vl fcs0
  fcs0             U1.5-P1-I1/Q1  FC Adapter

        Part Number.................09P5079
        EC Level....................A
        Serial Number...............1C2250AE1A
        Manufacturer................001C
        Feature Code................2765
        FRU Number.................09P5080
        Network Address.............10000000C92D6BBA
        ROS Level and ID............02C03951
        Device Specific.(Z0)........2002606D
        Device Specific.(Z1)........00000000
        Device Specific.(Z2)........00000000
        Device Specific.(Z3)........03000909
        Device Specific.(Z4)........FF401210
        Device Specific.(Z5)........02C03951
        Device Specific.(Z6)........06433951
        Device Specific.(Z7)........07433951
        Device Specific.(Z8)........20000000C92D6BBA
        Device Specific.(Z9)........CS3.91A1
        Device Specific.(ZA)........C1D3.91A1
        Device Specific.(ZB)........C2D3.91A1
        Device Specific.(YL)........U1.5-P1-I1/Q1
```

For this adapter, the more useful fields are the FRU Number, Network Address, and Device Specific.(Z9).

If you do not have physical access to the machine to check the adapter type label, you can search the IBM Web site using the FRU Number. A search of the IBM Web site indicates that this adapter is a 6228 adapter. At the time of this publication, adapter microcode downloads are located at:

http://techsupport.services.ibm.com/server/mdownload/adapter.html

Checking the readme for the 6228 adapter shows that field Z9 is the microcode level. The level is determined by dropping the CS, for CS3.91A1, the adapter microcode level is 3.91A1.

The other useful field is the Network Address. This is the World Wide Name (WWN) for this fibre channel adapter. The WWN is useful for fibre channel switch zoning and for disk subsystem configuration.

For disk devices, the **lscfg** output provides different information. For DS4000 devices (FAStT) there is no additional information. Example 7-66 shows the lscfg output for a DS4000 disk device.

Example 7-66 lscfg -vl for DS4000 (FAStT)

```
# lscfg -vl hdisk5
  hdisk5           U1.5-P1-I1/Q1-W200200A0B812106F-L9000000000000  1742-900
(900 ) Disk Array Device
```

For ESS 2105 storage, the ESS host attachment script needs to be installed for AIX to correctly identify the disk device“IBM Enterprise Storage Server® considerations” on page 483. If the ESS host attachment script is not installed, an ESS disk device will show up as `MPIO Other FC SCSI Disk Drive`.

Example 7-67 lscfg -vl for ESS without host attachment script

```
# lscfg -vl hdisk2
  hdisk2           U1.5-P1-I1/Q1-W5005076300CD9589-L5313000000000000
                                       MPIO Other FC SCSI Disk Drive

        Manufacturer................IBM
        Machine Type and Model......2105800
        Part Number.................
        ROS Level and ID............312E3632
        Serial Number...............31322513
        EC Level....................
        FRU Number..................
        Device Specific.(Z0)........000003329F001002
        Device Specific.(Z1)........
        Device Specific.(Z2)........0013
        Device Specific.(Z3)........16602
        Device Specific.(Z4)........
        Device Specific.(Z5)........
```

```
        Device Specific.(Z6)........
```

With the ESS MPIO-capable attachment script, ESS devices show up as in Example 7-68.

Example 7-68 lscfg -vl for ESS with host attachment script

```
# lscfg -vl hdisk2
  hdisk2           U1.5-P1-I1/Q1-W5005076300CD9589-L5313000000000000
                                                          IBM MPIO FC 2105

        Manufacturer................IBM
        Machine Type and Model......2105800
        Serial Number...............31322513
        EC Level....................1.62
        Device Specific.(Z0)........10
        Device Specific.(Z1)........00AC
        Device Specific.(Z2)........0013
        Device Specific.(Z3)........16602
        Device Specific.(Z4)........05
        Device Specific.(Z5)........00
```

The **lscfg** output tell us that the ESS storage is a 2105 model 800. And the serial number of the disk is 31322513. The serial number corresponds to the volume label from the ESS Specialist interface.

Using lsattr

The **lsattr** command displays attribute characteristics and possible values of attributes for devices in the system. Each device has different characteristics that can be modified. To change attribute characteristics the **chdev** command is used. Some attribute changes require that the device be in a defined state. To change a device state, use the **rmdev** and **mkdev** commands.

To display the effective characteristics of a device and to see which attributes can be changed, use the -El flags as in example Example 7-69. The fibre channel adapter also has a child device, fscsi0, which has attributes as well.

Example 7-69 lsattr -El for fibre channel adapters

```
# lsattr -El fcs0
bus_intr_lvl  547       Bus interrupt level                             False
bus_io_addr   0xfc00    Bus I/O address                                 False
bus_mem_addr  0xe0020000 Bus memory address                             False
init_link     al        INIT Link flags                                 True
intr_priority 3         Interrupt priority                              False
lg_term_dma   0x800000  Long term DMA                                   True
max_xfer_size 0x100000  Maximum Transfer Size                           True
```

```
num_cmd_elems 200        Maximum number of COMMANDS to queue to the adapter True
pref_alpa     0x1        Preferred AL_PA                                    True
sw_fc_class   2          FC Class for Fabric                                True

# lsattr -El fscsi0
attach       switch       How this adapter is CONNECTED         False
dyntrk       no           Dynamic Tracking of FC Devices        True
fc_err_recov delayed_fail FC Fabric Event Error RECOVERY Policy True
scsi_id      0x661600     Adapter SCSI ID                       False
sw_fc_class  3            FC Class for Fabric                   True
```

The output of lsattr -El has four columns and from left to right they are:

- attribute — attribute name, used in **chdev**
- value — current setting
- description — description
- user_settable — False=not-settable , True=settable

For ESS devices, the default settings are different depending on whether the ESS host attachment script as installed prior to configuring the disk devices. In Example 7-70 and Example 7-70 the differences between with and without the host attachment script is visible.

Example 7-70 lsattr -El for ESS without host attachment script

```
# lsattr -El hdisk2
PCM             PCM/friend/fcpother              Path Control Module             False
algorithm       fail_over                        Algorithm                       True
clr_q           no                               Device CLEARS its Queue on error True
dist_err_pcnt   0                                Distributed Error Sample Time   True
dist_tw_width   50                               Distributed Error Sample Time   True
hcheck_interval 0                                Health Check Interval           True
hcheck_mode     nonactive                        Health Check Mode               True
location                                         Location Label                  True
lun_id          0x5313000000000000               Logical Unit Number ID          False
max_transfer    0x40000                          Maximum TRANSFER Size           True
node_name       0x5005076300c09589               FC Node Name                    False
pvid            0000331209edfde20000000000000000 Physical volume identifier      False
q_err           yes                              Use QERR bit                    True
q_type          simple                           Queuing TYPE                    True
queue_depth     1                                Queue DEPTH                     True
reassign_to     120                              REASSIGN time out value         True
reserve_policy  single_path                      Reserve Policy                  True
rw_timeout      30                               READ/WRITE time out value       True
scsi_id         0x651000                         SCSI ID                         False
start_timeout   60                               START unit time out value       True
ww_name         0x5005076300cd9589               FC World Wide Name              False
```

Of particular interest is the queue_depth value of 1, and the algorithm value of fail_over.

Example 7-71 lsattr -El for ESS without host attachment script

```
# lsattr -El hdisk2
PCM             PCM/friend/sddpcm                PCM                                             True
PR_key_value    none                             Reserve Key                                     True
algorithm       load_balance                     Algorithm                                       True
dist_err_pcnt   0                                Distributed Error Percentage                    True
dist_tw_width   50                               Distributed Error Sample Time                   True
hcheck_interval 20                               Health Check Interval                           True
hcheck_mode     nonactive                        Health Check Mode                               True
location                                         Location Label                                  True
lun_id          0x5313000000000000               Logical Unit Number ID                          True
lun_reset_spt   yes                              Support SCSI LUN reset                          True
node_name       0x5005076300c09589               FC Node Name                                    False
pvid            0000331209edfde20000000000000000 Physical volume identifier                      False
q_type          simple                           Queuing TYPE                                    True
qfull_dly       20                               delay in seconds for SCSI TASK SET FULL         True
queue_depth     20                               Queue DEPTH                                     True
reserve_policy  no_reserve                       Reserve Policy                                  True
rw_timeout      60                               READ/WRITE time out value                       True
scbsy_dly       20                               delay in seconds for SCSI BUSY                  True
scsi_id         0x651000                         SCSI ID                                         True
start_timeout   180                              START unit time out value                       True
ww_name         0x5005076300cd9589               FC World Wide Name                              False
```

With the ESS host attachment script installed prior to defining the disk devices, the attributes for the disk are automatically set to values that improve performance.

For DS4000 (FAStT) devices, two additional devices besides hdisks are present to the operating system. The two additional devices are the Disk Array Router (darX) and the Disk Array Controller (dacX).

Example 7-72 shows attributes for DS4000 disk devices. Some of the attributes that are listed as False, can be changed, but not from the operating system. For example, the cache_method can be changed at the disk subsystem. Once the change is made there, the device can be reconfigured to detect the change.

Example 7-72 attributes for DS4000 disk devices

```
# lsattr -El hdisk4
PR_key_value   none                             Persistant Reserve Key Value          True
cache_method   fast_write                       Write Caching method                  False
ieee_volname   600A0B800012106E0000002140960E7F IEEE Unique volume name              False
lun_id         0x0005000000000000               Logical Unit Number                   False
```

```
max_transfer    0x100000                         Maximum TRANSFER Size                  True
prefetch_mult   1                                Multiple of blocks to prefetch on read False
pvid            000684ffbecd9b200000000000000000 Physical volume identifier            False
q_type          simple                           Queuing Type                           False
queue_depth     10                               Queue Depth                            True
raid_level      5                                RAID Level                             False
reassign_to     120                              Reassign Timeout value                 True
reserve_policy  single_path                      Reserve Policy                         True
rw_timeout      30                               Read/Write Timeout value               True
scsi_id         0x660100                         SCSI ID                                False
size            16384                            Size in Mbytes                         False
write_cache     yes                              Write Caching enabled                  False
```

Example 7-73 Attributes for DS4000 array and controller devices

```
# lsattr -El dac0
GLM_type        low               GLM type                 False
alt_held_reset  no                Alternate held in reset  False
cache_size      1024              Cache Size in MBytes     False
controller_SN   1T34058487        Controller serial number False
ctrl_type       1742-0900         Controller Type          False
location                          Location Label           True
lun_id          0x0               Logical Unit Number      False
node_name       0x200200a0b812106e FC Node Name            False
passive_control no                Passive controller       False
scsi_id         0x650100          SCSI ID                  False
utm_lun_id      0x001f000000000000 Logical Unit Number     False
ww_name         0x200200a0b812106f World Wide Name         False
# lsattr -El dar0
act_controller dac0,dac1 Active Controllers                        False
aen_freq       600       Polled AEN frequency in seconds           True
all_controller dac0,dac1 Available Controllers                     False
autorecovery   no        Autorecover after failure is corrected    True
balance_freq   600       Dynamic Load Balancing frequency in seconds True
cache_size     1024      Cache size for both controllers           False
fast_write_ok  yes       Fast Write available                      False
held_in_reset  none      Held-in-reset controller                  True
hlthchk_freq   600       Health check frequency in seconds         True
load_balancing no        Dynamic Load Balancing                    True
switch_retries 5         Number of times to retry failed switches  True
```

The **lsattr** command can be used to show the default values for a device or a specific attribute. Example 7-74 on page 494 shows the default value for a specific attribute. To get all the default values for a device, omit the -a flag in the command.

Example 7-74 default value for a specific attribute

```
# lsattr -D -l hdisk3 -a queue_depth
queue_depth 20 Queue DEPTH True
```

The default value for the attribute queue_depth for device hdisk3 is 20.

The lsattr command can be used to show possible values for a specific attribute. Example 7-75 shows how to get the possible values for a specific attribute.

Example 7-75 possible values for a specific attribute

```
# lsattr -R -l hdisk3 -a queue_depth
1...256 (+1)
```

The possible values for the attribute queue_depth for device hdisk3 are 1-256.

Using chdev

The **chdev** command changes the characteristics of a device. The **lsattr** command is useful in determining which values can be changed and what the possible values are.

Many devices attributes require the device to not be in use in order to make a change. It may be necessary to change the device status to defined. Changing the device status to defined is done with the **rmdev** command.

In Example 7-76 we change the queue_depth for a DS4500 disk device from 10 to 20.

Example 7-76 chdev for a disk device

```
# lsattr -El hdisk5 -a queue_depth
queue_depth 10 Queue Depth True
# chdev -l hdisk5 -a queue_depth=20
hdisk5 changed
# lsattr -El hdisk5 -a queue_depth
queue_depth 20 Queue Depth True
```

This works without errors because hdisk5 does not belong to a volume group and is in a state where its attributes can be changed.

In Example 7-77 we have to unmount a filesystem in order to vary off the volume group to which the hdisk belongs.

Example 7-77 steps to change an active disk device

```
# lsattr -El hdisk4 -a queue_depth
queue_depth 10 Queue Depth True
```

```
# chdev -l hdisk4 -a queue_depth=20
Method error (/etc/methods/chgfcparray):
        0514-062 Cannot perform the requested function because the
                 specified device is busy.

# umount /fastfs
# varyoffvg fastvg
# chdev -l hdisk4 -a queue_depth=20
hdisk4 changed
# lsattr -El hdisk4 -a queue_depth
queue_depth 20 Queue Depth True
```

7.3.3 The ioo command

The **ioo** command manages all the I/O-related tuning parameters, while the **vmo** command manages all the other Virtual Memory Manager, or VMM, parameters previously managed by the vmtune command. The commands are part of the bos.perf.tune fileset, which also contains the **tunsave**, **tunrestore**, **tuncheck**, and **tundefault** commands.

Misuse of the ioo command can cause performance degradation or operating-system failure. Before experimenting with ioo, you should be thoroughly familiar with the Virtual Memory Manager (VMM). For more details, consult also Chapter 5, “Memory analysis and tuning” on page 297.

ioo syntax

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

```
ioo [ -p | -r ] { -o Tunable [ =NewValue ] } | {-d Tunable} | -D | -a
ioo -L [ Tunable ]
```

Useful combinations

- **ioo -L** Table of tunables
- **ioo -ra** Show reboot values
- **ioo -o maxpgahead=16** Change tunable value
- **ioo -h maxpgahead** Show help for a tunable

Examples for ioo

The **ioo** command has many parameters that can be tuned. Due to its system impact, the **ioo** command can only be executed by the root user. The -L flag can be used to list one or all tunables. Example 7-78 on page 496 shows the first 15 lines and the last 15 lines from the **ioo -L** command. The full table can viewed on an AIX 5.3 system or looked up in the command reference documentation.

Example 7-78 ioo -L output

```
# ioo -L
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
------------------------------------------------------------------------------
minpgahead                2      2      2      0      4K     4KB pages         D
     maxpgahead
------------------------------------------------------------------------------
maxpgahead                8      8      8      0      4K     4KB pages         D
     minpgahead
------------------------------------------------------------------------------
pd_npages                 64K    64K    64K    1      512K   4KB pages         D
------------------------------------------------------------------------------
maxrandwrt                0      0      0      0      512K   4KB pages         D
------------------------------------------------------------------------------
numclust                  1      1      1      0      2G-1   16KB/cluster      D
------------------------------------------------------------------------------
numfsbufs                 186    186    186    1      2G-1                     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      64K    4KB pages         D
------------------------------------------------------------------------------
j2_maxPageReadAhead       128    128    128    0      64K    4KB pages         D
------------------------------------------------------------------------------
j2_nBufferPerPagerDevice  512    512    512    0      256K                     M
------------------------------------------------------------------------------
j2_nPagesPerWriteBehindCluster
                          32     32     32     0      64K                      D
------------------------------------------------------------------------------
j2_maxRandomWrite         0      0      0      0      64K    4KB pages         D
------------------------------------------------------------------------------
j2_nRandomCluster         0      0      0      0      64K    16KB clusters     D
------------------------------------------------------------------------------
jfs_clread_enabled        0      0      0      0      1      boolean           D
------------------------------------------------------------------------------
jfs_use_read_lock         1      1      1      0      1      boolean           D
------------------------------------------------------------------------------
j2_inodeCacheSize         400    400    400    1      1000                     D
------------------------------------------------------------------------------
j2_metadataCacheSize      400    400    400    1      1000                     D
------------------------------------------------------------------------------
pv_min_pbuf               256    256    256    256    2G-1                     D
------------------------------------------------------------------------------
j2_dynamicBufferPreallocation
                          16     16     16     0      256    16k slabs         M
```

```
------------------------------------------------------------------------------
j2_maxUsableMaxTransfer   512    512    512    1      4K     pages             M
------------------------------------------------------------------------------
j2_non_fatal_crashes_system
                          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
```

Example 7-79 shows all of the reboot values for **ioo** that will be used on the next boot of the system.

Example 7-79 Show all reboot values with ioo

```
# ioo -ra
                    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 = 128
       j2_nBufferPerPagerDevice = 512
j2_nPagesPerWriteBehindCluster = 32
              j2_maxRandomWrite = 0
              j2_nRandomCluster = 0
             jfs_clread_enabled = 0
              jfs_use_read_lock = 1
              j2_inodeCacheSize = 400
            j2_metadataCacheSize = 400
                      pv_min_pbuf = 256
 j2_dynamicBufferPreallocation = 16
```

```
      j2_maxUsableMaxTransfer = 512
   j2_non_fatal_crashes_system = 0
```

Specific help for each tunable can be displayed using the -h flag as shown in Example 7-80.

Example 7-80 Displaying help for ioo tunable parameter

```
# ioo -h maxpgahead
Help for tunable maxpgahead:
Specifies the maximum number of pages to be read ahead when processing a
sequentially accessed file. Default: 8 (the default should be a power of two
and should be greater than or equal to minpgahead); Range: 0 to 4096. Observe
the elapsed execution time of critical sequential-I/O-dependent applications
with the time command. Because of limitations in the kernel, do not exceed 512
as the maximum value used. The difference between minfree and maxfree should
always be equal to or greater than maxpgahead. If execution time decreases with
higher maxpgahead, observe other applications to ensure that their performance
has not deteriorated.
```

Changing tunable values

Before modifying any tunable parameter, you should first carefully read about all its characteristics. Detailed information on each tunable can be found in the AIX product documentation. You must then make sure that the Diagnosis and Tuning sections for this parameter truly apply to your situation and that changing the value of this parameter could help improve the performance of your system.

You can set tunables using the -o option. Example 7-81 shows how to increase the value of maxpgahead.

Example 7-81 Increasing value of maxpgahead using ioo

```
# ioo -o maxpgahead=16
Setting maxpgahead to 16
```

However the help for this tunable indicates that you have to also make sure the difference between minfree and maxfree is greater than or equal to maxpgahead. Since the default values for minfree and maxfree are 120 and 128 respectively, we either need to change those values, or set maxpgahead back to its default value. Example 7-82 shows how to set a tunable back to its default value.

Example 7-82 Restoring a tunable to its default value using ioo.

```
# ioo -d maxpgahead
Setting maxpgahead to 8
```

7.3.4 The lvmo command

The **lvmo** command sets or displays pbuf tuning parameters. Misuse of the lvmo command can cause performance degradation or operating-system failure. You must have root authority to run this command.

The **lvmo** command is part of the bos.rte.lvm fileset that is installed during installation of the operating system.

lvmo syntax

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

```
lvmo -v Name -o Tunable [ =NewValue ]
lvmo -a
```

Useful combinations

- **lvmo -a** Show LVM pbuf tunable values
- **lvmo -v rootvg -o pv_pbuf_count=2048** Change pbuf value

Examples for lvmo

The **lvmo** command follows a similar convention to the commands **vmo** and **ioo**. The -v flag allows you to specify a volume group for the commands to take place. The default volume group is rootvg (see Example 7-83).

Example 7-83 listing pbuf statistics for a volume group

```
# lvmo -v dasvg -a
vgname = dasvg
pv_pbuf_count = 512
total_vg_pbufs = 512
max_vg_pbuf_count = 16384
pervg_blocked_io_count = 7455
global_pbuf_count = 512
global_blocked_io_count = 7455
```

The **lvmo** command has three tunables:

pv_pbuf_count	The number of pbufs that will be added when a physical volume is added to the volume group.
max_vg_pbuf_count	The maximum number of pbufs that can be allocated for the volume group. The volume group must be varied off and varied on again for this value to take effect.
global_pbuf_count	The minimum number of pbufs that will be added when a physical volume is added to any volume group.

To increase the pbufs for a physical volume added to a specific volume group, you need to specify the -v flag as well as the pv_pbuf_count tunable. Example 7-84 shows how this is done.

Example 7-84 Increase pbufs for a specific volume group

```
# lvmo -v dasvg -o pv_pbuf_count=1024
# lvmo -v dasvg -a
vgname = dasvg
pv_pbuf_count = 1024
total_vg_pbufs = 1024
max_vg_pbuf_count = 16384
pervg_blocked_io_count = 7455
global_pbuf_count = 512
global_blocked_io_count = 7455
```

7.3.5 The vmo command

The **vmo** command manages Virtual Memory Manager tunable parameters. Some of these parameters have an effect on storage performance. Notably the minfree and maxfree parameters which are tightly associated with the maxpgahead and minpgahead parameters of the **ioo** command.

For more detail on the vmo command see 5.2.1, “The vmo command” on page 317.

Part 3

Case studies and miscellaneous tools

Part 3 provides two case studies for performance monitoring and tuning a NIM server and gives a practical example of using the new tools and features in an IBM @server p5 with Micro-Partitioning and SMT.

We describe how to use the Workload Manager and Partition Load Manager for performance monitoring and system analysis and introduce the Resource Monitoring and Control (part of RSCT) functionality for monitoring system performance.

In Chapter 10, “Performance monitoring APIs” on page 583 we also provide information about the Perfstat API and usage examples for programming applications to use this interface.

8

Case studies

This chapter presents practical examples for a NIM environment and a POWER5 case study.

NIM case study

In the first section of this chapter we go through the performance tuning process for an AIX system that is using the Network Installation Management (NIM) software. Network Install Manager is a complex application which relies on several subsystems to provide software installation and maintenance in an AIX, or a mixed AIX/Linux environment (from AIX 5L V5.3).

NIM uses a client server model which provides clients with all the necessary resources for booting, installing, maintaining or diagnosing (AIX only) client machines. In a NIM environment, the following subsystems should be considered as candidates for performance tuning: network (TCP/IP), NFS, Virtual Memory Manager, Disk I/O and Logical Volume Manager.

POWER5 case study

This section considers specific POWER5 performance issues. We monitor the CPU performance of a POWER5-based system using a simple case scenario.

8.1 Case study: NIM server

In this case study we utilized an IBM @server pSeries model 690 server that was configured into four separate logical partitions (LPAR). Each partition included one 10/100 Ethernet adapter, one gigabit Ethernet adapter, and one internal SCSI hard disk. Each partition had two processors and 4GB of RAM. The partition configured as the NIM master (server) also has two fibre channel adapters and an additional local SCSI hard disk assigned to it.

For our first test, we had all partitions using the 10/100 Ethernet adapters, connected to a switch, with effective link parameters of 100Mbps, full duplex. The NIM server resources were allocated on the second internal SCSI hard disk. Later, we configured NIM to use the gigabit ethernet (connected to a GbitE switch), and also used the external fibre channel (FC) storage, connected via a Storage Area Network (a FC switch). A diagram of our test environment is presented in Figure 8-1.

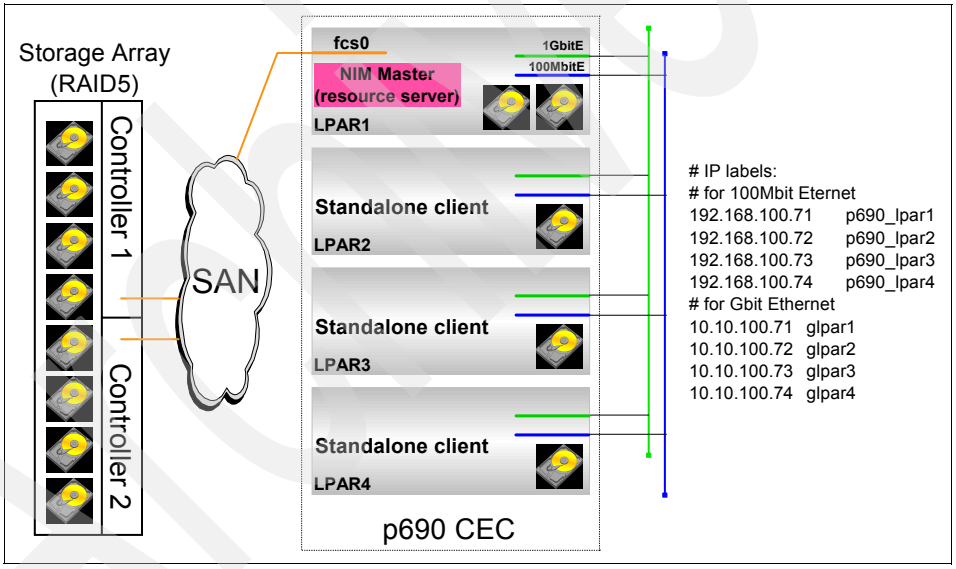

Figure 8-1 Test environment NIM diagram

8.1.1 Setting up the environment

We have installed the NIM master (in our case LPAR1) from AIX installation CD-ROMs with AIX 5L V5.3. For this purpose, we have assigned the CD-ROM drive available in the media drawer to LPAR1.

Once installed, we have configured the NIM master and start defining the resources to be used in our environment.

In our environment we want to install the three remaining LPARs (LPAR2, LPAR3, and LPAR4) form the NIM master. For this purpose, in the initial phase we need to define the following resources:

1. A NIM repository containing the software packages to be used for installing the clients (similar to the content of the AIX installation CD set). This type of resource is known as “lpp_source”.
2. A repository containing the binaries (executables) and libraries to be used for executing various operations (programs) during clients’ installation, together with the kernel image used for booting the clients. This is known as “Shared Product Object Tree” (SPOT), and is in fact a directory similar to /usr file system.
3. The three clients (LPAR2, LPAR3, and LPAR4), which are defined in the NIM environment as “Standalone” machines (after installation they will boot from their own disk and will run an independent copy of the operating system).

   These machines are defined using the MAC address of the network (Ethernet) adapter to be used for installation and an associated IP address.

While defining the resources mentioned before, we observed that defining the LPPSOURCE and SPOT type resources is very I/O disk resource demanding:

- In fact, creation of the LPPSOURCE consists of copying the necessary LPPs (Licensed Program Products) from the AIX installation CD-ROMs to a designated space on the disk.
- Also, creating the SPOT, is similar to an installation process, where the installation takes place on the defined disk (file system) space.

Once these resources are created, we proceed to installing the clients. Installing the clients in a NIM environment can be of three types: SPOT installation, bos rte, and mksysb. During initial client installation (of bos rte type), the following resources are used:

- The bootp server (to allow clients to boot over the network)
- The tftp server (to transfer the kernel to be loaded by the clients)
- The NFS subsystem (to run the install programs and to retrieve the necessary LPPs for installing the client)

Since the NIM software repository resides on a file system, and this file system is NFS exported to the NIM clients, the following subsystems are also involved during the installation process:

- The Virtual Memory Manager
- The TCP/IP subsystem
- The NFS subsystem

– The Logical Volume Manager

Thus, we found useful to tune these subsystems for obtaining the maximum performance for our NIM master. We started by monitoring an idle NIM master, and then gradually, tried to identify the bottlenecks during various NIM operations.

8.1.2 Monitoring NIM master using topas

During client installation process, to begin the performance tuning process for the system, we start by monitoring the system using **topas**.

Example 8-1 topas output for NIM server

```
Topas Monitor for host:    p690_lpar1           EVENTS/QUEUES    FILE/TTY
Fri Oct 22 17:23:42 2004   Interval:  2         Cswitch    1040  Readch        0
                                                Syscall     204  Writech     199
Kernel    6.2   |##                          |  Reads         0  Rawin         0
User      0.0   |                            |  Writes        0  Ttyout      199
Wait      0.0   |                            |  Forks         0  Igets        0
Idle     93.8   |########################### |  Execs         0  Namei        2
                                                Runqueue     0.0  Dirblk       0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue    0.0
en1   11473.5   8387.0   705.0   458.8 22488.2
lo0       0.0      0.0     0.0     0.0     0.0  PAGING           MEMORY
                                                Faults         0  Real,MB    4095
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  Steals         0  % Comp     17.9
hdisk1   12.5  11136.0    90.0 22272.0     0.0  PgspIn         0  % Noncomp  27.6
...(lines omitted)...
```

From the topas output, the only resource that is running at maximum speed is the ethernet adapter en1. A 10/100 adapter running at 100_Full_Duplex, has a transmit speed limit of approximately 10 Megabytes per second (10 MB/second = 10,000 KB/second). The **topas** output shows en1 at 11,437.5 KB/second. None of the other devices are overutilized at this point. The direct attached SCSI disk hdisk1, which contains the file systems being used for the NIM resources, is only 12.5 percent busy.

When a resource is running at its maximum speed, and is the limiting factor in the system, additional resources need to be added. In this case, tuning will not be able to compensate for the device data rate limit of the ethernet adapter.

To increase the network throughput bandwidth, we have chosen to allocate a Gigabit Ethernet adapter to the system. A Gigabit Ethernet adapter has a one direction maximum data rate of 100 Mbytes per second, 10 times the rate of a 100Mbit adapter.

Creating a benchmark

One of the difficulties with just monitoring the NIM server as it handles NIM client requests, is that the workload varies. For tuning, it is desirable to have a representative workload that can be used as a benchmark. This benchmark workload can be run before and after tuning to see if any improvement occurs. This does not eliminate the need to monitor the system to determine actual workload benefits from tuning, but does provide a useful way to see if performance tuning is helping a related workload.

Since NIM uses NFS to transfer data between the server and clients, a simple workload is to mount an NFS directory on the clients and generate I/O with the **dd** command. The NIM server can **rsh** to the NIM clients which will be useful in automating the workload.

Example 8-2 shows how to export the directory that is being used for NIM resources. Then we use the remote shell (rsh) to mount the exported directory on each of the NIM clients.

Example 8-2 setting up for NIM benchmark run

```
# exportfs -i -o root=glpar2:glpar3:glpar4 /dasbk
# for i in 2 3 4 ; do
> rsh glpar$i "mount 192.168.100.71:/dasbk /mnt"
> done
```

To facilitate the benchmarking effort we created two scripts, one for generating read I/O and one for generating write I/O.

Example 8-3 NIM write I/O benchmark script

```
#!/usr/bin/ksh

for i in 2 3 4
  do
    rsh glpar$i "dd if=/dev/zero of=/mnt/file$i bs=128K count=8000" &
  done

#wait command waits for all background processes to finish before continuing
wait
```

Example 8-4 NIM read I/O benchmark script

```
#!/usr/bin/ksh

for i in 2 3 4
  do
    rsh glpar$i "dd if=/mnt/file$i of=/dev/null bs=128K count=8000" &
  done
```

```
#wait command waits for all background processes to finish before continuing
wait
```

We first execute the **writenim.sh** with the **timex** command to get the total run time of the script.

Example 8-5 Script for NIM write I/O benchmark

```
# timex ./writenim.sh
8000+0 records in
8000+0 records out
8000+0 records in.
8000+0 records out.
8000+0 records in.
8000+0 records out.

real 386.79
user 0.02
sys  0.00
```

We observed that it took 386.79 seconds to write 3000 MB (1000 MB per NIM client). This gives us a throughput of 7.75 MB/second. We also collected a topas screen output, which shows similar results to the actual workload from Example 8-1 on page 506.

Example 8-6 topas output from write I/O benchmark script

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Tue Oct 26 10:19:18 2004   Interval:  2        Cswitch    1750  Readch        0
                                               Syscall     207  Writech      95
Kernel   13.0   |####                       |  Reads         0  Rawin         0
User      0.0   |                           |  Writes        0  Ttyout       95
Wait      0.0   |                           |  Forks         0  Igets         0
Idle     87.0   |#########################  |  Execs         0  Namei         2
                                               Runqueue    0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
en1   12011.6  16283.0  1748.0 23809.5   213.7
en0       0.1      3.0     0.0     0.1     0.0  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults        1  Real,MB    4095
                                               Steals        0  % Comp     12.8
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp  10.7
hdisk1   84.5  11538.0   358.5     4.0 23072.0  PgspOut       0  % Client   10.9
```

In order to get an accurate result from the read I/O script, we need to unmount all the related filesystems. This is necessary to flush the caches, that is both the

filesystem cache of the NIM server as well as the NFS client caches of the NIM clients.

Example 8-7 script for NIM read I/O benchmark

```
# umount /dasbk
# mount /dasbk
# for i in 2 3 4 ; do
> rsh glpar$i umount /mnt
> rsh glpar$i mount 192.168.100.71:/dasbk /mnt
> done
# timex ./readnim.sh
8000+0 records in.
8000+0 records out.
8000+0 records in
8000+0 records out
8000+0 records in.
8000+0 records out.

real 268.74
user 0.02
sys  0.00
```

The read benchmark executed in 268.74 seconds, so the read throughput was 11.16 MB/second (3000 MB / 268.74 second). In Example 8-8 we collected a topas screen output which indicates that we have reached the limit of hdisk0 as well.

Example 8-8 topas output from read I/O benchmark script

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Tue Oct 26 10:32:38 2004   Interval:  2        Cswitch    1232  Readch        0
                                               Syscall     204  Writech     170
Kernel   10.0   |###                         | Reads         0  Rawin         0
User      0.0   |                            | Writes        0  Ttyout      170
Wait     46.2   |##############              | Forks         0  Igets         0
Idle     43.8   |#############               | Execs         0  Namei         2
                                               Runqueue    0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   3.0
en1   11799.0   8635.0   801.0   488.0 23110.1
en0       0.5      5.0     0.0     1.0     0.0  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults      197  Real,MB    4095
                                               Steals        0  % Comp     12.8
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp   5.5
hdisk1  100.0  11494.0   302.5 22988.0     0.0  PgspOut       0  % Client    5.7
```

8.1.3 Upgrading NIM environment to Gbit Ethernet

Now that we have a representative workload, we can add the gigabit ethernet adapter and rerun the benchmark workloads. This will give us an idea of what performance increase we may get in the actual workload.

Running the NIM script for write I/O resulted in a time of 260.40 seconds for a throughput of 11.5 MB/second. The execution is identical to Example 8-5 on page 508.

Output from the **topas** command was captured and is shown in Example 8-9

Example 8-9 The topas output from write I/O benchmark script with gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Tue Oct 26 11:17:07 2004   Interval:  2        Cswitch   14586  Readch       0
                                               Syscall      64  Writech    148
Kernel   63.5   |##################          |  Reads         0  Rawin        0
User      0.0   |                            |  Writes        0  Ttyout     148
Wait      0.0   |                            |  Forks         0  Igets        0
Idle     36.5   |###########                 |  Execs         0  Namei        0
                                               Runqueue     2.5  Dirblk       0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue    0.0
en0   72998.9  99075.0  9752.0   141.4K 1247.4
en1       0.6      8.0     1.0      0.8     0.3  PAGING           MEMORY
lo0       0.0      0.0     0.0      0.0     0.0  Faults         0  Real,MB    4095
                                                 Steals         0  % Comp     17.3
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn         0  % Noncomp  50.5
hdisk1  100.0  42304.0   661.0     0.0 84608.0  PgspOut        0  % Client   50.7
hdisk3    0.0      0.0     0.0     0.0     0.0  PageIn         0
```

The network throughput and CPU utilization has increased, but hdisk1 (the actual disk used for NIM repository) is now 100% busy, and is the current bottleneck. Although we increased the throughput of the networking component ten fold, our benchmark did not see the same amount of performance improvement.

This is typical of the performance tuning process. Increasing one resource often moves the bottleneck to a different component of the system.

Running the NIM script for read I/O resulted in a time of 175.87 seconds for a throughput of 17.1 MB/second. The execution is identical to Example 8-7 on page 509.

Output from the **topas** command was captured and is shown in Example 8-10

Example 8-10 topas output from read I/O benchmark script with gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
```

```
Tue Oct 26 11:23:31 2004   Interval:  2          Cswitch     919  Readch        0
                                                 Syscall      59  Writech     162
Kernel     5.0   |##                          |  Reads         0  Rawin        0
User       0.0   |                            |  Writes        0  Ttyout      162
Wait      95.0   |########################### |  Forks         0  Igets        0
Idle       0.0   |                            |  Execs         0  Namei        0
                                                 Runqueue     0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out   Waitqueue    8.0
en0   13896.1   9133.0   969.0   527.7 27264.4
en1       0.6      8.0     1.0     0.8     0.4  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults       526  Real,MB     4095
                                                 Steals         0  % Comp      17.2
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn         0  % Noncomp    9.5
hdisk1  100.0  13556.0   372.5 27112.0     0.0  PgspOut        0  % Client     9.7
hdisk3    0.0      0.0     0.0     0.0     0.0  PageIn      3388
```

NIM workload results with gigabit ethernet

Utilizing the benchmark script, we did see a performance improvement in both read I/O and write I/O. Now we want to see what kind of improvement we get when the NIM server is handling NIM client requests. We collected topas output as well as iostat output while three NIM client installs were processing simultaneously.

Example 8-11 NIM server topas output with gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Tue Oct 26 08:36:13 2004   Interval:  2        Cswitch    2143  Readch        0
                                               Syscall     268  Writech     500
Kernel    15.0   |#####                       |  Reads         0  Rawin        0
User       0.0   |                            |  Writes        1  Ttyout      326
Wait      85.0   |#######################     |  Forks         0  Igets        0
Idle       0.0   |                            |  Execs         0  Namei        2
                                               Runqueue     0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out   Waitqueue    8.0
en0   28098.3  18455.0  1901.0  1027.2 55169.4K
en1       0.5      4.0     2.0     0.2     0.7  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults       532  Real,MB     4095
                                                 Steals         0  % Comp      15.2
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn         0  % Noncomp   38.8
hdisk1  100.0  27352.0   156.0 54704.0     0.0  PgspOut        0  % Client    38.9
```

The output is similar to what was seen during the benchmark script runs. Network throughput has increased, but hdisk1 has become completely busy.

During the NIM client installation process, we collected **iostat** command output, using **iostat 5 >> iostat.out**. This started **iostat** collecting statistics every 5

seconds and saved the output to the file iostat.out. Once the client installs completed the command was stopped by pressing Ctrl-C.

Example 8-12 NIM server iostat output with gigabit ethernet

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.2        275.4               0.0     16.2       0.2      83.6

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       3.2       0.8          0        16
hdisk1         100.0     27371.2     167.8     136856         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        146.2               0.3     14.6       0.0      85.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       4.0       1.2          0        20
hdisk1         100.0     26365.6     181.0     131828         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.0        224.4               0.2     10.3       0.4      89.1

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.2       4.0       0.8          0        20
hdisk1         100.0     22391.2     186.4     111956         0
```

8.1.4 Upgrading the disk storage

Using a single locally attached SCSI drive poses many problems. One of the most important issues is that there is no data redundancy. Secondly, the performance is not sufficient to handle the client requests.

We have decided to use an IBM storage subsystem DS4500 (FAStT 900) to move the NIM server resources to. We have assigned two LUNs to the NIM server. The two LUNs reside on separate RAID groups. The RAID groups are comprised of 7 disks each and are configured in RAID5, with a stripe size of 64kB.

There are many ways of configuring these two DS4500 LUNs. Since the disk subsystem is handling the data redundancy, we do not need to use LVM mirroring. Our two main choices at this point are either a spread, or a striped logical volume:

– A spread logical volume also known as course striping, alternates data between the hdisks in a volume group on a physical partition level (PP).

– A striped logical volume alternates data between hdisks on a finer basis. With a striped logical volume, you can specify a stripe size from 4KB-128KB (must be a power of two).

We decided to use a JFS2 file system, so we also have additional choices on how to create the JFS2 log (in-line or on a separate logical volume).

Configuring the LVM

AIX 5.3 has the device drivers and disk type pre-loaded for DS4500 disk devices. After configuring the DS4500 storage and assigning the LUNs to the server, the command **cfgmgr** detects the new storage and configures it to the system. After the disk is configured to the system, the disk need to assigned to a volume group. Once assigned to a volume group, the disk is automatically split into physical partitions. These physical partitions can then be made into logical volumes and the filesystem configured on the logical volumes.

To start the process we create a new volume group with the DS4500 disk devices. Example 8-13 we first list the disks available to the system with the **lsdev** command, then define the volume group with **mkvg**. Finally we check the characteristics of the volume group with **lsvg**.

Example 8-13 Creating a new volume group

```
# lsdev -Cc disk
hdisk0 Available 3s-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 5M-08-00-8,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk3 Available 41-08-02     MPIO Other FC SCSI Disk Drive
hdisk4 Available 41-08-02     1742-900 (900) Disk Array Device
hdisk5 Available 4Q-08-02     1742-900 (900) Disk Array Device

# mkvg -y ds4500vg hdisk4 hdisk5
ds4500vg

# lsvg ds4500vg
VOLUME GROUP:       ds4500vg                VG IDENTIFIER:
0022be2a00004c00000000ffd6b94f26
VG STATE:           active                  PP SIZE:        32 megabyte(s)
VG PERMISSION:      read/write              TOTAL PPs:      1022 (32704
megabytes)
MAX LVs:            256                     FREE PPs:       1022 (32704
megabytes)
LVs:                0                       USED PPs:       0 (0 megabytes)
OPEN LVs:           0                       QUORUM:         2
TOTAL PVs:          2                       VG DESCRIPTORS: 3
STALE PVs:          0                       STALE PPs:      0
ACTIVE PVs:         2                       AUTO ON:        yes
MAX PPs per VG:     32512
```

```
MAX PPs per PV:     1016                 MAX PVs:        32
LTG size (Dynamic): 1024 kilobyte(s)     AUTO SYNC:      no
HOT SPARE:          no                   BB POLICY:      relocatable
```

The volume group contains two physical volumes, with a physical partition (PP) size of 32 megabytes. There are a total of 1022 PP, and the volume group has a logical track group (LTG) of 1024 kilobytes.

Spread versus striped logical volume

We cannot be sure up front which type of logical volume will give us the best performance. To help determine which type to use, we will create both types of logical volumes and run the benchmark script.

Example 8-14 Creating stripe and spread logical volumes

```
# mklv -y'spreadlv' -t'jfs2' -e'x' ds4500vg 120
spreadlv
# mklv -y'stripelv' -t'jfs2' '-S64K' ds4500vg 120 hdisk4 hdisk5
stripelv
```

Now that the logical volumes are created, we can create the jfs2 filesystems on these logical volumes.

Example 8-15 Create and mount filesystems on previously defined logical volumes

```
# mklv -y'spreadlv' -t'jfs2' -e'x' ds4500vg 120
spreadlv
# mklv -y'stripelv' -t'jfs2' '-S64K' ds4500vg 120 hdisk4 hdisk5
stripelv
# crfs -v jfs2 -d'spreadlv' -m'/spreadfs' -A'No' -p'rw' -a agblksize='4096'
File system created successfully.
3931836 kilobytes total disk space.
New File System size is 7864320
# crfs -v jfs2 -d'stripelv' -m'/stripefs' -A'No' -p'rw' -a agblksize='4096'
File system created successfully.
3931836 kilobytes total disk space.
New File System size is 7864320
# mount /spreadfs
# mount /stripefs
```

With the filesystems mounted, we need to export the filesystems and mount them on the NIM clients like we did in Example 8-2 on page 507.

Example 8-16 Exporting filesystems for benchmark testing

```
# exportfs -i -o root=glpar2:glpar3:glpar4 /spreadfs
```

```
# exportfs -i -o root=glpar2:glpar3:glpar4 /stripefs
```

Benchmarking spread file system

With the file systems exported, we can mount one of them and run the benchmark.

Example 8-17 Mounting spreadfs on NIM clients

```
# for i in 2 3 4; do
> rsh glpar$i "mount glpar1:/spreadfs /mnt"
> done
```

The hostname *glpar2* is the IP label (as associated in the /etc/hosts file) for LPAR2 Gbit Ethernet interface, and so on (see the IP labels assignment in Figure 8-1 on page 504).

With the spreadfs file system NFS mounted on each of the NIM clients, we run the same script from Example 8-3 on page 507.

Example 8-18 Script for NIM write I/O benchmark - gigabit and DS4500

```
# timex ./writenim.sh
8000+0 records in.
8000+0 records out.
8000+0 records in
8000+0 records out
8000+0 records in.
8000+0 records out.

real 44.99
user 0.02
sys  0.01
```

With both gigabit ethernet and the DS4500 disk subsystem, the write throughput has increased dramatically to 66.7 MB/second (3000 MB / 44.99 second).

In Example 8-19 we show the full topas screen output as there is enough load for the other values to be of interest.

Example 8-19 The topas output for NIM write benchmark - DS4500 spread and gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Wed Oct 27 08:59:00 2004   Interval:  2        Cswitch   12640  Readch        0
                                               Syscall      61  Writech     163
Kernel   92.0   |########################  |  Reads          0  Rawin         0
User      0.0   |                          |  Writes         0  Ttyout      163
Wait      0.0   |                          |  Forks          0  Igets         0
Idle      8.0   |###                       |  Execs          0  Namei         0
```

```
                                                  Runqueue    0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out   Waitqueue   0.0
en0     110.8K   151.0K13837.0   219.8K 1861.7
en1       0.3      3.0     1.0     0.1     0.4  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults        0  Real,MB    4095
                                                 Steals        0  % Comp     17.2
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp  50.4
hdisk5  100.0  54720.0    53.5     0.0   106.9K PgspOut       0  % Client   50.4
hdisk4   84.5  59408.0   140.0     0.0   116.0K PageIn        0
hdisk2    0.0      0.0     0.0     0.0     0.0  PageOut   28542  PAGING SPACE
                                                 Sios      27255  Size,MB     512
Name            PID  CPU%  PgSp Owner                           % Used      1.0
nfsd         409606  50.1   0.8 root            NFS (calls/sec)  % Free     98.9
j2pg         147544   0.0   0.2 root            ServerV2       0
topas        307366   0.0   1.2 root            ClientV2       0   Press:
gil           73764   0.0   0.1 root            ServerV3    3412   "h" for help
pilegc        61470   0.0   0.2 root            ClientV3       0   "q" to quit
```

The topas output shows that some of the system resources are being fully utilized. This is a good thing. The CPU is only 8 percent idle, but is showing zero wait time. The new disks are fully utilized, but appear to be slightly imbalanced. And the Gbit Ethernet card throughput (one direction) is close to its maximum of 100 MB/second.

Now we run the read I/O benchmark, making sure to flush the caches.

Example 8-20 Script for NIM read I/O benchmark - DS4500 spread and gigabit

```
# umount /spreadfs
# mount /spreadfs
# for i in 2 3 4 ; do
> rsh glpar$i umount /mnt
> rsh glpar$i mount glpar1:/spreadfs /mnt
> done
# timex ./readnim.sh
8000+0 records in.
8000+0 records out.
8000+0 records in.
8000+0 records out.
8000+0 records in
8000+0 records out

real 30.20
user 0.02
sys  0.00
```

With both gigabit ethernet and the DS4500 disk subsystem, the read throughput has increased dramatically to 99.3 MB/second (3000 MB / 30.20 second).

In Example 8-21 on page 517 we show the full topas screen output as there is enough load for the other values to be of interest.

Example 8-21 The topas output for NIM read benchmark - DS4500 spread and gigabit ethernet

```
Topas Monitor for host:    p690_lpar1           EVENTS/QUEUES    FILE/TTY
Wed Oct 27 09:04:54 2004   Interval:  2         Cswitch    6905  Readch        0
                                                Syscall      53  Writech     185
                                                Reads         0  Rawin         0
Kernel    52.2   |###############             |  Writes        0  Ttyout      185
User       0.0   |                            |  Forks         0  Igets        0
Wait      24.0   |#######                     |  Execs         0  Namei        0
Idle      23.8   |#######                     |  Runqueue    2.5  Dirblk       0
                                                Waitqueue    0.5
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out
en0     102.5K 69925.0  7118.0  4028.4   201.1K
en1       0.3      4.0     1.0     0.2      0.4  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0      0.0  Faults       648 Real,MB     4095
                                                Steals         0 % Comp      17.2
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn         0 % Noncomp   38.9
hdisk4   56.5  53248.0  1524.0   104.0K     0.0  PgspOut        0 % Client    39.0
hdisk5   43.0  49152.0  1360.0 98304.0      0.0  PageIn     25584
hdisk2    0.0      0.0     0.0     0.0      0.0  PageOut        0 PAGING SPACE
                                                Sios       25602 Size,MB      512
Name            PID  CPU%  PgSp Owner                           % Used       1.0
nfsd         409606  25.0   0.8 root             NFS (calls/sec) % Free      98.9
topas        254150   0.0   1.5 root             ServerV2       0
j2pg         147544   0.0   0.2 root             ClientV2       0   Press:
gil           73764   0.0   0.1 root             ServerV3    3198   "h" for help
aixmibd      442596   0.0   0.6 root             ClientV3       0   "q" to quit
```

Although the read throughput was higher, the resources do not report being as busy as with the write workload. The network adapter is still close to its limit at 100 MB/second.

Now that we have some numbers for a spread file system, we will run the same benchmark for the striped file system.

Benchmarking stripe file system

To prepare for running the same benchmark against the stripe file system, we need to unmount the spread file system from the NIM clients. Then the striped file system needs to mounted, and the script run.

Example 8-22 Script for NIM write I/O benchmark - gigabit and DS4500 striped

```
# for i in 2 3 4 ; do
> rsh glpar$i "umount /mnt"
> rsh glpar$i "mount glpar1:/stripefs /mnt"
```

```
> done
# timex ./writenim.sh
8000+0 records in.
8000+0 records out.
8000+0 records in.
8000+0 records out.
8000+0 records in
8000+0 records out

real 98.00
user 0.02
sys  0.00
```

The striped filesystem finished in 98 seconds giving a throughput of 30.6 MB/s (3000 MB/98 seconds). This is much slower than the spread filesystem which finished in less than half the time at 45 seconds.

This may be a good indication that in our environment we are using large sequential reads and writes (typical NIM environment).

Example 8-23 The topas output for NIM write benchmark - DS4500 stripe and gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Wed Oct 27 09:25:51 2004   Interval:  2        Cswitch    7700  Readch        0
                                               Syscall      54  Writech     142
Kernel   94.1   |########################## |  Reads         0  Rawin         0
User      0.0   |                           |  Writes        0  Ttyout      142
Wait      0.0   |                           |  Forks         0  Igets         0
Idle      5.9   |##                         |  Execs         0  Namei         0
                                               Runqueue   17.9  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
en0   96998.7    127.4K10872.0   183.2K 1512.8
en1       0.5      5.0     1.0     0.6     0.3  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults        0  Real,MB    4095
                                                Steals    23301  % Comp     17.2
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp  77.1
hdisk5  100.0  49019.5   140.5     4.0 95584.0  PgspOut       0  % Client   77.2
hdisk4   95.3  46992.8   782.1    20.0 91616.0  PageIn        0
hdisk2    0.0      0.0     0.0     0.0     0.0  PageOut   24054  PAGING SPACE
                                                Sios      23294  Size,MB     512
Name            PID  CPU%  PgSp Owner                          % Used      1.0
nfsd         409606  52.2   0.8 root            NFS (calls/sec)  % Free     98.9
lrud          45078  26.1   0.1 root            ServerV2       0
j2pg         147544   0.0   0.2 root            ClientV2       0   Press:
topas        286806   0.0   1.5 root            ServerV3    2976   "h" for help
gil           73764   0.0   0.1 root            ClientV3       0   "q" to quit
```

Disk Busy% increased, but KBPS for the disks is lower. The extra overhead in splitting the I/Os into 64 KB strips between the DS4500 RAID LUNs resulted in decreased write performance. Course striping from implementing a spread file system appears to outperform fine striping.

After observing the write performance, it is time to finish the benchmark comparison by running the read I/O script.

Example 8-24 Script for NIM read I/O benchmark - gigabit and DS4500 striped

```
# umount /stripefs;mount /stripefs
# for i in 2 3 4 ; do
> rsh glpar$i "umount /mnt"
> rsh glpar$i "mount glpar1:/stripefs /mnt"
> done
# timex ./readnim.sh
8000+0 records in.
8000+0 records out.
8000+0 records in.
8000+0 records out.
8000+0 records in
8000+0 records out

real 29.46
user 0.02
sys  0.00
```

Read throughput for the striped filesystem is comparable and finished less than a second quicker than the spread filesystem. With both gigabit ethernet and the DS4500 disk subsystem, the read throughput for the striped file system has increased to 101.8 MB/second (3000 MB / 29.46 second).

In Example 8-25 we show the topas screen output for the NIM read benchmark to the striped file system.

Example 8-25 The topas output for NIM read benchmark - DS4500 stripe and gigabit ethernet

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Wed Oct 27 09:32:52 2004   Interval:  2        Cswitch    7185  Readch        0
                                               Syscall      59  Writech     228
Kernel    53.5   |################           |  Reads         0  Rawin         0
User       0.0   |                           |  Writes        0  Ttyout      228
Wait      25.0   |#######                    |  Forks         0  Igets         0
Idle      21.5   |#######                    |  Execs         0  Namei         0
                                               Runqueue     0.5  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue    5.0
en0     104.4K 71323.0  7186.0  4107.7   204.8K
en1       0.5     3.0     1.0     0.6     0.5  PAGING           MEMORY
```

```
lo0       0.0       0.0      0.0      0.0      0.0  Faults       639  Real,MB    4095
                                                     Steals         0  % Comp     17.2
Disk    Busy%      KBPS      TPS KB-Read KB-Writ  PgspIn         0  % Noncomp  46.4
hdisk4   89.0  52166.0  1377.0   101.9K      0.0  PgspOut        0  % Client   46.4
hdisk5   70.0  52140.0  1361.0   101.8K      0.0  PageIn     26065
hdisk2    0.0       0.0      0.0      0.0      0.0  PageOut        0  PAGING SPACE
                                                     Sios       26060  Size,MB     512
Name            PID  CPU%  PgSp Owner                                 % Used      1.0
nfsd         409606  25.0   0.8 root              NFS (calls/sec)  % Free     98.9
topas        266346   0.0   1.5 root              ServerV2         0
j2pg         147544   0.0   0.2 root              ClientV2         0   Press:
aixmibd      442596   0.0   0.6 root              ServerV3      3257   "h" for help
rpc.lockd    385272   0.0   0.2 root              ClientV3         0   "q" to quit
```

The disk Busy% is much higher for the striped filesystem, but the load is better balanced. The rest of the resource utilization is similar to the spread file system.

8.1.5 Real workload with spread file system

Although the striped file system outperformed the spread file system for read operations, the difference was small. The difference between write throughput was significant, the spread filesystem had more than double the write throughput of the striped filesystem. Because of these results, we select the spread filesystem.

Important: Little benefit is gained from doing a similar feature twice. For example it is common knowledge that with database applications, you want to avoid double buffering. Double buffering is where both the application and the operating system use memory as cache. This consumes memory that could be used for other operations, and wastes CPU cycles doing redundant caching. Likewise with striping. The DS4500 LUNs are already striped across disks. Using LVM striping across DS4500 LUNs results in, for lack of a better term, double striping. The DS4500 controller is going to receive small stripes alternating between different disk groups. This extra striping is not benefiting the disk subsystem, and could even be causing performance degradation.

In summary, it is important to understand the workload the application is generating so as to make the most efficient use of system resources.

With the NIM resources moved to the spread filesystem, we can now monitor a real workload. For this we will simultaneously restore three NIM clients and monitor the output with **topas**, **iostat,** and **sar**.

Example 8-26 The topas output after gigabit ethernet and DS4500 storage

```
Topas Monitor for host:    p690_lpar1          EVENTS/QUEUES    FILE/TTY
Wed Oct 27 17:18:50 2004   Interval:  2        Cswitch    5126  Readch        0
                                               Syscall     218  Writech     645
Kernel   34.5   |##########                |  Reads         0  Rawin         0
User      0.2   |#                         |  Writes        1  Ttyout      189
Wait      4.5   |##                        |  Forks         0  Igets         0
Idle     61.2   |##################        |  Execs         0  Namei         2
                                               Runqueue    0.0  Dirblk        0
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out  Waitqueue   0.0
en0   73874.8  47437.0  4809.0  2651.3   141.7K
en1       0.3      5.0     1.0     0.2     0.4  PAGING           MEMORY
lo0       0.0      0.0     0.0     0.0     0.0  Faults      175  Real,MB    4095
                                               Steals        0  % Comp     18.1
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp  37.2
hdisk5   31.0  36776.0   261.0 73552.0     0.0  PgspOut       0  % Client   37.3
hdisk4   28.0  35480.0   199.0 70960.0     0.0  PageIn    17942
hdisk2    0.0      0.0     0.0     0.0     0.0  PageOut       0  PAGING SPACE
                                               Sios      18022  Size,MB     512
Name            PID  CPU%  PgSp Owner                          % Used      1.0
nfsd         409606   0.0   0.8 root            NFS (calls/sec)  % Free     98.9
topas        348170   0.0   1.2 root            ServerV2       0
sadc         393360   0.0   0.2 root            ClientV2       0   Press:
nimesis      262222   0.0   1.4 root            ServerV3    2232   "h" for help
gil           73764   0.0   0.1 root            ClientV3       0   "q" to quit
```

The three NIM clients definitely made use of the additional network and storage resources. They have not “maxed out” the available resources based on the benchmark testing.

To collect iostat information we executed the command **`iostat 5 60 >> iostat.out`**. This collected statistics every 5 seconds for 60 intervals, for a total of five minutes. We scanned through the output and Example 8-27 contains the interval where activity was the highest.

Example 8-27 The iostat output after gigabit ethernet and DS4500 storage

```
tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
          0.4         80.8                0.2     25.7       73.1       1.0

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk0           0.0       0.0      0.0          0         0
hdisk1           0.0       0.0      0.0          0         0
hdisk3           0.0       0.0      0.0          0         0
hdisk2           0.0       0.0      0.0          0         0
dac0             0.0     27161.6     97.4     135808         0
dac0-utm         0.0       0.0      0.0          0         0
dac1             0.0     25395.2     97.0     126976         0
```

```
dac1-utm          0.0       0.0       0.0          0          0
hdisk4           18.8   27161.6      97.4     135808          0
hdisk5           20.0   25395.2      97.0     126976          0
cd0               0.0       0.0       0.0          0          0
```

The dac0 and dac1 items in the Disks: column are the DS4500 controllers. If there were more disks per controller and activity on the disks, the dac0 and dac1 would show a cumulative value. As there is only one disk per controller in our configuration, the values are the same for the disk and its associated controller.

The DS4500 system is handling the I/O requests and has some throughput left over. The load is spread fairly evenly over the two DS4500 hdisks.

The sar command is another useful way of collecting disk statistics. To collect the disks statistics in Example 8-28 we executed the command **sar -d 5 60 >> sar.out**. We then scanned the output and selected the interval with the highest utilization.

Example 8-28 The sar output after gigabit ethernet and DS4500 storage

```
117:15:28     device    %busy    avque    r+w/s    Kbs/s   avwait   avserv
...(lines omitted)....
               dac0       0      0.0      145    33231      0.0      0.0
           dac0-utm       0      0.0        0        0      0.0      0.0
               dac1       0      0.0      150    32261      0.0      0.0
           dac1-utm       0      0.0        0        0      0.0      0.0
             hdisk4      24      0.0      145    33231      0.0      0.0
             hdisk5      24      0.0      150    32261      0.0      0.0
...(lines omitted)....
```

The output from **sar -d** is similar to **iostat**, but with **sar** we get some additional values that can be useful. Details on the output for **sar** command can be found in 7.2.6, “The sar -d command” on page 478.

Zero values for *avque*, *avwait*, *avserv* are desirable. Nonzero values should be subject for further investigation and tuning.

8.1.6 Summary

Performance tuning is an iterative process. This case study went through a couple of basic iterations of the process. It is important to accurately identify system bottlenecks and then make the correct choice as to add resources, tune resources, or leave it alone. Performance tuning on a production system is risky. Having system backups and system documentation can go a long way in recovering from bad tuning choices. An understanding of the actual system

workload and the effects of tuning commands is the important first step in the performance tuning process.

8.2 POWER5 case study

This chapter provides POWER5 specific performance issue. We provides the monitoring of the CPU performance of POWER5-based system using a simple case scenario.

The performance described in this chapter is a sample in a certain specific environment. The actual throughput and performance are affected by various factors, such as hardware configuration, partition configuration, and the characteristic of a process. User needs to evaluate the performance according to each environment. There is no assurance that the performance described in this chapter applicable to other similar environments.

8.2.1 POWER5 introduction

We described the outline of Micro-Partitioning and simultaneous multi-threading (SMT) which are the feature of POWER5 in 4.1.2, “Performance considerations with POWER5-based systems” on page 172.

With these new technologies, the calculation of the performance statistics has changed. In previous version of AIX (AIX 5L V5.1 and V5.2), the performance statistics was calculated using each processor usage. In traditional processor utilization, data collection is sample based. There are 100 samples per second sorted into four categories: %usr, %sys, %wait, and %idle.

In a shared-partition environment, we have to consider that there is unused time slice in each entitled processor capacity. When a virtual processor or SMT thread becomes idle, it is able to cede processor cycle to Hypervisor, and then the Hypervisor can dispatch unused processor cycles for other work.

In order to collect CPU utilization at a processor thread level (in an SMT environment), in the POWER5 architecture has implemented a new register, called the Processor Utilization Resource Register (PURR). Each thread has its own PURR. The units are the same as the timebase register and the sum of the PURR values for both threads is equal to timebase register.

For more information about calculation using PURR, refer to the redbook *Advanced POWER Virtualization on IBM @server p5 Servers Architecture and Performance Considerations*, SG24-5768.

8.2.2 High CPU

In our test environment we have simulated a CPU load to verify the output of various AIX performance monitoring commands.

LPAR configuration

To verify the LPAR configuration, use **lparstat -i** command, as shown in Example 8-29. The test described in “Monitoring CPU utilization” on page 525 is performed using this configuration.

Example 8-29 Verifying the LPAR configuration

```
r33n05:/ # lparstat -i
Node Name                                  : r33n05
Partition Name                             : r33n05
Partition Number                           : 3
Type                                       : Shared
Mode                                       : Uncapped
Entitled Capacity                          : 0.50
Partition Group-ID                         : 32771
Shared Pool ID                             : 0
Online Virtual CPUs                        : 1
Maximum Virtual CPUs                       : 40
Minimum Virtual CPUs                       : 1
Online Memory                              : 7168 MB
Maximum Memory                             : 15360 MB
Minimum Memory                             : 1024 MB
Variable Capacity Weight                   : 128
Minimum Capacity                           : 0.10
Maximum Capacity                           : 4.00
Capacity Increment                         : 0.01
Maximum Dispatch Latency                   : 9999999
Maximum Physical CPUs in system            : 4
Active Physical CPUs in system             : 4
Active CPUs in Pool                        : -
Unallocated Capacity                       : 0.00
Physical CPU Percentage                    : 50.00%
Unallocated Weight                         : 0
r33n05:/ #
```

To view the current simultaneous multi-threading (SMT) mode settings, use the **smtctl** command, as shown in Example 8-30. In this example, SMT mode is disabled.

Example 8-30 Displaying the current SMT mode setting

```
r33n05:/ # smtctl
```

```
This system is SMT capable.

SMT is currently disabled.

SMT boot mode is set to enabled.


Processor 1 has 1 SMT threads
SMT thread 0 is bound with processor 1
r33n05:/ #
```

Monitoring CPU utilization

AIX 5L V5.3 provides several commands to monitor the CPU utilization. Also there are new commands for displaying and performing dynamic configuration changes. These new commands are **lparstat** and **smtctl**.

From the output of Example 8-29 on page 524, and Example 8-30 on page 524, we get the following CPU related information:

- Entitled capacity is 0.5
- The number of Virtual CPU is 1
- SMT mode setting is disabled

In our scenario, we change SMT mode setting from disable to enable while monitoring commands are running. To turn the SMT on, use the following command:

```
smtctl -m on -w now
```

For more information about this command, refer to “The smtctl command” on page 276.

Example 8-31 on page 526 shows how to display the changes in CPU utilization using the **lparstat** command. Since this partition is a shared-partition, following statistics are displayed to report shared physical and logical processor utilization and entitled capacity utilization:

physc	Shows the number of physical processors consumed.
%entc	Shows the percentage of the entitled capacity consumed.
lbusy	Shows the percentage of logical processor(s) utilization that occurred while executing at the user and system level.

Because the SMT mode turned on, the number of logical CPUs was also changed from one to two. And after the configuration was changed, we observed

that the value of *%use* and *%sys* slightly decreased compared to previous case (SMT mode off).

Example 8-31 Statistics information of the lparstat command

```
System configuration: type=Shared mode=Uncapped smt=Off lcpu=1 mem=7168 ent=0.50

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint
-----  ----  -----  ----- ----- ----- ------   ---  ---- -----
 15.9  84.1    0.0    0.0  1.00 200.2  100.0     -   100     0
 15.5  84.5    0.0    0.0  1.00 199.9  100.0     -   100     0
 15.5  84.5    0.0    0.0  1.00 199.9  100.0     -   100     1
 15.4  84.6    0.0    0.0  1.00 200.0  100.0     -   100     0
 15.4  84.6    0.0    0.0  1.00 200.0  100.0     -   100     0
 15.4  84.6    0.0    0.0  1.00 200.0  100.0     -   100     0
System configuration changed. The current iteration values may be inaccurate.
  9.1  64.9    0.3   25.7  1.00 200.1   96.4     - -1036941459     2

System configuration: type=Shared mode=Uncapped smt=On lcpu=2 mem=7168 ent=0.50

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint
-----  ----  -----  ----- ----- ----- ------   ---  ---- -----
 14.5  68.5    0.0   17.0  1.00 200.0   97.3     -   200     0
 14.3  68.8    0.0   16.9  1.00 200.0   97.4     -   200     0
 14.4  68.9    0.0   16.7  1.00 200.0   97.9     -   200     0
 14.4  68.9    0.0   16.7  1.00 200.0   95.4     -   200     0
 14.4  68.9    0.0   16.7  1.00 200.0   99.0     -   200     0
... lines omitted ...
```

The **vmstat** command can also provide system-wide CPU utilization. Example 8-32 shows the changes in CPU utilization using the **vmstat** command.

Example 8-32 Statistics information of the vmstat command

```
System configuration: lcpu=1 mem=7168MB ent=0

kthr    memory              page              faults              cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 3  0 183852 1635807   0   0   0   0    0   0   2 330610 206349 16 84  0  0  1.00 200.4
 3  0 183856 1635803   0   0   0   0    0   0   0 330201 206398 16 84  0  0  1.00 200.0
 3  0 183856 1635803   0   0   0   0    0   0   1 331949 207594 16 84  0  0  1.00 200.0
 3  0 183856 1635803   0   0   0   0    0   0   0 331351 207210 15 85  0  0  1.00 200.0
 3  0 183856 1635803   0   0   0   0    0   0   0 331128 207033 15 85  0  0  1.00 200.0
 3  0 183856 1635803   0   0   0   0    0   0   0 332095 207415 15 85  0  0  1.00 200.0
System configuration changed. The current iteration values may be inaccurate.
 2  0 184153 1635506   0   0   0   0    0   0  10 229907 191740  9 65 26  0  1.00 200.3

System configuration: lcpu=2 mem=7168MB ent=0
```

```
kthr    memory              page              faults           cpu
----- ----------- ------------------------ ------------ -----------------------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa    pc    ec
 3  0 184153 1635506   0   0   0    0    0   0   0 430302 370946 14 69 17  0  0.99 198.1
 3  0 184153 1635506   0   0   0    0    0   0   0 422901 365423 14 69 17  0  1.00 200.1
 3  0 184154 1635505   0   0   0    0    0   0   0 424881 371246 14 69 17  0  1.00 200.1
 3  0 184154 1635505   0   0   0    0    0   0   0 418912 371810 14 69 17  0  1.00 200.1
 3  0 184154 1635505   0   0   0    0    0   0   0 424568 367992 14 69 17  0  1.00 200.1
 3  0 183946 1635712   0   0   0    0    0   0   2 424401 364435 14 69 17  0  1.00 200.1
 3  0 183946 1635712   0   0   0    0    0   0   6 421948 364210 14 69 17  0  1.00 200.1
 3  0 183845 1635812   0   0   0    0    0   0  12 424660 367271 14 69 17  0  1.00 200.1
 3  0 183845 1635812   0   0   0    0    0   0   1 423924 364879 14 69 17  0  1.00 200.1
 3  0 183845 1635812   0   0   0    0    0   0   5 425998 367266 14 69 17  0  1.00 200.1
 3  0 183845 1635812   0   0   0    0    0   0   3 422701 367368 14 69 17  0  1.00 200.1
... lines omitted ...
```

The **mpstat** command is useful to investigate every logical CPU utilization. Example 8-33 shows how to of display the changes in CPU utilization, using the **mpstat** command. Because the SMT mode was turned on, the number of logical CPUs also changed from one to two. Before the SMT was turned on, only the lines for “CPU0” and “ALL” were reported. And after the configuration was changed, the line for “CPU1” was added to the report. With regard to each logical CPU, we can see that %usr value of CPU1 increased a little, and %sys value of CPU1 decreased, and %usr and %sys of CPU2 also decreased.

Example 8-33 Statistics information of the mpstat command

```
System configuration: lcpu=1 ent=0.5

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0    0    0    0 202317 207605 6398    4    0 100 330610 16 84  0  0 1.00 100.1  100
ALL    0    0    0 202317 207605 6398    4    0 100 330610 16 84  0  0 1.00 200.1  100
------------------------------------------------------------------------------
  0    0    0    0 201076 206396 6376    4    0 100 330201 16 84  0  0 1.00 199.9  100
ALL    0    0    0 201076 206396 6376    4    0 100 330201 16 84  0  0 1.00 199.9  100
------------------------------------------------------------------------------
  0    0    0    0 202194 207607 6499    4    0 100 331949 16 84  0  0 1.00 199.9  100
ALL    0    0    0 202194 207607 6499    4    0 100 331949 16 84  0  0 1.00 199.9  100
------------------------------------------------------------------------------
  0    0    0    0 201824 207213 6475    4    0 100 331351 15 85  0  0 1.00 199.9  100
ALL    0    0    0 201824 207213 6475    4    0 100 331351 15 85  0  0 1.00 199.9  100
------------------------------------------------------------------------------
  0    0    0    0 201699 207058 6438    4    0 100 331128 15 85  0  0 1.00 199.9  100
ALL    0    0    0 201699 207058 6438    4    0 100 331128 15 85  0  0 1.00 199.9  100
------------------------------------------------------------------------------
  0    0    0    0 202242 207457 6288    4    0 100 332095 15 85  0  0 1.00 200.0  100
ALL    0    0    0 202242 207457 6288    4    0 100 332095 15 85  0  0 1.00 200.0  100
```

```
--------------------------------------------------------------------------------

System configuration changed. The current iteration values may be inaccurate.
  0    0    0    0 85317 95448 10284    4 13806 100 147128 11 88  0  1 0.58 116.0  101
  1    0    0    0 52226 120148 67918    1 13817 100 84520  7 33  0 60 0.00  0.8 -1036941560
  U    -    -    -    -    -    -    -    -   -    -  -  - 20 78 0.49 98.0    -
ALL    0    0    0 137543 215596 78202    5 27623 100 231648  0  1 20 79 0.58 116.9 -1036941459
--------------------------------------------------------------------------------

System configuration: lcpu=2 ent=0.5

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0    0    0    0 145401 160058 14781    2 28249 100 247053 17 81  0  2 0.49 98.8  100
  1    0    0    0 111684 260265 148616    1 28251 100 179047 12 57  0 31 0.51 101.2  100
ALL    0    0    0 257085 420323 163397    3 56500 100 426100 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 145380 156860 11656    2 27441 100 244538 17 82  0  2 0.49 98.9  100
  1    0    0    0 110224 259681 149465    1 27442 100 178412 12 56  0 32 0.51 101.1  100
ALL    0    0    0 255604 416541 161121    3 54883 100 422950 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 144738 161093 16525    2 25027 100 246852 17 81  0  2 0.49 98.9  100
  1    0    0    0 112291 256378 144082    1 25028 100 177993 12 57  0 31 0.51 101.0  100
ALL    0    0    0 257029 417471 160607    3 50055 100 424845 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 142620 165471 23010    2 25380 100 238842 17 80  0  3 0.49 99.0  100
  1    0    0    0 112281 253830 141552    2 25381 100 180098 12 58  0 30 0.50 101.0  100
ALL    0    0    0 254901 419301 164562    4 50761 100 418940 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 146475 154862 8574    2 25786 100 247945 17 82  0  1 0.49 98.9  100
  1    0    0    0 110335 260850 150502    2 25786 100 176635 12 56  0 32 0.51 101.1  100
ALL    0    0    0 256810 415712 159076    4 51572 100 424580 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 146874 153136 6442    1 27873 100 247752 17 82  0  1 0.49 98.8  100
  1    0    0    0 109546 263523 153952    2 27874 100 176579 12 56  0 32 0.51 101.2  100
ALL    0    0    0 256420 416659 160394    3 55747 100 424331 14 69  0 17 1.00 200.0  200
--------------------------------------------------------------------------------
  0    0    0    0 144702 153055 8546    2 27174 100 245862 17 82  0  1 0.49 98.7  100
  1    0    0    0 109726 261928 152198    2 27176 100 176046 12 56  0 32 0.51 101.2  100
ALL    0    0    0 254428 414983 160744    4 54350 100 421908 14 69  0 17 1.00 200.0  200
... lines omitted ...
```

The **sar** command with the **-P** flag can also provide utilization for every logical CPU. Example 8-34 shows the changes for logical CPU utilization using the **sar** command.

Example 8-34 Statistics information of the sar command

```
AIX r33n05 3 5 00C3E3CC4C00    10/26/04
```

```
System configuration: lcpu=1 ent=0.50

18:14:43 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:14:44  0       17      83       0       0    1.00   100.0
          -       17      83       0       0    1.00   100.0
18:14:45  0       16      84       0       0    1.00   100.0
          -       16      84       0       0    1.00   100.0
18:14:46  0       16      84       0       0    0.99   100.0
          -       16      84       0       0    0.99   100.0
18:14:47  0       15      85       0       0    1.00   100.0
          -       15      85       0       0    1.00   100.0
18:14:48  0       15      85       0       0    1.00   100.0
          -       15      85       0       0    1.00   100.0
18:14:49  0       15      85       0       0    1.00   100.0
          -       15      85       0       0    1.00   100.0

Average   0       16      84       0       0    1.00   100.0
          -       16      84       0       0    NaNQ    NaNQ

System configuration changed. The current iteration values may be inaccurate.

18:14:50  0       11      89       0       0    0.81   161.2
          1        1       4       0      94    0.00     0.4
          -        9      72       0      18    0.81   161.6

System configuration: lcpu=2 ent=0.50

18:14:51 cpu    %usr    %sys    %wio   %idle   physc   %entc
18:14:51  0       17      81       0       2    0.49    49.4
          1       12      56       0      31    0.51    50.6
          -       15      68       0      17    1.00   100.0
18:14:52  0       17      80       0       3    0.50    49.5
          1       12      57       0      31    0.51    50.5
          -       14      69       0      17    1.00   100.0
18:14:53  0       17      82       0       1    0.49    49.4
          1       12      56       0      32    0.51    50.6
          -       14      69       0      17    1.00   100.0
18:14:54  0       16      79       0       4    0.49    49.5
          1       12      59       0      29    0.50    50.5
          -       14      69       0      17    0.99   100.0
18:14:55  0       17      82       0       1    0.50    49.5
          1       12      56       0      32    0.51    50.5
          -       14      69       0      17    1.00   100.0
18:14:56  0       17      82       0       1    0.49    49.4
          1       12      56       0      32    0.51    50.6
          -       14      69       0      17    1.00   100.0
18:14:57  0       17      82       0       1    0.49    49.4
          1       12      56       0      32    0.51    50.6
          -       14      69       0      17    1.00   100.0
```

```
18:14:58  0       17      82       0       1    0.49    49.4
          1       12      56       0      32    0.51    50.6
          -       15      69       0      17    1.00   100.0
... lines omitted ...
```

After the configuration was changed, SMT mode is enabled and so each virtual processors was configured as 2-way logical processor. To display simultaneous multi-threading threads utilization, use the **mpstat** command with the -s flag as in Example 8-35. In this example, we run this command after the configuration was changed because the -s flag is available only in a partition with SMT enabled. In this case, both cpu0 and cpu1 are using the virtual processor about 50%.

Example 8-35 Displaying the simultaneous multi-threading threads utilization

```
System configuration: lcpu=2 ent=0.5

     Proc0
    100.05%
 cpu0    cpu1
 49.37%  50.68%
--------------------------------------------------------------------------------
-
     Proc0
     99.98%
 cpu0    cpu1
 49.32%  50.66%
--------------------------------------------------------------------------------
-
     Proc0
     99.98%
 cpu0    cpu1
 49.31%  50.67%
--------------------------------------------------------------------------------
-
     Proc0
     99.98%
 cpu0    cpu1
 49.37%  50.61%
--------------------------------------------------------------------------------
-
     Proc0
     99.98%
 cpu0    cpu1
 49.32%  50.66%
--------------------------------------------------------------------------------
-
     Proc0
     99.98%
```

```
cpu0    cpu1
49.37%  50.61%

... lines omitted...
```

8.2.3 Evaluation

In this case scenario, the CPU utilization was changed by changing SMT mode from disable to enable.

With regard to CPU, POWER5-based systems support the following dynamic configuration changes:

- Remove, move, and add entitled shared processor capacity
- Add and remove virtual processors
- Change between capped and uncapped processing capacity
- Change the weight of an uncapped partition

Change to these parameter values also effects overall system performance. For more information, refer to the redbook *Advanced POWER Virtualization on IBM @server p5 Servers Architecture and Performance Considerations*, SG24-5768.

9

Miscellaneous tools

In the first section of this chapter we present the Workload Manager (WLM) feature on AIX which provides a set of tools that assist in gleaning useful performance statistics and provides the administrator an efficient mechanism to control allocation of resources to processes.

The second section introduces the Partition Load Manager (PLM) software which is part of the Advanced POWER Virtualization feature and helps customers to maximize the utilization of processor and memory resources of DLPAR capable logical partitions running AIX 5L on pSeries servers.

In the third section, we present a short comparison between two techniques of vertical server consolidation: Workload Manager and partitioning (with Partition Load Manager - PLM).

The fourth section of this chapter introduces the Resource Monitoring and Control subsystem and a short overview about how to use the RMC for monitoring system performance.

9.1 Workload manager monitoring (WLM)

This section introduces the WLM as a monitoring tool for performance related problems in AIX 5L. WLM is a complex tool which can be used, beside performance monitoring, for gathering accounting data, and also for managing the load on a standalone system.

In conjunction with dynamic LPAR, WLM may also be used as a resource provisioning tool in a partitioned environment.

For more details about auditing and load management functions of WLM, refer to these publications:

- *AIX 5L Workload Manager (WLM)*, SG24-5977
- *Accounting and Auditing on AIX*, SG24-6396

9.1.1 Overview

It is imperative for businesses today to understand the behavior of applications under workload and react to changes in workload; to ensure better response times and the optimum utilization of resources; to guarantee the uptime of servers in accordance with service level agreements, and effectively gather statistics on resource usage.

It is becoming increasingly vital for system administrators today to be able to determine and control resource usage by processes. There is a need to monitor how the resources on a system are being used, and to implement effective mechanisms to efficiently balance the allocation of resources among the processes.

The WLM feature on AIX provides a set of tools that assist in gleaning useful performance statistics and provide the administrator an efficient mechanism to control allocation of resources to processes.

WLM is primarily intended for use with large systems running multiple applications, databases and transaction processing systems, where workloads are combined into a single large system (“vertical” server consolidation).

Workload Manager provides the flexibility for dividing system resources between jobs without having to partition the system (where reinstallation and reconfiguration are required). WLM also provides an effective means of isolation between jobs with very different system behaviors.

More and more organizations are charging user communities for computing services being used. WLM can be effectively used in conjunction with the AIX

accounting subsystem to profile accounting information for WLM classes. These resource usage statistics can be used for billing users for the system resources.

9.1.2 WLM concepts

This section introduces the WLM terminology used throughout this chapter.

Definitions

The functionality of WLM is based on entities called *classes*. System administrators can define classes with a set of attributes and resource limits and assign processes to a class based on assignment rules for the class. AIX WLM provides the ability to control allocation of resources (CPU, physical memory and bandwidth) to these classes.

Processes are placed in these classes based on *users, groups, application paths, process types, or application tags*. These attributes form the assignment rules for classification of processes.

User ID The user name owning a process can be used to classify the process to a class. The user ids are available in the /etc/password file or the NIS. The **`smitty lsuser`** command will list the users on the system

Group The group name of a process can be used to classify the process to a class. The group names are available in the /etc/group file or the NIS. The **`smitty lsgroup`** command will list the group name on the system

Application path The complete path name of the binary running the application.

Process types Process type attributes specifying if the process is 32-bit or 64-bit can be used to determine the class for a process.

Application tag An attribute set by the WLM API to enable classification for different instances of the same binary application.

Resource usage can be monitored and controlled at the class level. As the resource limits are set and the resource utilization regulated for each class, applications are prevented from interfering with each other when sharing a single server.

Web servers, databases, and batch programs executing low priority tasks in the background can be grouped into separate distinct classes.

Class hierarchy

A hierarchy of classes can be specified and processes automatically assigned to these classes by their characteristics, and manually placed in the classes based on simple rules.

The class hierarchy with two levels can be set up depending upon the needs of the organization by defining *superclasses and subclasses.*

SuperClass A superclass is a class that has subclasses associated with it. No processes can belong to the superclass without also belonging to a subclass. A superclass has a set of resource limitation values and resources target shares that determines the amount of resources that can be used by the processes that belong to the superclass

Subclass A subclass is a class associated with exactly one superclass. A subclass has resource limitation values that determines the resources that can be used by the processes used in the subclass.

WLM supports **32 superclasses** (27 user-defined and 5 predefined). Each superclass in turn can have **12 subclasses** (10 user-defined and 2 predefined).

The predefined superclasses are automatically created and are classified as:

Default As the name suggests it is the default class and all non-root processes that are not automatically assigned to a specific superclass are assigned to the default superclass.

System System superclass has all privileged (root) processes assigned to it if they are not assigned by rules to a specific class.

Shared Shared superclass receives all the memory pages that are shared by processes in more than one superclass.

Unclassified Memory pages that cannot be directly tied to any processes (and thus, to any class) at the time of the initial classification are charged to the Unclassified superclass.

Unmanaged A special superclass to which no processes are assigned. This class is used to accumulate the memory usage for all pinned pages that are not managed by WLM.

Class attributes

Class tiers Tiers define class importance relative to other classes. Ten tiers (0 through 9) can be defined to prioritize classes, with 0 being the most important and 9 least important.

Inheritance Specifies whether the child process inherits the class assignment from its parent.

Localshm Prevents memory segments belonging to one class from migrating to shared class.

Shares Numbers for each class to determine the percentage share for allocation of CPU, memory and disk I/O for the class.

Resource Set Limits the set of resources a given class has access in terms of CPUs.

9.1.3 Administering WLM

Working with WLM might seem a fairly sophisticated task, but, in fact, if you only need specific WLM functionality (like performance monitoring), it is simple enough to set up WLM and get fast results.

WLM configuration - A six step process

WLM can be set up on the system using the following six simple steps:

1. Determining the processes running on the system
2. Classification of the processes
3. Creation of WLM classes for these processes
4. Assigning the processes to pertinent classes using assignment rules
5. Verifying the classes and assignment rules
6. Starting WLM in passive mode

The central idea is to classify the processes on the system, based on certain parameters like the applications or workloads these processes belong to. Subsequently these processes can be grouped into WLM classes and each class can be monitored, and managed, separately for its resource usage.

The steps to set up WLM are detailed in “Setting up WLM” on page 538.

WLM administration tools

WLM can be administered in three different ways:

Command Line WLM can be administered using simple commands and editing a few configuration files

SMIT System Management Interface Tool - The hugely popular ASCII based AIX system administration tool provides a menu based interface to WLM commands

WebSM Web-based System Manager - graphical tool for managing AIX systems and convenient to use,

We have used SMIT in the examples throughout this chapter. For more information, check the redbook *AIX 5L Workload Manager (WLM)*, SG24-5977.

Table 9-1 on page 545 provides a list and a brief introduction to WLM commands and the WebSM tool.

Setting up WLM

This section describes the steps needed to configure the WLM on AIX.

1. Determine the processes running on the system

The first step is to check for all the processes running on the system and determine what the processes are doing and which application or workload they belong to, and decide on how to classify the processes.

The following command can be used to check for the processes on the system:

```
ps -e -o pid,tag,user,group,comm,args
```

Example 9-1 Sample output of ps -e -o pid,tag,user.group,comm,args

```
[p630n02][/]> ps -e -o pid,tag,user,group,comm,args

 4470 -                root   system sshd     /usr/sbin/sshd
 5082 -                root   system hostmibd /usr/sbin/hostmibd
 5168 -                root   system shlap    /usr/ccs/bin/shlap
 5476 -                root   system errdemon /usr/lib/errdemon
 6542 -                root     cron cron     /usr/sbin/cron
 6722 -                root   system getty    getty /dev/console console
 8010 -                root   system dtlogin  /usr/dt/bin/dtlogin -daemon
13336 -               user1    staff prog1    ./prog1 -c 1000
13592 -                root   system telnetd  telnetd -a
19750 -                root   system prog3    ./prog3 -m 2000
20056 -                root   system ksh      -ksh
23016 -                root   system sshd     sshd: root@pts/5
23882 -                root   system ksh      -ksh
24428 -               user2    staff prog2    ./prog2 -c 500
```

The processes **prog1**, **prog2** and **prog3** (bold in the above example) will be used in this section for illustration.

prog1 CPU intensive program executed by user1

prog2 CPU intensive program executed by user2

prog3 Memory intensive program

Note: The programs simulate resource utilization and have been used for illustration purposes only.

2. Classify the processes

The next step is to define your classes. In order to define which classes you need, you must know your users and their computing needs, the applications on your system, and their resource needs, and the requirements of your business (that is, which tasks are critical and which can be given lower priority.

Because WLM regulates the resource utilization among the classes, you should group the same in the same classes the applications and/or users with the same resource utilization patterns. For instance, you generally want to separate the interactive jobs that typically consume very little CPU time but require quick response time when activated from batch type jobs that, typically, are very CPU and memory intensive.

In the Example 9-1 on page 538 the **prog1**, **prog2** and **prog3** can be grouped into different classes.

3. Creating WLM classes

Once the processes have been classified, it is time to create the WLM classes for these processes. We can use the **smitty wlm** fast path to create the WLM classes.

- **smitty wlm**

Example 9-2 Smitty menu screen for WLM

```
                                Workload Manager

Move cursor to desired item and press Enter.

  Manage time-based configuration sets

  Work on alternate configurations
  Work on a set of Subclasses
  Show current focus (Configuration, Class Set)

  List all classes
  Add a class
  Change / Show Characteristics of a class
  Remove a class
  Class assignment rules
```

```
  Start/Stop/Update WLM
  Assign/Unassign processes to a class/subclass
```

- Select *"Add a Class"* from the smitty screen. A smitty screen with fields to specify the attributes of the class will be displayed
- Specify the attributes of the class. The *Inheritance* and the *Localshm* characteristics must be set to Yes. The **<tab>** key maybe use to change the values from the default No to Yes in the screen.

Inheritance means that when a process starts a subprocess it has the same class. This is useful for applications that start a lot of other processes, such as database starting connections for users from a listener type process. Localshm means that any shared memory created by a process in a class belongs to that class too. This is useful for databases that access shared memory, such as the DB2® buffer pool or Oracle SGA.

Example 9-3 Smitty menu screen for General characteristics of a class

```
                    General characteristics of a class

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Class name                                         [app1]
  Description                                        [CPU Intensive]
  Tier                                               [0]                    +#
  Resource Set                                                              +
  Inheritance                                        [Yes]                  +
  User authorized to assign its processes to this cl []                     +
  ass
  Group authorized to assign its processes to this c []                     +
  lass
  User authorized to administrate this class         []                     +
  (Superclass only)
  Group authorized to administrate this class        []                     +
  (Superclass only)
  Localshm                                           [Yes]                  +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

We have created a WLM class app1 for the prog1 process in this example. Similarly, WLM classes *app2* and *app3* have been created for the **prog2**, and **prog3** programs respectively, using the same steps as described in this section.

4. Assigning the process to a class based on assignment rules

After the creation of WLM classes, the processes have to be assigned to these classes based on some assignment rules.Select the "*Class Assignment rules*" from the initial smitty screen for the Workload Manager. A SMIT screen with operations for the WLM class rules will be displayed.

Example 9-4 SMIT menu screen for class assignment rules

```
                              Class assignment rules

Move cursor to desired item and press Enter.

  List all Rules
  Create a new Rule
  Change / Show Characteristics of a Rule
  Delete a Rule
  Attribute value groupings


F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

- Select "*Create a new Rule*" from the smitty screen. This will display a screen to specify the attributes for creating a rule of a WLM class.

Example 9-5 Creating a new rule for a WLM class

```
                                Create a new Rule

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Order of the rule                                  [1]                  #
* Class name                                          app1                +
* User                                               [-]                  +
* Group                                              [user1]              +
  Application                                        [-]
  Type                                               [-]                  +
  Tag                                                [-]
```

Example 9-6 Creating a new rule for a WLM class

```
                                Create a new Rule

Type or select values in entry fields.
```

```
Press Enter AFTER making all desired changes.

                                                  [Entry Fields]
* Order of the rule                              [1]                        #
* Class name                                      app3                      +
* User                                           [-]                        +
* Group                                          [-]                        +
  Application                                    [-]
  Type                                           [/work/app3/prog3] +
  Tag                                            [-]
```

Note: The application being classified should be a binary. In case of a script being used, the binary being invoked in the script should be entered.

5. Verifying WLM classes and assignment rules

After the creation of the WLM classes and assignment of the processes to these classes based on assignment rules, it is worthwhile to list the classes and rules for verification

- Select "*List All Classes*" from the initial smitty screen for the Workload Manager. This will display the defined WLM classes

Example 9-7 Smitty screen to list all classes

```
                                  Workload Manager

Move cursor to desired item and press Enter.

  Manage time-based configuration sets

  Work on alternate configurations
  Work on a set of Subclasses
  Show current focus (Configuration, Class Set)

  List all classes
  Add a class
  Change / Show Characteristics of a class
  Remove a class
  Class assignment rules

  Start/Stop/Update WLM
  Assign/Unassign processes to a class/subclass
```

Example 9-8 Screen output listing the WLM classes

```
                                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

System
Default
Shared
app1
app2
app3
```

The default super classes System, Default and Shared are listed along with the sample classes we have created, i.e., app1, app2 and app3.

- Select *"List all Rules"* from the initial smitty screen for Class assignment rules. This will display the assignment rules defined for WLM classes.

Example 9-9 Smitty screen for Class assignment rules

```
                                Class assignment rules

Move cursor to desired item and press Enter.

  List all Rules
  Create a new Rule
  Change / Show Characteristics of a Rule
  Delete a Rule
  Attribute value groupings
```

Example 9-10 Screen output listing class assignment rules

```
                                      COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

 #  Class     User      Group     Application             Type      Tag
001 app3      -         -         /work/app3/prog3        -         -
002 app2      user2     -         -                       -         -
003 app1      user1     -         -                       -         -
004 System    root      -         -                       -         -
```

```
005 Default   -         -         -                    -         -
```

6. Starting WLM in passive mode

WLM can be run in either “passive” or “active” mode.

Passive WLM places all processes in the defined classes and lets you monitor the classes without controlling anything.

Active WLM proactively controls the classes based on the share, tier, rset, and limit attributes.

- Select *“Start/Stop/Update WLM”* from the initial Workload Manager screen.This will display the screen to start, stop or update WLM.

Example 9-11 Screen output for starting/stopping/updating WLM

```
                                    Start/Stop/Update WLM

Move cursor to desired item and press Enter.

  Start Workload Manager
  Update Workload Manager
  Stop Workload Manager
  Show WLM status
```

- Select *“Start Workload Manager”.* A screen to select attributes for starting the Workload Manager is displayed.
- Specify *“Management mode”* as Passive and select No for *“Enforce Resource Set bindings”.*

Example 9-12 Smitty screen output for starting WLM

```
                                Start Workload Manager

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* Configuration, or for a set: set name/currently          current
      applicable configuration
  Management mode                                        Passive
  Enforce Resource Set bindings                          No
  Disable class total limits on resource usage           Yes
  Disable process total limits on resource usage         Yes
  Start now, at next boot, or both ?                     Now
```

- Select *“Show WLM status*” from the “*Start/Stop/Update WLM*” screen. This will display information about WLM status.

Example 9-13 Smitty screen output for WLM class listing

```
                                                    COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

WLM is running in passive mode, Rset bindings not active.
Checking classes and rules for 'current' configuration...
System
Default
Shared
app1
app2
app3
```

WLM commands

WLM configuration can also be done using simple command line options. Table 4-1 gives a brief overview of the WLM commands and their usage.

Table 9-1 WLM commands

Command	Description	Usage
mkclass	Creates a WLM class	mkclass <class name> mkclass -a inheritance=yes -a localshm =yes <class name>
wlmassign	Assigns a process to a WLM class	wlmassign <class name> <process id>
lsclass	Returns the list of superclasses	lsclass
wlmcheck	Checks WLM settings	wlmcheck
rmclass	Removes a WLM class	rmclass <class name>
wlmcntrl -p	Starts WLM in passive mode	wlmcntrl -p
wlmcntrl -a	Starts WLM in active mode	wlmcntrl -a
wlmcntrl -o	Stops WLM	wlmcntrl -o

The class assignment rules for a class can be added by editing the **/etc/wlm/current/rules** file. All the user defined classes must be added above

the System and Default class line, because the rules file is examined from top to bottom to decide the class of a process.

Example 9-14 A sample /etc/wlm/current/rules file

```
* class resvd user group application type tag
app3    -       -       -       /work/app3/prog3        -       -
app2    -       user2   -       -       -       -
app1    -       user1   -       -       -       -
System   -    root   -    -           -   -
Default  -    -      -    -           -   -
```

Example 9-15 A sample /etc/wlm/current/classes file

```
System:
Default:
Shared:
app1:
        description = "CPU intensive app"
        inheritance = "yes"
        localshm = "yes"
app2:
        description = "CPU intensive app"
        inheritance = "yes"
        localshm = "yes"
app3:
        description = "Memory intensive app"
        inheritance = "yes"
        localshm = "yes"
```

9.1.4 WLM performance tools

Various tools are available on AIX to monitor WLM class resource usage. These tools give an idea of how the resources are being utilized on the system by the processes, and can be used by system administrators for resource monitoring and control. Some of these tools are available with the AIX operating system and the others have to be installed separately.

This section provides a brief introduction to the following most commonly used tools for monitoring WLM classes. Please refer to the redbook *AIX 5L Workload Manager (WLM)*, SG24-5977.

- wlmstat
- topas
- svmon
- Performance Toolbox

wlmstat

The **wlmstat** command reports the WLM per class resource utilization. If a count is specified, wlmstat loops *count* times and *sleeps* interval seconds after each block is displayed.

wlmstat -l [Class] -t [Tier] [Interval][Count].

wlmstat displays information about CPU, memory and disk I/O utilization for all the predefined and user defined classes.

Example 9-16 Sample output of wlmstat command

```
p630n02][/etc/wlm/current]> wlmstat
         CLASS CPU MEM DKIO
     Unmanaged   0  14   0
       Default   0   0   0
        Shared   0   1   0
        System   0   7   0
          app1  44   1   0
          app2  22   0   0
          app3   8  55   0
         TOTAL  74  64   0
```

wlmstat can be used to display individual information in detail on CPU, memory or disk I/O using the Svc, Svm or Svi flags respectively.

Example 9-17 Sample wlmstat output displaying detailed CPU usage statistics

```
[p630n02][/etc/wlm/current]> wlmstat -Svc
         CLASS tr i #pr CPU sha min smx hmx des  rap urap pri
     Unmanaged  0 0   0   0  -1   0 100 100 100  100    0  10
       Default  0 0   0   0  -1   0 100 100 100  100    0   0
        Shared  0 0   0   0  -1   0 100 100 100  100    0   0
        System  0 0  80   0  -1   0 100 100 100  100    0   0
          app1  0 1   2  43  -1   0 100 100 100  100    0   0
          app2  0 1   2  19  -1   0 100 100 100  100    0   0
          app3  0 1   1  06  -1   0 100 100 100  100    0   0
```

topas

The topas command displays performance statistics updated on the screen at regular intervals. When used with -**W** flag the command displays information on percentage of CPU, memory and disk I/O utilization for the WLM classes.

Example 9-18 topas -W

```
[p630n02][/work]> topas -W
Topas Monitor for host:    p630n02     Interval:   2    Mon Oct 25 15:13:04
2004

WLM-Class (Passive)             CPU%      Mem%     Disk-I/O%
app1                            47         1          0
app2                            23         0          0
app3                            12        55          0
System                           0         7          0
Shared                           0         1          0
Default                          0         0          0
Unmanaged                        0        14          0
Unclassified                     0         0          0
```

svmon

The **svmon** command captures and analyzes a snapshot of virtual memory. svmon provides the ability to report workload management related activity with the following 2 types of report:

Class Report Prints memory usage information pertinent to a class. Usage is with the -W flag.

Tier Report Prints memory usage information with respect to a class tier. Usage is with the -T flag.

Example 9-19 Using svmon with WLM

```
[p630n02][/work]> svmon -W app3
WLM is running in passive mode

===============================================================================
Superclass                          Inuse      Pin     Pgsp  Virtual
app3                               512670        4        0   512077

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Virtual
   20564         - work                             -  65536     0    0 65536
   90572         - work                             -  65536     0    0 65536
   88551         - work                             -  65536     0    0 65536
   c8579         - work                             -  65536     0    0 65536
   485e9         - work                             -  65536     0    0 65536
   f855f         - work                             -  65536     0    0 65536
   c85b9         - work                             -  65483     0    0 65485
   18543         - work                             -  53319     0    0 53319
   18463         - clnt /dev/hd2:49716              -    304     0    -     -
   48589         - clnt /dev/hd9var:467             -     57     0    -     -
```

```
98433        - clnt /dev/hd3:4103              -     57     0    -     -
  540        - clnt /dev/hd2:1098              -     42     0    -     -
80550        - work                            -     37     0    0    37
48449        - clnt /dev/hd2:289               -     32     0    -     -
20544        - clnt /dev/hd2:1025              -     19     0    -     -
104e2        - clnt /dev/hd2:45304             -     17     0    -     -
  560        - work                            -     15     0    0    15
```

Performance Toolbox

The **wlmmon** and **wlmperf** commands provide graphical views of Workload Manager resource activities by class.

The **wlmmon** and **wlmperf** commands generate resource usage reports of system WLM activity. The wlmperf command, which is a part of the Performance Toolbox (PTX), can generate reports from trend recordings made by PTX daemons for periods covering minutes, hours, days, weeks, or months.

The wlmmon command generates three types of visual reports:

- Snapshot display
- Detailed display
- Tabulation display

While the **wlmstat** command provides a per-second view of WLM activity, it is not suitable for the long term analysis (it is resource consuming). To supplement the **wlmstat** command, the **wlmmon** and **wlmperf** commands provide reports of WLM activity over much longer time periods, with minimal system impact.

9.2 Partition load manager (PLM)

The Partition Load Manager (PLM) software is part of the Advanced POWER Virtualization feature and helps customers to maximize the utilization of processor and memory resources of DLPAR capable logical partitions running AIX 5L on pSeries servers.

This section is based on the redbook *Advanced POWER Virtualization on IBM @server p5 Servers: Introduction and Basic configuration*, SG24-7940.

9.2.1 PLM introduction

The PLM is a resource manager, which assigns and moves resources based on defined policies and utilization of the resources in an IBM @server pSeries based on POWER5 architecture (@server p5). PLM manages memory, both dedicated processor and partitions using Micro-Partitioning technology to

readjust the resources. This adds additional flexibility on top of the micro-partitions flexibility added by the POWER Hypervisor.

PLM, however, has no knowledge about the importance of any workload running in the partitions and cannot readjust priority based on the changes of types of workloads. Currently, PLM only manages partitions running AIX.

PLM is set up in a partition or on another system running AIX 5L V5.2 ML4 or AIX 5L V5.3. Linux or i5/OS support for PLM and the clients is not available. You can have other installed applications on the partition or system running the PLM as well. A single instance of the PLM can only manage a single server.

To configure PLM, you can use the command line interface or the Web-based System Manager for graphical set up.

PLM uses a client/server model to report and manage resource utilization. The clients (managed partitions) notify the PLM server when resources are either under or over-utilized. Upon notification of one of these events, the PLM server makes resource allocation decisions based on a policy file defined by the system administrator.

PLM uses the Resource Monitoring and Control (RMC) subsystem for network communication, which provides a robust and stable framework for monitoring and managing resources. Communication with the Hardware Management Console (HMC) to gather system information and execute commands PLM requires a configured SSH connection (both server and client running on all partitions managed bt PLM). Figure 9-1 on page 551 shows an overview of the PLM components.

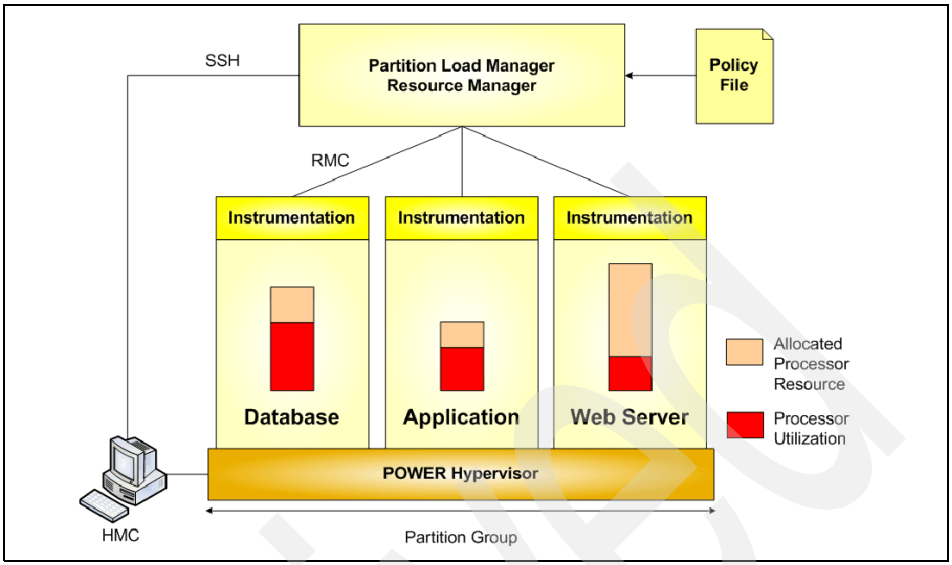

Figure 9-1 PLM overview

The policy file defines managed partitions, their entitlements, their thresholds, and organizes the partitions into groups. Every node managed by PLM must be defined in the policy file along with several associated attribute values:

- Optional maximum, minimum, and guaranteed resource values
- The relative priority or weight of the partition
- Upper and lower load thresholds for resource event notification

For each resource (processor and memory), the administrator specifies an upper and a lower threshold for which a resource event should be generated. You can also choose to manage only one resource.

Partitions that have reached an upper threshold become resource requesters. Partitions that have reached a lower threshold become resource donors. When a request for a resource is received, it is honored by taking resources from one of three sources when the requester has not reached its maximum value:

- A pool of free, unallocated resources
- A resource donor
- A lower priority partition with excess resources over entitled amount

As long as there are resources available in the free pool, they will be given to the requester. If there are no resources in the free pool, the list of resource donors is checked. If there is a resource donor, the resource is moved from the donor to the requester. The amount of resource moved is the minimum of the delta values

for the two partitions, as specified by the policy. If there are no resource donors, the list of excess users is checked.

When determining if resources can be taken from an excess user, the weight of the partition is determined to define the priority. Higher priority partitions can take resources from lower priority partitions. A partition's priority is defined as the ratio of its excess to its weight, where excess is expressed with the formula (current amount - desired amount) and weight is the policy defined weight. A lower value for this ratio represents a higher priority. Figure 9-2 shows an overview of the process for partitions.

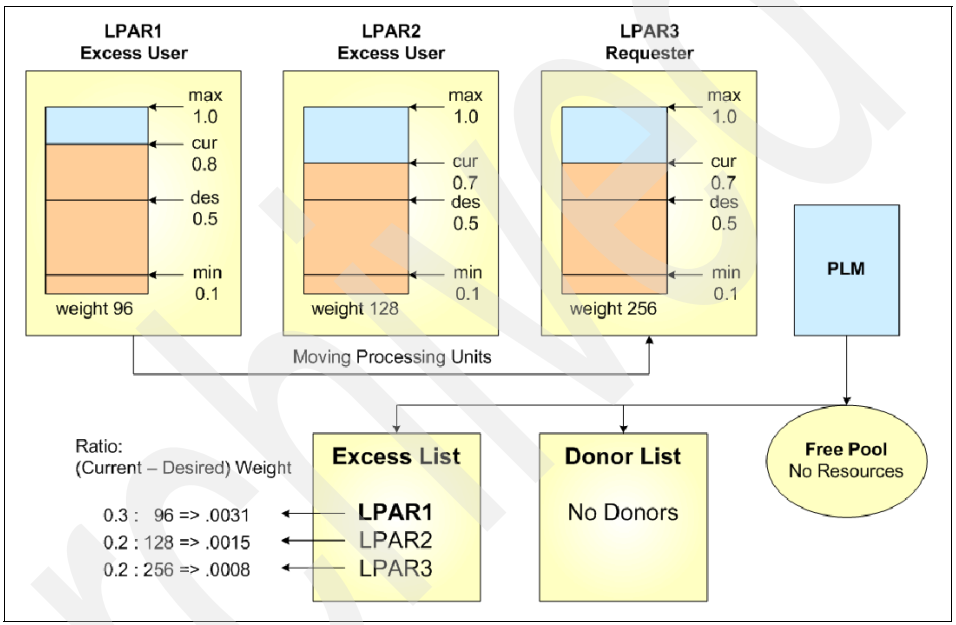

Figure 9-2 PLM resource distribution for partitions

In Figure 9-2, all partitions are capped partitions. LPAR3 is under heavy load and over its high CPU average threshold value becoming a requestor. There are no free resources in the free pool and no donor partitions available. PLM now checks the excess list to find a partition having resources allocated over its guaranteed value and with a lower priority. Calculating the priority, LPAR1 has the highest ratio number and therefore the lowest priority. PLM deallocates resources from LPAR1 and allocates them to LPAR3.

If the request for a resource cannot be honored, it is queued and re-evaluated when resources become available. A partition cannot fall below its minimum or rise above its maximum definition for each resource.

The policy file, once loaded, is static, and has no knowledge of the nature of the workload on the managed partitions. A partition's priority does not change upon the arrival of high priority work. The priority of partitions can only be changed by some action, external to PLM, by loading a new policy.

PLM handles memory and both types of processor partitions: dedicated and shared processor partitions. All the partitions in a group must be of the same processor type.

9.2.2 Memory management

PLM manages memory by moving Logical Memory Blocks (LMBs) across partitions. To determine when there is demand for memory, PLM uses two metrics:

- Utilization percentage (ratio of memory in use to available)
- The page replacement rate

For workloads that result in significant file caching, the memory utilization on AIX may never fall below the specified lower threshold. With this type of workload, a partition may never become a memory donor, even if the memory is not currently being used.

In the absence of memory donors, PLM can only take memory from excess users. Since the presence of memory donors cannot be guaranteed, and is unlikely with some workloads, memory management with PLM may only be effective if there are excess users present. One way to ensure the presence of excess users is to assign each managed partition a low guaranteed value, such that it will always have more than its guaranteed amount. With this sort of policy, PLM will always be able to redistribute memory to partitions based on their demand and priority.

9.2.3 Processor management

For dedicated processor partitions, PLM moves physical processors, one at a time, from partitions that are not utilizing them, to partitions that have demand for them. This enables dedicated processor partitions running AIX 5L Version 5.2 and AIX 5L Version 5.3 to better utilize their resources. If one partition needs more processor capacity, PLM automatically moves processors from a partition that has idle capacity.

For shared processor partitions, PLM manages the entitled capacity and the number of virtual processors (VPs) for capped or uncapped partitions. When a partition has requested more processor capacity, PLM will increase the entitled capacity for the requesting partition if additional processor capacity is available.

For uncapped partitions, PLM can increase the number of virtual processors to increase the partition's potential to consume processor resources under high load conditions. Conversely, PLM will also decrease entitled capacity and the number of virtual processors under low-load conditions, to more efficiently utilize the underlying physical processors.

9.3 A comparison of WLM and PLM

AIX offers two methods of vertical server consolidation: workload management with Workload Manager, and partitioning, of which the most recent development is shared processor logical partitions (Micro-Partitioning technology) with PLM. This section compares these two approaches.

With the introduction of shared processor logical partitions (SPLPARs) and PLM, partitions are approaching the flexibility and granularity of WLM classes in their responses to changing load, while providing the additional security of separate operating systems. The sections below compare WLM classes and SPLPARs in terms of their ability to dynamically provision resources (CPU, memory and I/O) to applications, and the features they provide. SPLPARs are not necessarily smaller than 1 CPU, but they can be given CPU entitlement in fractions of 0.01 CPUs (1.75 CPUs, for example). We assume that WLM is configured on dedicated processors.

Table 9-2 Requirement and configuration

WLM	SPLPAR+PLM
WLM is provided free with AIX.	Micro-Partitioning and PLM are provided as part of the advanced POWER virtualization feature for AIX, which is a chargeable option.
WLM is installed by default.	No additional hardware is required.The managed server must have an HMC. LPARs must be defined and installed, and have Resource Management and Control (RMC) connections to the PLM server. The PLM server must be separately installed.
WLM classes, tiers, limits, shares and rules must be manually configured.	POWER Hypervisor (PHYP) entitlements, and PLM shares and capping must be manually configured.

Table 9-3 Allocation and separation

WLM	SPLPAR+PLM
All processes within an operating system (OS) are assigned to a class.	All processes run within a partition.
All classes run within the same OS. An OS crash will stop all the classes.	Partitions run separate OSs. An OS crash in one partition will have no effect on the others.
A process in one class can start a process in another class.	A process in one partition can only start a process in another partition using network communication.
A resource sets can be used to restrict a class to particular CPUs.	The administrator has no control over which CPUs in the shared pool are used by a particular partition. However, LPARs can be grouped so they only compete against others in the group.

Table 9-4 Performance overhead

WLM	SPLPAR+PLM
WLM is built into the definition of a process. Once running, the overhead is minimal.	Resource Management and Control (RMC) services gather and export the system status. The RMC daemon also processes reconfiguration (dynamic LPAR) requests from the HMC.
WLM can significantly increase the boot time of an OS if the number of disks attached is large.	The RMC services are always started on boot.
Only one OS is required.	Each partition must have its own OS.
Dedicated partitions are the 'default state' against which SPLPAR performance is measured. AIX 5.3 on POWER5 has set a number of benchmark records.	The performance penalty of sharing processors depends on factors such as the size of the partition and the number of other partitions running.

Table 9-5 Resource entitlement

WLM	SPLPAR+PLM
Classes can have maximum, minimum, and target resource entitlements. A class may be given less than its target, if all classes are under heavy load. It will only be given less than its minimum if it cannot use the resources, or if a higher tier class (see "prioritization") takes all the resources.	Partitions can have maximum, minimum, and guaranteed resource entitlements in the PHYP. A partition will only be given less than its guaranteed amount if it cannot use the resources assigned to it. It will never be given less than its minimum entitlement.
Target entitlements are known as shares. The resources given to a class are determined by its share divided by the total number of shares for active classes. An active class is one with running processes.	Partitions are assigned a share in PLM. The resources given to an LPAR are determined by its share divided by the total number of shares for active LPARs. PLM will override the PHYP's normal distribution of these additional resources.
A class with a maximum entitlement of 100% can use any free resources on the system.	An uncapped partition can use any free resources on the system, as PLM will increase a partition's virtual processors in order to exploit additional CPUs.
I/O throughput can be controlled. I/O resources can be shared between classes.	I/O throughput is not controlled. I/O resources can only be shared through a VIO server. PLM cannot move I/O resources between partitions.
The sum of the defined minimum resource entitlements of all the classes cannot exceed the total capacity of the system, even if some classes are not active (have no processes running).	The sum of the defined minimum capacity entitlements can exceed the total capacity of the system as long as not all the partitions are started.

Table 9-6 Prioritization

WLM	SPLPAR+PLM
Classes can be put into tiers. Processes in a lower tier class will only run if no higher tier processes are running. Higher tier classes, therefore, cannot be limited by lower tier classes, but lower tier classes can be starved.	PLM has no concept of the importance of a workload beyond the share setting (see "resource entitlement"). Running a lower priority SPLPAR will limit the resources available to a higher priority SPLPAR because the lower priority SPLPAR will still use its guaranteed entitlement. However, lower priority SPLPARs cannot be starved.

WLM	SPLPAR+PLM
Processes can be started, and classes activated, even if they cannot achieve their minimum entitlement.	New partitions will not start if their minimum requirements cannot be met.

Table 9-7 Speed of response to changing load

WLM	SPLPAR+PLM
There is no latency associated with a class using additional CPU.	There is a latency associated with dynamically adding virtual processors. Furthermore, if a high number of virtual processors are made permanently available instead, a performance overhead is incurred. Additional entitlement (up to 100% of a partition's virtual processors) can be added without delay.
Monitoring is constant. Access to a class's resources is provided on a per-minute basis (as long as the class can use its full entitlement).	Monitoring is based on 10 second intervals. By default, a threshold must be reached 6 times in order to trigger a dynamic LPAR event. Entitlement changes are made only when an event is triggered, but excess capacity is distributed constantly (based on shares).

WLM still provides a greater degree of control and granularity, and classes are still more dynamic in their response to changes in load than an SPLPAR, although these differences are becoming less noticeable. By running separate operating systems, SPLPARs provide an additional degree of separation with clear advantages for availability. PLM can also run with dedicated partitions, avoiding the performance overhead of SPLPARs, but reducing the granularity of control still further.

9.4 Resource monitoring and control (RMC)

The Resource Monitoring and Control (RMC) application is a part of Reliable Scalable Cluster Technology (RSCT). RMC is the strategic technology for monitoring and event management 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 various aspects of the system resources (hardware and software), and can specify a wide range of actions to be taken when a threshold

or specified condition is met. If configured, RMC can also react in response (automated response) to conditions and events occurred on the system on in a cluster.

RMC monitors, among other things, several performance related aspects, like CPU, memory, file systems, paging space etc.

By monitoring conditions of interest and providing automated responses when these conditions occur, RMC helps maintain system availability.

The whole RSCT package is composed by following filesets

rsct.core	Core RSCT component including RMC
rsct.basic	Basic functions supporting availability infrastructure such as Topology Services (HATS) and Group Services (HAGS)
rsct.compat.basic	Event Management (HAEM)
rsct.compat.clients	Client services of Event Management (HAEM)

RMC is included in the rsct.core package, which is installed automatically with AIX 5L Version 5.3. The RSCT application executables reside in /usr/sbin/rsct/bin directory. This package provides basic RMC services and some additional RSCT functions.

The other RSCT packages such as rsct.basic and rsct.compat.basic come with AIX 5.3 installation media, but they aren't installed automatically.

Services provided by those packages such as HATS, HAGS, and HAEM are very important to certain applications. Cluster Systems Management (CSM), Parallel System Support Programs (PSSP), and High Availability Cluster Multi-Processing/Enhanced Scalability (HACMP/ES) are applications using those services. Note that HAEM has been moved from the rsct.basic and rsct.clients packages to the rsct.compat package, and it is currently supported only in PSSP, and partially in HACMP.

RMC can be configured and used through the WebSM Graphical User Interface (GUI), but it also provides command line interface programs (commands) that can be used to manage it. For additional information, see the *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.

9.4.1 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 5.3 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.

9.4.2 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 sources, depending on the resource. RMC "aware" 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 rearm[1] expression is true.

Comparing RMC with HAEM

High Availability Event Management (HAEM) is another facility of monitoring and controlling system resource that used by old version of RSCT. Now, all of its basic functions have been replaced by RMC equivalents. For instance, HACMP/ES Version 5.2 is mostly implemented by using RMC facilities and it is different from traditional way of its development which is based on HAEM infrastructure.

When you compare RMC with HAEM, you can find many similarities. 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.

[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.

Abstractions used in RMC

In order to provide consistent monitoring and controlling interfaces of system resources, RMC maintains some abstractions that will provides more concrete logical infrastructures. In the performance monitoring perspective, we need to understand some of those abstractions and relationship between them.

Some important abstractions are listed in following paragraph

Physical/Logical device
: This means actually devices which we encounter in everyday life, such as filesystem, paging device, CPU, memory and so on. Most of important system devices are predefined as the RMC resource.

Resource
: The fundamental concept of RMC's architecture. It is mapped to an instance of a physical or logical devices that provides services to some other component of the system.

Resource class
: A set of resources of the same type. For example, the resource group IBM.PagingSpace contains resources that indicates physical entity "/dev/hd6" and "/dev/paging00"

Resource manager
: A daemon process that provides the interface between RMC and actual physical or logical entities. This also trigger registered response action when specified condition is met.

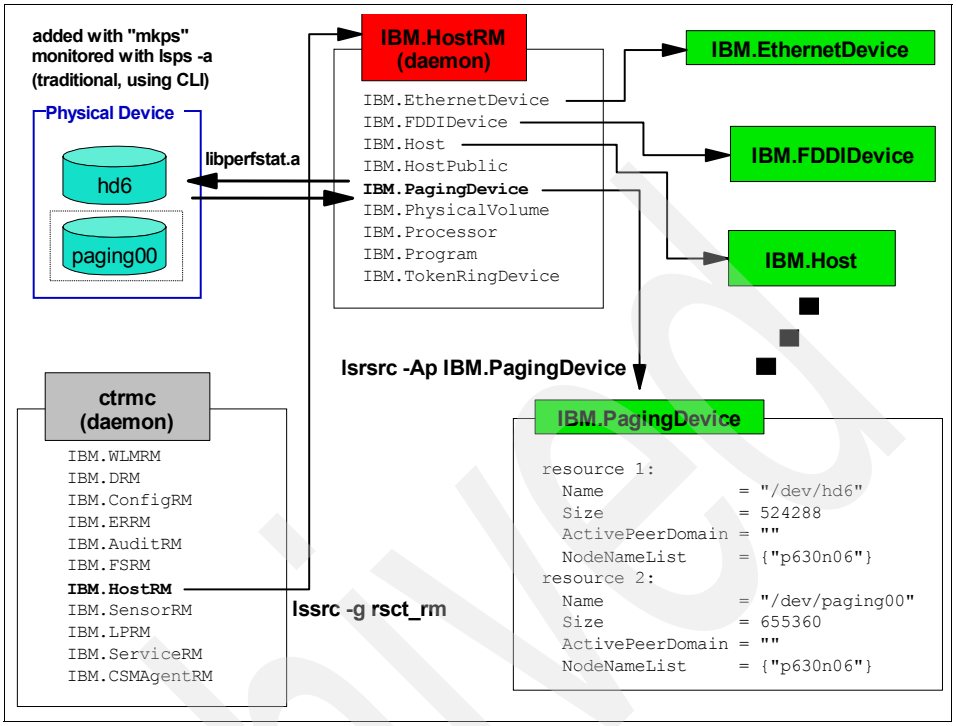

Figure 9-3 RMC diagram

Figure 9-3 illustrates how RMC works when it monitors a specific device. In this case, “/dev/hd6” and “/dev/paging00” or system paging devices are physical entities to be monitored. These entities are mapped to the RMC resources (the instance of IBM.PagingSpace resource class). Between physical devices and resource, resource manager exists and is responsible for defining and mapping those two abstractions. The resource manager IBM.HostRM is also responsible for other important resource classes such as IBM.PhysicalVolume, IBM.Processor and so on. Resource managers running on the system are registered in the form of the SRC subsystem. Of course, IBM.HostRM is one of those. Therefore, the status of the resource manager can be monitored by **`lssrc -s IBM.HostRM`** command.

In order to gather performance data, the IBM.HostRM takes advantage of the calls of perfstat library (/usr/lib/libperfstat.h) which is very relevant to general performance monitoring commands, such as **`vmstat, iostat`** and **`topas`**. For other basic system information, general commands and calls are used as well. Then RMC commands like **`lsrsrc`** use RSCT libraries (/usr/sbin/rsct/lib/libct_*) and retrieve information gathered by resource managers.

Most of the attributes we can expect from the certain system device are predefined in resource classes and supported by RMC. For instance, you can see the status of paging devices by issuing **`lsrsrc -Ad IBM.PagingDevice`**. This result should be same as the execution result of **`lsps -a`**.

Here, we have a detailed explanation on abstractions mentioned so far.

Resource managers

A resource manager is a stand-alone daemon. The resource manager contains definitions of all resource classes that the resource manager manages.

You can list resource managers in your system by using **`lssrc -g rsct_rm`** command.The following resource managers are provided with the RMC fileset:

IBM.AuditRM The Audit Log resource manager (AuditRM) provides a system-wide facility for recording information about the 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.

Beside the basic resource managers, you can also add customized resource managers for the specific needs of applications. For example, a resource manager IBM.DMSRM will be added to the system when you install CSM.

Resource classes

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 class definition includes a description of all attributes, actions, and other characteristics of a resource class. Resource classes can be seen by "lsrsrc" command and each resource classes is under control of a certain resource manager. The following list describes resource manager and its resource classes:

- Audit Log resource manager (IBM.AuditLog)

  - IBM.AuditRM
  - IBM.AuditLogTemplate
- Configuration resource manager (IBM.ConfigRM)
  - IBM.NetworkInterface
  - IBM.ServiceEvent
  - IBM.ManagementServer
  - IBM.HostPublic
  - IBM.DRM
- File System resource manager (IBM.FSRM)
  - IBM.FileSystem
- Host resource manager IBM.HostRM
  - IBM.ATMDevice
  - IBM.EthernetDevice
  - IBM.FDDIDevice
  - IBM.Host
  - IBM.PagingDevice
  - IBM.PhysicalVolume
  - IBM.Processor
  - IBM.Program
  - IBM.TokenRingDevice
- Event Response resource manager (IBM.ERRM)
  - IBM.Association
  - IBM.Condition
  - IBM.EventResponse

When any physical changes of the system occur (addition or removal of a physical device), it may happen that RMC will not reflect these changes automatically. An easy way to reflect this to the RMC resource class is to issue the **refrsrc** command with proper resource manager. For instance, if an additional ethernet adapter is added by hot-plug facility of the PCI I/O slot, this cannot be listed immediately by **lsrsrc IBM.EthernetDevice** command. To make this device visible from RMC and have the RMC controlling the device, you need to run the command **refrsrc IBM.HostRM**.

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 or rebirth. To find out more about the definition of a class, see “Examining resource classes” on page 565.

9.4.3 Verifying RMC facilities

We are going to see how to verify the status of various RMC objects. SRC commands and RMC commands will be used to verify and control the status of each object.

Verifying that the RMC is active

To verify that the RMC daemons and are active, run the **lssrc** command as shown in Example 9-20.

Example 9-20 Using lssrc to verify RMC daemon

```
# lssrc -g rsct
Subsystem         Group            PID          Status
 ctrmc            rsct             18330        active
 ctcas            rsct             22188        active
```

The output shows that RMC (ctrmc) is active as well as ctcas is running.

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 **rmcctrl** 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.

Verifying the status of resource managers

To verify resource managers are active, run the **lssrc** command as shown in Example 9-21.

Example 9-21 Using lssrc to see resource manager status

```
# 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
```

Examining resource classes

By using **lsrsrc** without any flags, it will show all defined resource classes, as shown in Example 9-22 on page 566.

Example 9-22 Defined resource classes

```
# 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"
```

Examine resources and their attributes

The **lsrsrc** command enables you to verify resources and their attributes. You can combine some flags to examine each of classes in more detail. Dynamic and persistent attributes are defined in the resource classes, and these can be seen for each of the resources.

Persistent attributes define the characteristics of the resource, and they are not dynamically changed by the system. Example of persistent attributes are Device Name, IP Address, and so on.

When we use the -ap (default) flags to the **lsrsrc** command, it will only show the persistent attributes defined for the specified class. Example 9-23 shows the persistent attributes for the IBM.Host resource class.

Example 9-23 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"
```

Dynamic attributes reflect internal states or performance variables of resources and resource classes. For example, all the file system resources have dynamic attributes such as, operational state, %total used, % inode used, and so on.

To verify the dynamic attributes, use the -ad flags with the **lsrsrc** command, as shown in Example 9-24. Note that we get the current value of the attribute as well[2].

Example 9-24 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
        KMemFailProtcbRate  = 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.

```
        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 analyze the class structure, use the **lsrsrcdef** command, as shown in Example 9-25 (we have used for this example the IBM.PhysicalVolume resource class).

Example 9-25 Using lsrsrcdef

```
# lsrsrcdef 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 9-25 on page 568, `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` (See Example 9-26).

Example 9-26 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 9-27 on page 570, or the -xab flags in combination with `-t`. The **`-t`** flag is for formatting the command output in a tabular manner.

Example 9-27 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"
```

9.4.4 Examples using RMC

In this section, we will provide a specific case of system monitoring and how to utilize the RMC facilities. In this case, the PctFree attribute of one of the paging devices will be monitored by RMC. When this value reaches a level set by the user, execution of a response script will be triggered. Then the script will gather paging space related performance data. We use the command line interface, since this is also used for most of the performance monitoring and tuning tools. The GUI (Graphical User Interface) is explained in the redbook *AIX 5L Differences Guide Version 5.3 Edition*, SG24-5765.

To start using monitoring with RMC you have to:

1. Determine monitoring object: You need to decide what resource to monitor and the desired threshold(s).
2. Set monitoring guideline and response action: Establish the monitoring guideline and determine what action to be performed when the event occurs.
3. Writing an event response script: Create a script that will perform the desired action.
4. Creating a condition: Create an RMC condition that meets the monitoring requirements.
5. Creating a response to condition event: Create an RMC response for the action script(s).
6. Associating response with condition: Create an RMC association between the defined RMC condition and RMC response.
7. Activate monitoring for the condition

We constructed our example according to the steps we mentioned in the previous list.

Determine the object to be monitored

In this example we are going to monitor system following paging spaces. Example 9-28 shows devices to be monitored in this case.

Example 9-28 Listing all paging spaces by using lsps

```
[p630n06][/]> lsps -a
Page Space      Physical Volume   Volume Group    Size %Used Active  Auto  Type
paging00        hdisk0            rootvg        2560MB     1   yes    no    lv
hd6             hdisk0            rootvg        2560MB     1   yes   yes    lv
```

In Example 9-29, you can see the same device can be monitored using RMC.

Example 9-29 Listing all paging spaces by using RMC command

```
[p630n06][/]> lsrsrc -Ap  IBM.PagingDevice
Resource Persistent Attributes for IBM.PagingDevice
resource 1:
        Name             = "/dev/hd6"
        Size             = 655360
        ActivePeerDomain = ""
        NodeNameList     = {"p630n06"}
resource 2:
        Name             = "/dev/paging00"
        Size             = 655360
        ActivePeerDomain = ""
        NodeNameList     = {"p630n06"}

[p630n06][/]> lsrsrc -Ad IBM.PagingDevice
Resource Dynamic Attributes for IBM.PagingDevice
resource 1:
        OpState = 1
        PctFree = 99
resource 2:
        OpState = 1
        PctFree = 99
```

Set monitoring guideline and response action

We want to see each paging device's usage by monitoring PctFree attribute of IBM.PagingDevice resource. We regard paging space usage more than 80% as serious situation and usage less than 50% as normal. The following table contains dynamic attributes of IBM.PagingDevice class and monitoring guidelines we want to define.

Table 9-8 monitoring devices and guidelines

Device	Dynamic Attribute	Event Condition	Rearm Condition	Response
/dev/hd6	PctFree	10% <	20% >	Execution of script
/dev/paging00	PctFree	10% <	20% >	Execution of script

We want a script to be executed, if usage of either “/dev/hd6” or “/dev/paging00” exceed the limit. The scripts will gather sufficient information to verify which process is responsible for the increasing the usage of the paging space. The result of this script will be stored in the certain location in the system and a mail with the same content will be sent to system administrator (root user).

Writing an event response script

The basic script in Example 9-30 is an example of how to gather top 10 paging consuming process and top 10 virtual memory consuming processes. It contains **vmstat** output as well. This also explains how to send an e-mail to the root user. The result will be mailed l to the **root** user when a condition occurs that triggers the activation of the event response script.

Example 9-30 An event response shell script example: pgsp_info.sh

```
#!/usr/bin/ksh

DT=`date +%H%M`
LOGFILE=/itso_files/pgsp_$DT.out
SVMON=/usr/bin/svmon
VMSTAT=/usr/bin/vmstat

EVENTTIME=$(perl -e 'use POSIX qw(strftime);print strftime("%Y-%m-%d
%T",localtime('${ERRM_TIME%,*}') );')

exec >> /itso_files/pgsp_$DT.out
exec 2>1&

echo "       TIME OF EVENT : $EVENTTIME"
echo "       CONDITION     : $ERRM_COND_NAME"
echo "       SERVERITY     : $ERRM_COND_SEVERITY"
echo "       EVENT TYPE    : $ERRM_TYPE"
echo "       EXPRESSION    : $ERRM_EXPR"
echo "       RESOURCE NAME : $ERRM_RSRC_NAME"
echo "       RESOURCE CLASS: $ERRM_RSRC_CLASS_NAME"
echo "       DATA TYPE     : $ERRM_DATA_TYPE"
echo "       DATA VALUE    : $ERRM_VALUE"
echo ""
echo "# Top 10 paging space using processes"
$SVMON -Pg -t 1 |grep Pid ; $SVMON -Pg -t 10 |grep "N"
```

```
echo ""
echo "# Top 10 virtual memory using processes"
$SVMON -P -t 1 |grep Pid ; $SVMON -P -t 10 |grep "N"

echo ""
echo "#vmstat output "
$VMSTAT 2 10

#Send execution result to root
cat $LOGFILE |mail -s "RSCT: $ERRM_COND_NAME $ERRM_COND_SEVERITY" root
```

Note: The output is also appended to a debug file in log directory (in this case, /itso_files/) named pgsp_$DT.out. It can be helpful to use logfiles when developing event response scripts.

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 9-31 shows how to use **`perl`** for the conversion.

Example 9-31 Using perl to convert ERRM_TIME

```
perl -e 'use POSIX qw(strftime);print strftime("%Y-%m-%d
%T",localtime('${ERRM_TIME%,*}') );'
```

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 9-32, a condition is defined to use the `IBM.PagingDevice` resource manager.

Example 9-32 Using mkcondition command

```
mkcondition -r IBM.PagingDevice \
        -e "PctFree < 20" \
        -E "PctFree > 50" \
        -d "Paging space usage more than 80%" \
        -D "Paging space usage less than 50%" \
        -s 'Name=="/dev/hd6" || Name=="/dev/paging00"' \
        -V "Pgsp_state"
```

This example creates a condition that monitors the system paging device “/dev/hd6” and, when the evaluation of `PctFree < 20` is true, it generates an event named `"Pgsp state"` and the monitoring stops. When the expression `PctFree >50` becomes true, monitoring will restart. This technique is necessary 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 "Pgsp_state"
```

To check how the definition of the condition appears to RMC, use the **lscondition** command, as in Example 9-33.

Example 9-33 Using the lscondition command

```
[p630n06][/itso_files]> lscondition "Pgsp_state"
Displaying condition information:

condition 1:
        Name             = "Pgsp_state"
        MonitorStatus    = "Not monitored"
        ResourceClass    = "IBM.PagingDevice"
        EventExpression  = "PctFree < 20"
        EventDescription = "Paging space usage more than 80%"
        RearmExpression  = "PctFree > 50"
        RearmDescription = "Paging space usage less than 50%"
        SelectionString  = "Name==\"/dev/hd6\" || Name==\"/dev/paging00\""
        Severity         = "c"
        NodeNames        = {}
        MgtScope         = "l"
```

Creating a response to 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 9-30 on page 572. We define our event response script to RMC:

```
mkresponse -n pgsp_resp -s /itso_files/pgsp_info.sh pgsp_resp_1
```

This 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 shown in Example 9-34.

Example 9-34 Using the lsresponse command

```
[p630n06]> lsresponse pgsp_resp
Displaying response information:

        ResponseName    = "pgsp_resp_1"
```

```
        Action          = "pgsp_resp"
        DaysOfWeek      = 1-7
        TimeOfDay       = 0000-2400
        ActionScript    = "/itso_files/pgsp_info.sh"
        ReturnCode      = 0
        CheckReturnCode = "n"
        EventType       = "a"
        StandardOut     = "n"
        EnvironmentVars = ""
        UndefRes        = "n"
```

Associating response with condition

Create an RMC association between the defined RMC condition and RMC response. 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  "Pgsp_state" "pgsp_resp_1"
```

To check how the definition of our condition/response connection appears to RMC, we can use the **lscondresp** command, as in Example 9-35.

Example 9-35 Using lsresponse command

```
[p630n06][/itso_files]> lscondresp Pgsp_
Displaying condition with response information:

condition-response link 1:
        Condition = "Pgsp_state"
        Response  = "pgsp_resp_1"
        State     = "Not active"
```

Note that we only used the first part of the condition name (Pgsp_).

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 shown in Example 9-36.

Example 9-36 Using the lscondresp command

```
[p630n06][/itso_files]> lscondresp
Displaying condition with response information:
Condition        Response                  State
"Pgsp_state"     "pgsp_resp_1"             "Not active"
...(lines omitted)...
```

The previous example (Example 9-36 on page 577) shows the condition and the response as **"Not active"**. The next step is to activate the monitoring of the condition and the response.

Activate monitoring for the condition

To activate monitoring of a condition, we use the **startcondresp** command. For our condition "Pgsp_state" we use the following command:

```
startcondresp "Pgsp_state"
```

After running the **startcondresp** command, the **"**"Pgsp_state" condition with the "pgsp_resp_1" response will be monitored (Active), as shown in Example 9-37.

Example 9-37 Using the lscondresp command to verify monitoring state

```
[p630n06][/itso_files]> lscondresp
Displaying condition with response information:
Condition        Response                 State
"Pgsp_state"     "pgsp_resp_1"            "Active"
...(lines omitted)...
```

When we check the condition again with the **lscondition** command we get the ouput shown in Example 9-38, which now indicates that the condition is "Monitored".

Example 9-38 Using the lscondition command

```
[p630n06][/]> lscondition Pgsp_
Displaying condition information:

condition 1:
        Name             = "Pgsp_state"
        MonitorStatus    = "Monitored and event monitored"
        ResourceClass    = "IBM.PagingDevice"
        EventExpression  = "PctFree < 20"
        EventDescription = "Paging space usage more than 80%"
        RearmExpression  = "PctFree > 50"
        RearmDescription = "Paging space usage less than 50%"
        SelectionString  = "Name==\"/dev/hd6\" || Name==\"/dev/paging00\""
        Severity         = "i"
        NodeNames        = {}
        MgtScope         = "l"
```

The **startcondresp** command can also be used to create a condition-response association, such as associating the condition "Pgsp_state", with an event response, such as "rsct.trapevent":

```
startcondresp  "Pgsp_state" "pgsp_resp_1"
```

Note, however, that this creates a condition-response association, and also activates it (see Example 9-39, and refer to “Associating response with condition” on page 577).

Example 9-39 Using the startcondresp and lscondresp commands

```
[p630n06][/itso_files]> startcondresp  "Pgsp_state" "pgsp_resp_1"
[p630n06][/itso_files]> lscondresp Pgsp_
Displaying condition with response information:

condition-response link 1:
        Condition = "Pgsp_state"
        Response  = "pgsp_resp_1"
        State     = "Not active"
```

How the condition/response event generation works

When the event-generating expressions for the **“**Pgsp_state**”** condition becomes true, our shell script generates an e-mail message (see Example 9-40).

Example 9-40 Sample monitoring output

```
[p630n06]> mail
mbox: A file or directory in the path name does not exist.
[p630n06][/itso_files]> mail -f /mbox
Mail [5.2 UCB] [AIX 5.X]  Type ? for help.
"/mbox": 2messages
>   1 root              Wed Oct 13 15:41  63/3414 "RSCT: Pgsp_state
Information"
    2 root              Wed Oct 13 15:41  63/3417 "RSCT: Pgsp_state
Information"
? 1
Message  1:
From root Wed Oct 13 15:41:57 2004
Date: Wed, 13 Oct 2004 15:41:18 -0500
From: root
To: root
Subject: RSCT: Pgsp_state Informational

        TIME OF EVENT : 2004-10-13 15:39:38
        CONDITION     : Pgsp_state
        SERVERITY     : Informational
        EVENT TYPE    : Event
        EXPRESSION    : PctFree < 20
        RESOURCE NAME : /dev/paging00
        RESOURCE CLASS: Paging Device
        DATA TYPE     : CT_INT64
        DATA VALUE    : 79

# Top 10 paging space using processes
```

```
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
  721148 elephant       1781946     4591   159003  1939918      Y     N     N
  241740 java             18545     4592    15630    30436      N     Y     N
  352468 java             18087     4596     8781    25327      N     Y     N
  401584 java              9701     4585     7200    19445      N     Y     N
  266260 Xvnc             11620     4572     5658    18671      N     N     N
  176224 snmpmibd64        5820     4591     4677    10676      Y     N     N
  110758 shlap64           5863     4591     4656    10662      Y     N     N
  487580 svmon_back.64     5930     4591     4537    10616      Y     N     N
  671988 nmon64            6216     4591     4537    10898      Y     N     N
  229500 rpc.statd         9087     4575     4006    15873      N     Y     N

# Top 10 virtual memory using processes
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
  721148 elephant       1781952     4591   159748  1940704      Y     N     N
  241740 java             18545     4592    15630    30436      N     Y     N
  352468 java             18087     4596     8781    25327      N     Y     N
  266260 Xvnc             11620     4572     5658    18671      N     N     N
  454694 IBM.ERrmd         9818     4585     3806    16219      N     Y     N
  401584 java              9701     4585     7200    19445      N     Y     N
  606408 lslv              9463     4573     3371    15586      N     N     N
  376908 Xvnc              9455     4572     3766    15680      N     N     N
  246000 Xvnc              9449     4572     3776    15680      N     N     N
  635000 sendmail          9419     4572     3371    15387      N     N     N

#vmstat output

System configuration: lcpu=4 mem=8192MB

kthr    memory              page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs us sy id wa
 1  1 2291621    96   0   1 450 535  550   0 204 3973 4322 25  1 30 44
 1  2 2292513     0   0   1 398 307  322   0 225 2905 4254 30  1 25 44
 1  2 2294039     3   0   4 769 816  832   0 222 4792 6362 25  2 32 42
 1  1 2295290     2   0   2 628 506  508   0 237 4743 5872 25  2 49 24
 1  1 2296048     3   0   3 382 200  204   0 206 3787 3994 25  1 45 29
 1  1 2297078     0   0   1 517 332  333   0 222 4203 5121 25  1 38 36
 1  3 2300218    15   0   8 1587 1642 1657   0 264 7443 14217 27  3 27 42
 1  5 2301388     2   0   3 583 769  769   0 178 2624 5699 25  1  4 69
 1  5 2302324    98   0   6 520 465  474   0 191 2158 4614 25  1  0 74
 1  5 2302917    19   0   1 259 361  365   0 179 1474 3042 25  1  0 74

?
```

With this information, you can find which process caused paging space problem. In this case, the process *elephant (*with process ID 721148) is the most suspicious one. This process is consuming a large amount of virtual memory and

paging space at the same time. This result also provides additional information, like needed active virtual memory (avm), and the amount of freelist the system is currently maintaining. Note that our event response script also appended the output to a file in the /itso_files directory named pgsp_$DT.out.

Stopping the monitoring of a condition

To stop monitoring a condition, use the **stopcondresp** command (here applied to our sample condition/response monitoring event for the `paging devices`):

```
stopcondresp "Pgsp_state"
```

To verify that the monitoring has stopped, use the **lscondresp** command, as in Example 9-41.

Example 9-41 Using the lscondresp command

```
[p630n06][/itso_files]> lscondresp
Displaying condition with response information:
Condition        Response                  State
"Pgsp_state"     "pgsp_resp_1"             "Not active"
...(lines omitted)...
```

Removing a response definition

Since RMC uses a hierarchical structure, whenever you want to remove an object (in this case, a response definition), you must remove also any dependencies, relations and associations.

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 pgsp_resp_1
```

Thus, you have to perform the same operation following these steps:

- First, remove the association of the response from the condition (in our example, between the "`Pgsp_state`" condition and **“**`pgsp_resp_1`**”** response) as shown below:

  ```
  rmcondresp "Pgsp_state" “pgsp_resp_1”
  ```

- Next, the response definition can be removed:

  ```
  rmresponse pgsp_resp_1
  ```

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:

```
rmcondition -f "Pgsp_state"
```

You can also perform the same operation in two steps, by first disassociating the response from the condition (in our example, between the "Pgsp_state" condition and “pgsp_state_1” response):

```
rmcondresp "Pgsp_state" “pgsp_resp_1”
```

And second, by removing the condition:

```
rmcondition "Pgsp_state
```

10

Performance monitoring APIs

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. Finally, we show some examples of using other performance-monitoring subroutines that are available on AIX.

This chapter contains the following sections:

10.1 The performance status (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 Perstat API enabled the developers to write a performance monitoring application with simple and consistent interface. Before Perfstat API, developers were supposed to manipulate various structures and calls in their own code. Without perfstat API, you were supposed to manipulate kernel memory interface ("/dev/kmem") directly. This also required that you have a good understanding about kernel data structures and related subroutines.

Now, with Perfstat API, all of these functions are integrated into one interface (Perfstat kernel extension - kex) which makes calls on behalf of user application. This kex contains ODM calls as well. For instance, with just a few subroutines in perfstat API, such as perfstat_disk() and perfstat_cpu_total(), you can simply write an application which is similar to **`iostat`**, and you just need to include one header file (*perfstat.h*) in your program.

On the contrast, without Perfstat API, you need to play with a lot of subroutines and structures, such as iostat.h, sysinfo,h, odm.h. Figure 10-1 on page 585 shows a comparison between traditional ways of monitoring application development and the newly introduced development method using the perfstat library.

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.

Note: The API is under development, and will have additional API subroutines and data structures in future releases.

The internal perfstat kex access mechanisms are not publicly available. Only the perfstat Library API will be maintained for public use.

The Perfstat API subroutines reside in the libperfstat.a library in the /usr/lib directory (or, in /lib, which 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 the bos.perf.perfstat fileset as prerequisite.

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 5.3 Technical Reference: Base Operating System and Extensions, Volume 1,* SC23-4913.

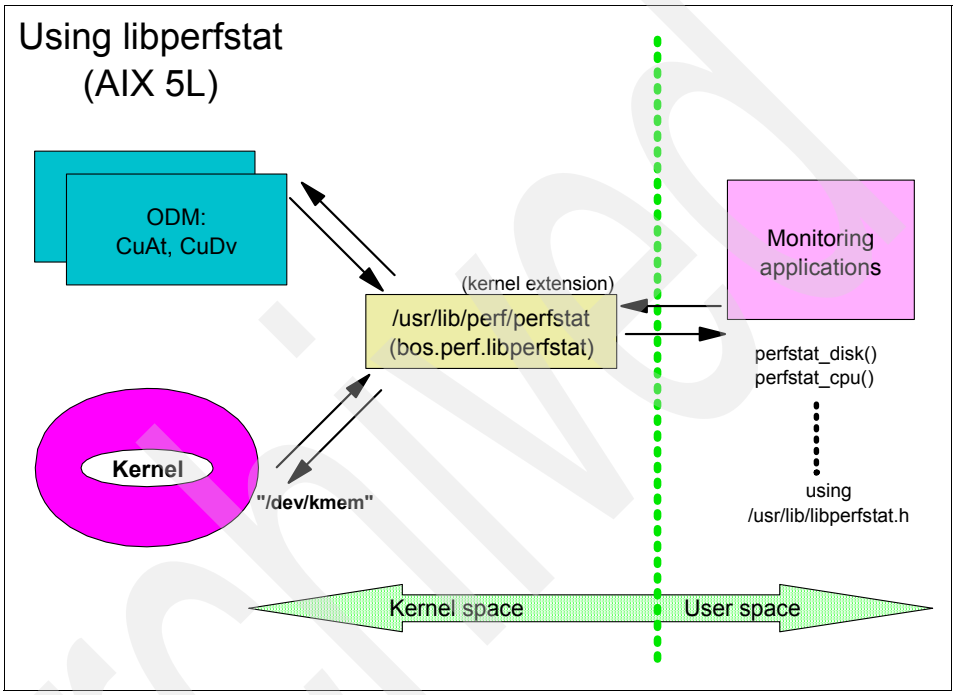

Figure 10-1 Comparison traditional monitoring application with application using perfstat

For comparison, the traditional way for performance monitoring coded inside applications is presented in Figure 10-2 on page 586.

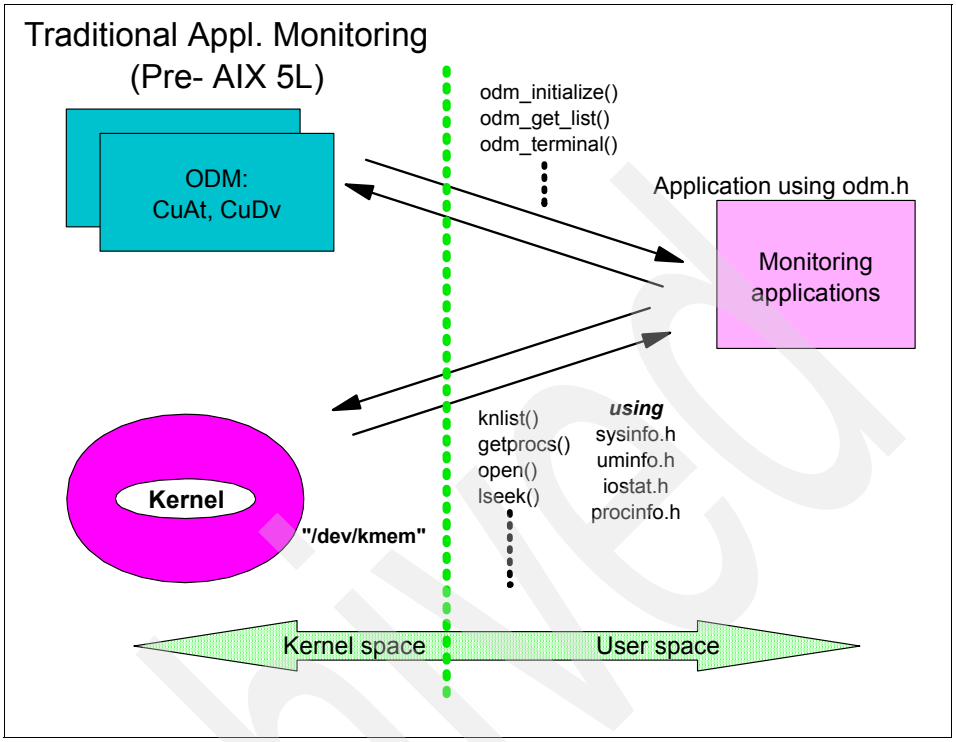

Figure 10-2 Traditional application monitoring (without libperfstat)

10.1.1 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 shown in Example 10-1.

Example 10-1 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.

10.1.2 Changing history of perfstat API

Ever since perfstat API was introduced in AIX 5L, the new functions and performance monitoring fields are added to the API, as the version of AIX

evolves. For instance, in AIX 5L V5.2, the *perfstat_diskadapter()* call has been added to perfstat API, and this enables users to retrieve performance statistics about disk adapters (such as SCSI and FC adapters). Any program using call can be run on AIX 5.2 or higher, but you can't run this on AIX 5.1. This means you have to be careful with the backward compatibility if you want to backport your applications to earlier operating system versions.

The Figure 10-3 is explains the relationship between perfstat call and each versions of AIX.

For complete specifications for changing history of perfstat API subroutines and structures, refer to *AIX 5L Version 5.3 Performance Tools Guide and Reference*, SC23-4906. The header file "/usr/include/libperfstat.h/" of each version of AIX provides detailed information about the calls supported.

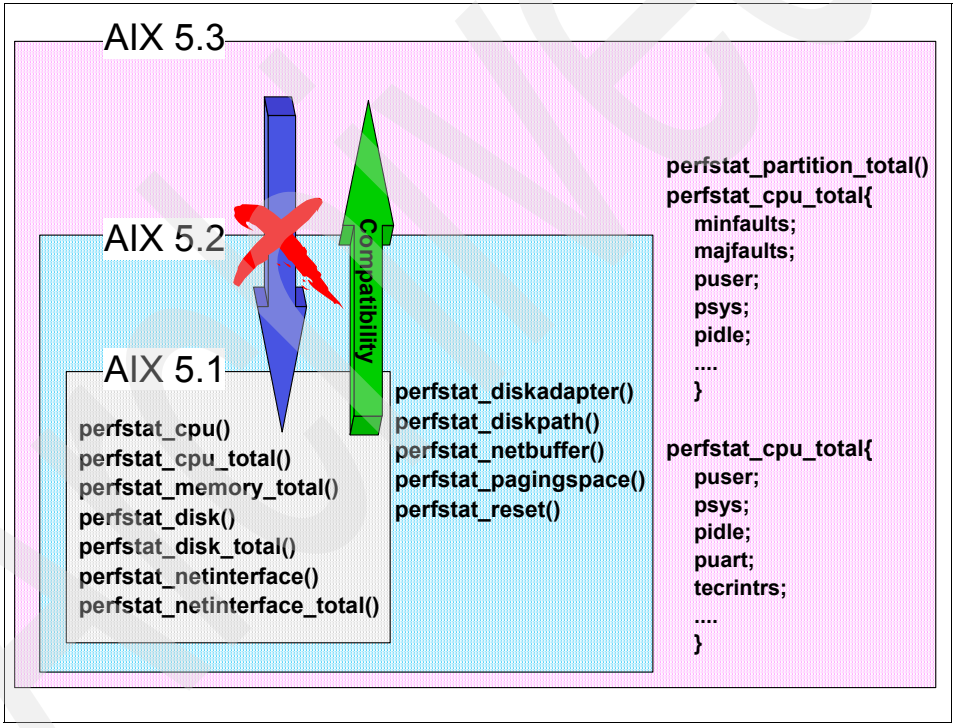

Figure 10-3 Additions have been made to the perfstat APIs

10.1.3 Subroutines

The Perfstat API subroutines cover various aspects of the monitored system, such as CPU, memory etc. The following is a classification of these subroutines.

Subroutine Types and classification

The following subroutines (components of Perfstat API) are categorized into CPU, disk, network, memory, disk, and other areas:

CPU related subroutines

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_cpu_total The perfstat_cpu_total subroutine returns global CPU usage statistics.

Memory related subroutines

perfstat_memory_total The perfstat_memory_total subroutine returns global memory usage statistics

perfstat_pagingspace The pefstat_pagingspace subroutine retrieves individual paging space usage. The same function can be used to retrieve the number of available sets of paging space statistics

Disk related subroutines

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_disk_total The perfstat_disk_total subroutine returns global disk usage statistics.

perfstat_diskadapter The perfstat_diskadapter subroutine retrieves one or more individual diskadapter usage statistics. The same function can also be used to retrieve the number of available sets of diskadapter statistics.

perfstat_diskpath The perfstat_diskpath subroutine retrieves one or more individual diskpath usage statistics. The same function can also be used to retrieve the number of available sets of diskpath statistics. This subroutine can be used for mpio environment

Network related subroutines

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_netinterface_total	The perfstat_netinterface_total subroutine returns global network interface usage statistics.
perfstat_netbuffer	The perfstat_netbuffer subroutine retrieves the individual network buffer allocation usage statistics. The same function can also be used to retrieve the number of available sets of network buffer allocation statistics.
perfstat_protocol	The perfstat_netbuffer subroutine retrieves the individual network buffer allocation usage statistics. The same function can also be used to retrieve the number of available sets of network buffer allocation statistics

Other subroutines

perfstat_partition_total	The pefstat_partition_total subroutine returns global partition usage statistics
perfstat_reset	The perfstat_reset subroutine is called to clear the dictionary whenever the machine configuration has changed

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 Example: A-2, “perfstat_dude.c program” on page 670.

Global and component-specific subroutine

Now, we are going to consider another classification of Perfstat subroutines. Subroutines can be classified into two major categories, one contains the global subroutines that reports a values about a set of components, and the other contains component-specific subroutines that reports a values about individual components on a system.

Global subroutines

Global subroutines are in the identical from and take the similar types of arguments. Example 10-2 shows the basic format of this type of subroutine.

Example 10-2 Global subroutine prototype

```
int
perfstat_comp_total (perfstat_id_t, perfstat_comp_total_t, sizeof_struct,
desired_number)
```

This subroutine retrieves statistics related to a set of components. In the returned structure only one set of data will be provided. Memory, disk, netinterface and partition are the available components for global subroutines. Subroutines belong to this categories are:

- perfstat_memory_total ()
- perfstat_disk_total ()
- perfstat_netinterface_total ()
- perfstat_partition_total ()

Component-specific subroutines

Component-specific subroutines are in the identical form and take the similar types of arguments. Example 10-3 shows the basic format of this type of subroutine.

Example 10-3 Component-specific subroutine prototype

```
int
perfstat_comp (perfstat_id_t, perfstat_comp_total_t, sizeof_struct,
desired_number)
```

The subroutine will retrieve individual component metrics. In returned structure multiple set of metric will be provided. CPU, disk, diskpath, diskadapter, netinterface, protocol, netbuffer, pagingspace can be the component for component-specific subroutines. Subroutines belong to this categories are:

- pefstat_cpu ()
- perfstat_disk ()
- perfstat_diskpath ()
- perfstat_diskadapter ()
- perfstat_netinterface ()
- perfstat_protocol ()
- perfstat_netbuffer ()
- perfstat_pagingspace ()

Subroutine specification and examples

In this section, we will cover detailed specifications for each subroutines and provide simple exemplary codes. We will cover most of subroutines provided by perfstat API, some of missing subroutines will be just listed in later section.

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)
```

Supported version

This subroutine is supported in AIX 5.1 and later versions.

Parameters

name Contains a name identifying the first CPU for which statistics are desired. "" is used to indicate the first available CPU. For example: cpu0, cpu1, and so on.

userbuff Points to the memory area that is to be filled with one or more perfstat_cpu_t structures.

sizeof_struct Specifies the size of the perfstat_cpu_t structure: sizeof(perfstat_cpu_t).

desired_number Specifies the number of perfstat_cpu_t structures to copy to userbuff.

Example

Example 10-4 shows a sample code that uses the perfstat_cpu_t structure to obtain information about CPU statistics.

Example 10-4 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  #ifdef _AIX530
28                printf("\tpsys : %llu\n", ub[i].psys);
29                printf("\tpuser : %llu\n", ub[i].puser);
30                printf("\tpidle : %llu\n", ub[i].pidle);
31                printf("\tpwait : %llu\n", ub[i].pwait);
32                printf("\trunqueue : %llu\n", ub[i].runque);
33                printf("\tdevintrs : %llu\n", ub[i].devintrs);
34                printf("\tsoftintrs : %llu\n", ub[i].softintrs);
35  #endif
36          }
37  }
```

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 (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. This code also contains newly added fields in *perfstat_cpu_t* structure. You can see this part from line 28 to the end of this code. With the AIX Version 5.3 and the later, you can retrieve these values from the system. In order to do so, you need to specify -D_AIX530 option with **cc** when compiling the code.

The output from the program is shown in Example 10-5.

Example 10-5 Sample output from the perfstat_cpu_t program

```
# perfstat_cpu_t
name    : cpu0
        user    : 143183
        sys     : 44509
        idle    : 82268216
        wait    : 3032
        pswitch : 111623167
        syscall : 9895110
        sysread : 857795
        syswrite: 176179
        sysfork : 3796
        sysexec : 4659
        readch  : 610497259
        writech : 10252878
        psys : 106205944754
        puser : 290208972941
        pidle : 182043749379
        pwait : 21854685
        runqueue : 3
        devintrs : 99015
        softintrs : 3331579
name    : cpu1
        user    : 114661
        sys     : 28145
        idle    : 82322323
        wait    : 36
        pswitch : 399428
        syscall : 134210
        sysread : 9696
        syswrite: 1485
        sysfork : 688
        sysexec : 50
        readch  : 5696737
        writech : 120190
        psys : 48027180264
        puser : 232364642278
        pidle : 143862899898
        pwait : 5398361
        runqueue : 0
        devintrs : 98044
        softintrs : 41197334
```

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
puser	Physical CPU user time (raw ticks, Only in AIX5.3)
psys	Physical CPU sys time (raw ticks, only in AIX5.3)
pidle	Physical CPU idle time (raw ticks, only in AIX5.3)
pwait	Physical CPU wait time (raw ticks, only in AIX5.3)
runque	Number of threads on the runque (Only in AIX5.3)
devintrs	number of device interrupts (Only in AIX5.3)
softintrs	number of offlevel handlers called (Only in AIX5.3)

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)
```

Supported version

This subroutine is supported in AIX 5.1 and later versions.

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 sample code shown in Example 10-6 uses the perfstat_cpu_total_t structure to obtain information about CPU statistics.

Example 10-6 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 #ifdef _AIX530
33            printf("psys        : %llu\n", ub.psys);
34            printf("puser       : %llu\n", ub.puser);
35            printf("pidle       : %llu\n", ub.pidle);
36            printf("pwait       : %llu\n", ub.pwait);
37>#endif
36     }
37 }
```

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. This code also contains newly added fields in *perfstat_cpu_total_t* structure. You can see this part from line 32 to the end of this code. With the AIX Version 5.3 and the later, you can retrieve these values from the system. In order to do this you need to specify -D_AIX530 option with compilation command **cc**. The output from of this program is shown in Example 10-7.

Example 10-7 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
writech     : 21921375932
devintrs    : 0
softintrs   : 0
lbolt       : 45567458
loadavg TO  : 132391
loadavg T-5 : 132552
loadavg T-15: 132367
runque      : 295136
swpque      : 385398
```

The following list contains the 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
writech	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
puser	Physical CPU user time (raw ticks, only in AIX53)
psys	Physical CPU sys time (raw ticks, only in AIX53)
pidle	Physical CPU idle time (raw ticks, only in AIX53)
pwait	Physical CPU wait time (raw ticks, only in AIX53)

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)
```

Supported version

This subroutine is supported in AIX 5.1 and later versions.

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 10-8 on page 597 uses the perfstat_memory_total_t structure to obtain information about memory statistics.

Example 10-8 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,
sizeof(perfstat_memory_total_t),1) >= 0) {
8            printf("virt_total: %llu\n", ub.virt_total);
9            printf("real_total: %llu\n", ub.real_total);
10           printf("real_free : %llu\n", ub.real_free);
11           printf("real_inuse: %llu\n", ub.real_inuse);
12           printf("pgbad     : %llu\n", ub.pgbad);
13           printf("pgexct    : %llu\n", ub.pgexct);
14           printf("pgins     : %llu\n", ub.pgins);
15           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 of this program is shown in Example 10-9.

Example 10-9 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:

`virt_total`	Total virtual memory (4K pages)
`real_total`	Total real memory (4K pages)
`real_free`	Free real memory (4K pages)
`real_pinned`	Real memory that is pinned (4K pages)
`real_inuse`	Real memory that is in use (4K pages)
`pgbad`	Count of bad pages
`pgexct`	Count of page faults
`pgins`	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_pagingspace

The perfstat_pagingspace retrieves individual paging space usage statistics.

```
perfstat_id_t *name;
perfstat_pagingspace_t *userbuff;
size_t sizeof_struct;
int desired_number;

int perfstat_pagingspace (name, userbuff, sizeof_struct, desired_number)
```

Supported version

This subroutine is supported in AIX 5.2 and later versions.

Parameters

Name	Contains either “”, FIRST_PAGINGSPACE, or a name identifying the first paging space for which statistics are desired. For example: paging00, hd6, ...

userbuff	Points to the memory area to be filled with one or more perfstat_pagingspace_t structures.
sizeof_struct	Specifies the size of the perfstat_pagingspace_t structure: sizeof(perfstat_pagingspace_t)
desired_number	Specifies the number of perfstat_pagingspace_t structures to copy to userbuff.

Example

The code in Example 10-10 uses the perfstat_pagingspace_t structure to obtain information about memory statistics.

Example 10-10 Sample perfstat_pagingspace program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>
4  int
5  main(int argc, char agrv[])
6  {
7          int             i, ret, tot;
8          perfstat_id_t   first;
9          perfstat_pagingspace_t *pinfo;
10          tot = perfstat_pagingspace(NULL, NULL,
sizeof(perfstat_pagingspace_t), 0);
11          pinfo = calloc(tot, sizeof(perfstat_pagingspace_t));
12          strcpy(first.name, FIRST_PAGINGSPACE);
13          ret = perfstat_pagingspace(&first, pinfo,
sizeof(perfstat_pagingspace_t), tot);
14          for (i = 0;
15               i < ret;
16               i++) {
17                  printf("\nStatistics for paging space : %s\n",
pinfo[i].name);
18                  printf("---------------------------\n");
19                 printf("type : %s\n", pinfo[i].type == LV_PAGING ? "logical
volume" : "NFS file");
20                  if (pinfo[i].type == LV_PAGING) {
21                          printf("volume group : %s\n",
pinfo[i].u.lv_paging.vgname);
22                  } else {
23                          printf("hostname : %s\n",
pinfo[i].u.nfs_paging.hostname);
24                          printf("filename : %s\n",
pinfo[i].u.nfs_paging.filename);
25                  } printf("size (in LP) : %llu\n", pinfo[i].lp_size);
26                  printf("size (in MB) : %llu\n", pinfo[i].mb_size);
27                  printf("used (in MB) : %llu\n", pinfo[i].mb_used);
28          }
```

```
29  }
```

On line 3 the libperfstat.h declaration file is included. Then on line 6 and 7 we declare variables for calling the `perfstat_pagingspace_total` subroutine, which we do on line 13. Note how the usage and reference of structures is done in the call. The output of this program is shown in Example 10-11.

Example 10-11 Sample output from the perfstat_pagingspace program

```
Statistics for paging space : hd6
--------------------------
type : logical volume
volume group : rootvg
size (in LP) : 64
size (in MB) : 512
used (in MB) : 4
```

These are definitions of each structure element:

`type`	type of paging device (LV_PAGING or NFS_PAGING) Possible values are: LV_PAGING logical volume NFS_PAGING NFS file
`lp_size`	size in number of logical partitions
`mb_size`	size in megabytes
`mb_used`	portion used in megabytes
`io_pending`	number of pending I/O
`active`	indicates if active (1 if so, 0 if not)
`automatic`	indicates if automatic (1 if so, 0 if not)

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: hdisk0, hdisk1, and so on.
userbuff	Points to the memory area that is to be filled with one or more perfstat_disk_t structures.
sizeof_struct	Specifies the size of the perfstat_disk_t structure; sizeof(perfstat_cpu_t).
desired_number	Specifies the number of perfstat_disk_t structures to copy to userbuff.

Example

The code in Example 10-12 uses the perfstat_disk_t structure to obtain information about disk statistics.

Example 10-12 Sample perfstat_disk_t program

```
#include <stdio.h>
#include <stdlib.h>
#include <libperfstat.h>

main()
{
    perfstat_id_t   name;
    perfstat_disk_t *ub;
    int             ndisk,i;

    ndisk = perfstat_disk (NULL,NULL,sizeof(perfstat_disk_t),0);
    ub = malloc(sizeof(perfstat_disk_t)*ndisk);

    strcpy(name.name,"");

    if (perfstat_disk (&name,ub,sizeof(perfstat_disk_t),ndisk) >= 0)
        for (i = 0; i < ndisk; i++) {
            printf("name       : %s\n",  ub[i].name);
            printf("\tdescription: %s\n",  ub[i].description);
            printf("\tvgname     : %s\n",  ub[i].vgname);
            printf("\tsize       : %llu\n", ub[i].size);
            printf("\tfree       : %llu\n", ub[i].free);
            printf("\tbsize      : %llu\n", ub[i].bsize);
            printf("\txrate      : %llu\n", ub[i].xrate);
            printf("\txfers      : %llu\n", ub[i].xfers);
            printf("\twblks      : %llu\n", ub[i].wblks);
            printf("\trblks      : %llu\n", ub[i].rblks);
            printf("\tqdepth     : %llu\n", ub[i].qdepth);
            printf("\ttime       : %llu\n", ub[i].time);
        }
```

```
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 of this program is shown in Example 10-13.

Example 10-13 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)
```

Supported version

This subroutine is supported in AIX 5.2 and later versions.

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 perfstat_disk_total_t structures.
sizeof_struct	Specifies the size of the perfstat_disk_total_t structure; sizeof(perfstat_cpu_t).
desired_number	In AIX 5.2, this must always be set to 1.

Example

The code in Example 10-14 uses the perfstat_disk_total_t structure to obtain information about disk statistics.

Example 10-14 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 of this program is shown in Example 10-15.

Example 10-15 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_diskadapter

The perfstat_diskadapter subroutine retrieves one or more individual diskadapter usage statistics. The same function can also be used to retrieve the number of available sets of diskadapter statistics.

```
perfstat_id_t *name;
perfstat_diskadapter_t *userbuff;
size_t sizeof_struct;
int desired_number;
```

```
int perfstat_diskadapter (name, userbuff, sizeof_struct, desired_number)
```

Supported version

This subroutine is supported in AIX 5.2 and later versions.

Parameters

name	Contains either "", FIRST_DISKADAPTER, or a name identifying the first disk adapter for which statistics are desired. For example: scsi0, scsi1, ...
userbuff	Points to the memory area to be filled with one or more perfstat_diskadapter_t structures.
sizeof_struct	Specifies the size of the perfstat_diskadapter_t structure: sizeof(perfstat_diskadapter_t)
desired_number	Specifies the number of perfstat_diskadapter_t structures to copy to userbuff.

Example

The code in Example 10-16 uses the perfstat_diskadapter_t structure to obtain information about disk statistics.

Example 10-16 Sample perfstat_diskadapter_t program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>
4  int
5  main(int argc, char *argv[])
6  {
7   int i, ret, tot;
8   perfstat_diskadapter_t *statp;
9   perfstat_id_t   first;
10  /* check how many perfstat_diskadapter_t structures are available */
11  tot = perfstat_diskadapter(NULL, NULL, sizeof(perfstat_diskadapter_t), 0);
12  /* allocate enough memory for all the structures */
13  statp = calloc(tot, sizeof(perfstat_diskadapter_t));
14  /* set name to first interface */
15 strcpy(first.name, FIRST_DISK);
16  /*
17  * ask to get all the structures available in one call
18  */
19  /* return code is number of structures returned */
20  ret = perfstat_diskadapter(&first, statp, sizeof(perfstat_diskadapter_t),
tot);
21  /* print statistics for each of the disk adapters */
22  for (i = 0;
24  i < ret;
```

```
25  i++) {
26  printf("\nStatistics for adapter : %s\n", statp[i].name);
27  printf("----------------------\n");
28  printf("description : %s\n", statp[i].description);
29  printf("number of disks connected : %d\n", statp[i].number);
29  printf("total disk size : %llu MB\n", statp[i].size);
30  printf("total disk free space : %llu MB\n", statp[i].free);
31  printf("number of blocks read : %llu\n", statp[i].rblks);
32  printf("number of blocks written : %llu\n", statp[i].wblks);
34  }
35 }
```

On line 3 the libperfstat.h declaration file is included. Then on line 8 and 9 we declare variables for calling the perfstat_diskadapter subroutine, which we do on line 20. Note how the usage and reference of structures is done in the call.The output of this program is shown in Example 10-17.

Example 10-17 Sample output from the perfstat_diskadapter_t program

```
# perfstat_diskadapter_t
Statistics for adapter : ide0
----------------------
description : ATA/IDE Controller Device
number of disks connected : 1
total disk size : 0 MB
total disk free space : 0 MB
number of blocks read : 0
number of blocks written : 0

Statistics for adapter : scsi0
----------------------
description : Wide/Ultra-3 SCSI I/O Controller
number of disks connected : 3
total disk size : 174464 MB
total disk free space : 120000 MB
number of blocks read : 23323
number of blocks written : 5448
```

These are definitions of each structure element as displayed above.

number	number of disks connected to adapter
size	total size of all disks (in MB)
free	free portion of all disks (in MB)
xrate	total kbytes/sec xfer rate capability
xfers	total number of transfers to/from disk
rblks	512 bytes blocks written via adapter
wblks	512 bytes blocks read via adapter

`time`	amount of time disks are active

perfstat_diskpath

The perfstat_diskpath subroutine retrieves one or more individual diskpath usage statistics. The same function can also be used to retrieve the number of available sets of diskpath statistics. This subroutine can be used for mpio environment.

```
perfstat_id_t *name;
perfstat_diskpath_t *userbuff;
size_t sizeof_struct; i
nt desired_number;

int perfstat_diskpath (name, userbuff, sizeof_struct, desired_number)
```

Parameters

name	Contains either "", FIRST_DISKPATH, a name identifying the first disk path for which statistics are desired, or a name identifying a disk for which path statistics are desired. For example: hdisk0_Path2, hdisk1_Path0, ... or hdisk5 (equivalent to hdisk5_Pathfirstpath)
userbuff	Points to the memory area to be filled with one or more perfstat_diskpath_t structures.
sizeof_struct	Specifies the size of the perfstat_diskpath_t structure: sizeof(perfstat_diskpath_t)
desired_number	Specifies the number of perfstat_diskpath_t structures to copy to userbuff.

Supported version

This subroutine is supported in AIX 5.2 and later versions.

Example

The code in Example 10-18 uses the perfstat_diskpath structure to obtain information about disk statistics.

Example 10-18 Sample perfstat_diskadapter_t program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>
4
5          main()
6          {
7              perfstat_id_t   name;
8              perfstat_diskpath_t *ub;
```

```
9               int             ndisk,i;
10
11               ndisk = perfstat_diskpath
(NULL,NULL,sizeof(perfstat_diskpath_t),0);
12               ub = malloc(sizeof(perfstat_diskpath_t)*ndisk);
13
14               strcpy(name.name,"");
15
16               if (perfstat_diskpath
(&name,ub,sizeof(perfstat_diskpath_t),ndisk) >= 0)
17                   for (i = 0; i < ndisk; i++) {
18                       printf("name       : %s\n",  ub[i].name);
19                       printf("\txrate      : %llu\n", ub[i].xrate);
20                       printf("\txfers      : %llu\n", ub[i].xfers);
21                       printf("\twblks      : %llu\n", ub[i].wblks);
22                       printf("\trblks      : %llu\n", ub[i].rblks);
23                       printf("\ttime       : %llu\n", ub[i].time);
24                       printf("\tadapter    : %s\n", ub[i].adapter);
25                   }
26          }
```

On line 3 the libperfstat.h declaration file is included. Then on lines 7 and 8 we declare variables for calling the `perfstat_diskpath` subroutine, which we do on line 16. Note how the usage and reference of structures is done in the call. The first call to `perfstat_diskpath` is done to acquire the number of available sets of diskpath (mpio paths) 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 of this program is shown in Example 10-19.

Example 10-19 Sample output from the perfstat_diskpath_t program

```
# perfstat_diskpath_t
name       : hdisk0_Path0
        xrate      : 0
        xfers      : 0
        wblks      : 0
        rblks      : 0
        time       : 0
        adapter    : scsi0
name       : hdisk1_Path0
        xrate      : 0
        xfers      : 492
        wblks      : 4856
        rblks      : 8
        time       : 0
        adapter    : scsi0
```

These are definitions of each structure element as displayed above.

xrate	total kbytes/sec xfer rate capability
xfers	total number of transfers via the path
rblks	512 bytes blocks written via the path
wblks	512 bytes blocks read via the path
time	amount of time disks are 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 perfstat_netinterface_t structures.
sizeof_struct	Specifies the size of the perfstat_netinterface_t structure; sizeof(perfstat_cpu_t).
desired_number	Specifies the number of perfstat_netinterface_t structures to copy to userbuff.

Supported version

This subroutine is supported in AIX 5.1 and later versions.

Example

The code in Example 10-20 uses the perfstat_netinterface_t structure to obtain information about network statistics.

Example 10-20 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("\tibytes     : %llu\n", ub[i].ibytes);
20              printf("\tierrors    : %llu\n", ub[i].ierrors);
21              printf("\topackets   : %llu\n", ub[i].opackets);
22              printf("\tobytes     : %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 of this program is shown in Example 10-21.

Example 10-21 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 10-20 on page 610. Note that the `type` value of 9, in the output above for token-ring, translates in hex to `IS088025` or token-ring (see Table 10-1).

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 10-1
`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 10-1 Interface types from if_types.h

Name	Type	Name	Type
1822	0x2	DS3	0x1e
HDH1822	0x3	SIP	0x1f
X25DDN	0x4	FRELAY	0x20

Name	Type	Name	Type
X25	0x5	RS232	0x21
ETHER	0x6	PARA	0x22
OTHER	0x1	ULTRA	0x1d
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)
```

Supported version

This subroutine is supported in AIX 5.1 and later versions.

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 10-22 uses the perfstat_netinterface_total_t structure to obtain information about CPU statistics.

Example 10-22 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 of this program is shown in Example 10-23.

Example 10-23 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 in previous example:

`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

perfstat_partition

The pefstat_partition_total subroutine returns global partition usage statistics

```
perfstat_id_t *name;
perfstat_partition_total_t *userbuff;
size_t sizeof_struct;
int desired_number;

int perfstat_partition_total(name, userbuff, sizeof_struct, desired_number)
```

Supported version

This subroutine is supported in AIX 5.3 and later versions.

Parameters

name	Must be set to NULL.
userbuff	Points to the memory area to be filled with the perfstat_partition_total_t structures.
sizeof_struct	Specifies the size of the perfstat_partition_total_t structure: sizeof(perfstat_partition_total_t).
desired_number	Must be set to 1

Example

The code in Example 10-24 uses the perfstat_partition structure to obtain information about partition statistics.

Example 10-24 Sample perfstat_partition_t program

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <libperfstat.h>
4  int
5  main(int argc, char *argv[])
6  {
7           perfstat_partition_total_t pinfo;
8           int             rc;
9           rc = perfstat_partition_total(NULL, &pinfo,
sizeof(perfstat_partition_total_t), 1);
10           if (rc != 1) {
11                   perror("Error in perfstat_partition_total");
12                   exit(-1);
13          }
14          printf("Partition Name : %s\n", pinfo.name);
15          printf("Partition Number : %u\n", pinfo.lpar_id);
16          printf("Type : %s\n", pinfo.type.b.shared_enabled ? "Shared" :
"Dedicated");
17          printf("Mode : %s\n", pinfo.type.b.capped ? "Capped" : "Uncapped");
18          printf("Entitled Capacity : %u\n", (double)
pinfo.entitled_proc_capacity / (double) 100.0);
19          printf("Partition Group-ID : %u\n", pinfo.group_id);
20          printf("Shared Pool ID : %u\n", pinfo.pool_id);
21          printf("Online Virtual CPUs : %u\n", pinfo.online_cpus);
22          printf("Maximum Virtual CPUs : %u\n", pinfo.max_cpus);
23          printf("Minimum Virtual CPUs : %u\n", pinfo.min_cpus);
24          printf("Online Memory : %llu MB\n", pinfo.online_memory);
25          printf("Maximum Memory : %llu MB\n", pinfo.max_memory);
26          printf("Minimum Memory : %llu MB\n", pinfo.min_memory);
27          printf("Variable Capacity Weight : %u\n",
pinfo.var_proc_capacity_weight);
28          printf("Minimum Capacity : %u\n", pinfo.min_proc_capacity);
```

```
29          printf("Maximum Capacity : %u\n", pinfo.max_proc_capacity);
30          printf("Capacity Increment : %u\n", pinfo.proc_capacity_increment);
31          printf("Maximum Physical CPUs in system: %u\n",
pinfo.max_phys_cpus_sys);
32          printf("Active Physical CPUs in system : %u\n",
pinfo.online_phys_cpus_sys);
33          printf("Active CPUs in Pool : %u\n", pinfo.phys_cpus_pool);
34          printf("Unallocated Capacity : %u\n", pinfo.unalloc_proc_capacity);
35          printf("Physical CPU Percentage : %4.2f%%\n",
36          (double) pinfo.entitled_proc_capacity / (double)
pinfo.online_cpus);
37          printf("Unallocated Weight : %u\n",
pinfo.unalloc_var_proc_capacity_weight);
38  }
```

On line 3 the libperfstat.h declaration file is included. Then on line 7 we declare variables for calling the `perfstat_partition_total` subroutine, which we do on line 9. Note how the usage and reference of structures is done in the call. The output of this program is shown in Example 10-25.

Example 10-25 Sample output from the perfstat_partition_t program

```
#perfstat_partition_t
Partition Name : partition01
Partition Number : 1
Type : Dedicated
Mode : Uncapped
Entitled Capacity : 1070176665
Partition Group-ID : 32769
Shared Pool ID : 0
Online Virtual CPUs : 1
Maximum Virtual CPUs : 2
Minimum Virtual CPUs : 1
Online Memory : 512 MB
Maximum Memory : 1024 MB
Minimum Memory : 512 MB
Variable Capacity Weight : 128
Minimum Capacity : 10
Maximum Capacity : 100
Capacity Increment : 1
Maximum Physical CPUs in system: 2
Active Physical CPUs in system : 2
Active CPUs in Pool : 0
Unallocated Capacity : 0
Physical CPU Percentage : 20.00%
Unallocated Weight : 0
```

These are definitions of each structure element as displayed in previous example:

`type`	set of bits describing the partition
`lpar_id`	logical partition identifier
`group_id`	identifier of the LPAR group this partition is a member of
`pool_id`	identifier of the shared pool of physical processors this partition is a member of
`online_cpus`	number of virtual CPUs currently online on the partition
`max_cpus`	maximum number of virtual CPUs this partition can ever have
`min_cpus`	minimum number of virtual CPUs this partition must have
`online_memory`	amount of memory currently online
`max_memory`	maximum amount of memory this partition can ever have
`min_memory`	minimum amount of memory this partition must have
`entitled_proc_capacity`	number of processor units this partition is entitled to receive
`max_proc_capacity`	maximum number of processor units this partition can ever have
`min_proc_capacity`	minimum number of processor units this partition must have
`proc_capacity_increment`	increment value to the entitled capacity */
`unalloc_proc_capacity`	number of processor units currently unallocated in the shared processor pool this partition belongs to
`var_proc_capacity_weight`	partition priority weight to receive extra capacity
`unalloc_var_proc_capacity_weight`	number of variable processor capacity weight units currently unallocated in the shared processor pool this partition belongs to
`online_phys_cpus_sys`	number of physical CPUs currently active in the system containing this partition
`max_phys_cpus_sys`	maximum possible number of physical CPUs in the system containing this partition
`phys_cpus_pool`	number of the physical CPUs currently in the shared processor pool this partition belong to
`puser`	Physical CPU user time (raw ticks)
`psys`	Physical CPU sys time (raw ticks)
`pidle`	Physical CPU idle time (raw ticks)
`pwait`	Physical CPU wait time (raw ticks)

pool_idle_time	number of clock ticks a processor in the shared pool was idle
phantintrs	number of phantom interrupts received by the partition
invol_virt_cswitch	number involuntary virtual CPU context switches
vol_virt_cswitch	number voluntary virtual CPU context switches
timebase_last	most recently cpu time base

Makefile for Perfstat

Example 10-26 shows a makefile for compiling the perfstat sample programs.

Example 10-26 Makefile

```
# nl 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_diskada[ter_t perfstat_diskpath_t perfstat_disk_total_t
perfstat_memory_total_t perfstat_pagingspace_t perfstat_netinterface_t
perfstat_netinterface_total_t perfstat_partition_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. In line 2 you can specify compilation options. 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 Perfstat API subroutines that are not covered in previous section. Refer to “Perfstat API programming” section in the manual “*AIX 5L Version 5.3 Performance Tool Guide and Reference*”, SC23-4906-00 for examples of these subroutines and the description of libperstat.h file.

perfstat_protocol The subroutine retrieves protocol usage statistics such as ICMP, ICMPv6, IP, IPv6, TCP, UDP, RPC, NFS, NFSv2, NFSv3. This subroutine is available from AIX 5.2.

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. This subroutine is available from AIX 5.2.

perfstat_reset The perfstat_reset subroutine flushes the information cache for the library and should be called whenever the machine configuration has changed.This subroutine is available from AIX 5.2.

10.2 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, an 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 as prerequisite.

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 5.3 Technical Reference: Base Operating System and Extensions, Volume 2,* SC23-4914.

10.2.1 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.

10.2.2 Terms and concepts for SPMI

Due to the fact that SPMI has been developed as a part of AIX *Performance Toolbox (PTX)*, SPMI inherited many of its features from *PTX*. So, most of its terminology, concepts and data organization of SPMI are derived from *PTX*. In this section, we cover the terminology that represents the data types and structures used in SPMI programming. We also cover the relationship between those and it's organization as well.

Terminology

From now on, we will see some new terms, especially for SPMI. Here, we have a list of essential terms for SPMI and brief explanation of these terms.

Context *Context* indicates a set of system component such as CPU, disk, memory and so on. It also indicates an individual component such as cpu0, hdisk1, ent0 and so on. This can be regarded as an abstraction for the same type of system components or each of substantial system component at the same time. With definitions of system header file, *Cx* stands for *context*, most of the time.

Metric *Metric* describes a probe in or instrumentation of system component a.k.a *context* in SPMI world. It contains the statistical value of *context*. For context *cpu0*, the *metric* that can contain system value is *kern*.

Statistic *Statistic* is synonymous to the term *metric* in PTX and SPMI. But this term is more preferred in describing the APIs. With definitions of system header file, *Stat* stands for *statistic*, most of the time.

Instantiation When multiple copies of a resource (context) 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 know as *instantiation*. We can say the *subcontext* cpu0 is instantiated by using template of its *parent context* or cpu.

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.

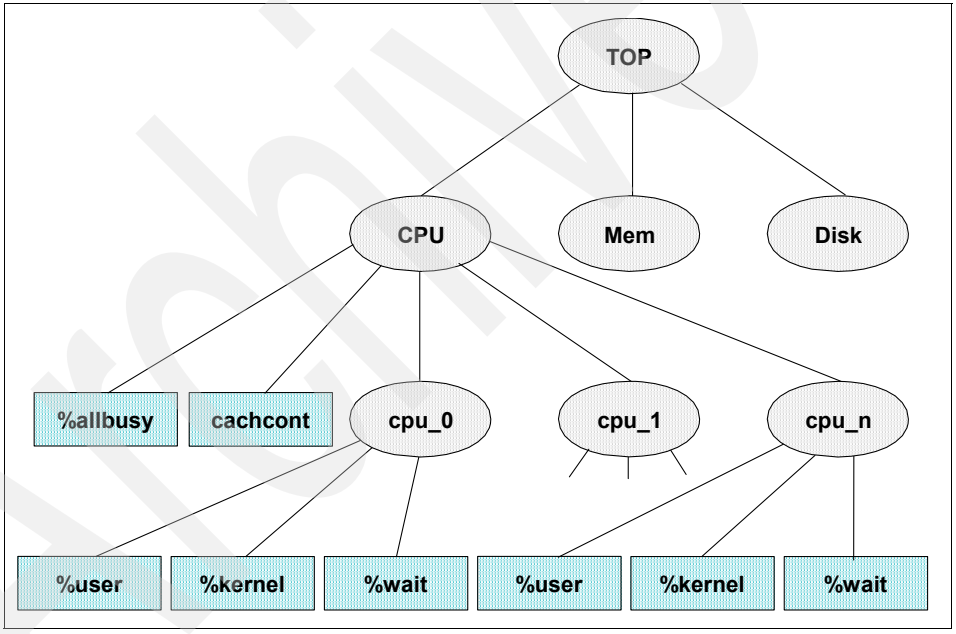

Figure 10-4 Sample Data Hierarchy for SPMI object

In Figure 10-4 each ellipse depicts a context or a subcontext; CPU, Memory, Disk and so on. Each rectangle depicts a metric (or statistic); %user, %kernel, %wait, and so on. In this case, cpu0 is a subcontext of parent context cpu. In other words, cpu0 context is a instance of cpu context.

Such a relationship between context, subcontext and metric can be expressed by the following example. This illustrates the SPMI data hierarchy for a metric:

```
CPU/cpu0/kern
```

The parents in the example above are `CPU` and `cpu0`, and the metric that can contain statistical value is `kern` (time spent executing in kernel mode). For more information about a list of available SPMI metrics, see also “Traversing and displaying the SPMI hierarchy” on page 635.

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.

Shared memory segment used for SPMI

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 **kill <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.

10.2.3 Subroutines

For a complete list of the SPMI API subroutines refer to *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 2,* SC23-4914.

To create a simple monitoring program using the SPMI API, the following subroutine sequence could be used to create a snapshot of the current values for specified statistics:

SpmiInit 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 SpmiStatVals 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 SpmiStatVals 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.

SpmiInit

The SpmiInit 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()
```

10.2.4 Basic layout of SPMI program

In this section, we will describe the basic layout of a SPMI program. To monitor the system using SPMI, we have to decide what kind of statistics (or metrics) to be monitored, define the statistics set, and run proper subroutines to get actual values for the defined sets from system. Finally, we have to print the retrieved values using appropriate subroutines. We use some pieces of codes from Source code, “spmi_dude.c” on page 679 and use those as sample. The basic layout of this code is illustrated in the Example 10-27. The complete source code will be provided as well.

Example 10-27 Basic layout of SPMI programs

```
main ()
{
   /* Initialization stage. Prepare shared memory area for program */
   SpmiInit()
   /*Define monitoring statistic set*/
   SpmiCreateStatSet()
   SpmiAddSetStat()
   /*Retrieve monitoring data*/
   SpmiGetStatSet()
   /*Traverse output data structure*/
   SpmiFirstVal () or SpmiNextVal()
   SpmiGetValue()
   /*Termination stage. Decrease the usage count for shared memory*/
   SpmiExit()
}
```

Decide which statistics (or metric) to be monitored

SPMI provides almost every performance items that can be monitored in AIX. Appendix A, “Spmi_traverse.c” on page 691 provides a complete list of available statistics. You can refer to the output of this program and choose the statistics you want to monitor (see Example 10-35 on page 636).

In this case, we choose some statistics and it is listed in following Example 10-28. This list is assigned to the string array *stat[]*.

Example 10-28 Statistics to be monitored

```
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
};
```

Define statistics set

With the statistics decided in the previous example, you need to define a structure for the set of statistics (statset). A *SpmiStatSet* structure defined in Spmidef.h can be used for this purpose. With this declared structure *SPMIset*, statistics from string array *stat[]* will be added by using ***SpmiPathAddSetStat()*** subroutine.

Example 10-29 Defining SpmiStatSet structure and adding statistics

```
   struct SpmiStatSet*SPMIset = NULL;
...(lines omitted)...

   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);
    }
```

Run the subroutine to collect data

With declared statset *SPMIset*, you can run *SpmiGetStatset ()* subroutine. At this point of subroutine, you will retrieve the actual performance data.

Example 10-30 Retrieve performance data using SpmiGetStatSet () subroutine

```
      if ((SpmiGetStatSet(SPMIset,TRUE)) != 0) {
```

Print out value from the result data structure

Example 10-31 shows the data structure resulted from running the SpmiGetStatSet() subroutine.

Example 10-31 The data structure result of SpmiGetStatSet() subroutine

```
      struct SpmiStatVals*SPMIval = NULL;
      struct SpmiStat*SPMIstat = NULL;
....(lines omitted)....
      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");
```

The execution result of this subroutine (Example 10-30 on page 630) is stored in the special SPMI data structure. In this data structure, *SPMIStatSet* structure plays a role of anchor point. This means the structure itself doesn't contain any data but you can find structures containing actually data by using proper subroutines. Figure 10-5 explains the relationship between the data structure and the subroutines which are used for traversing this data structure. In Figure 10-5, the system-defined keywords for the data structure and subroutines are in *italic* font and the keywords for the declared variables are using default (normal) font.

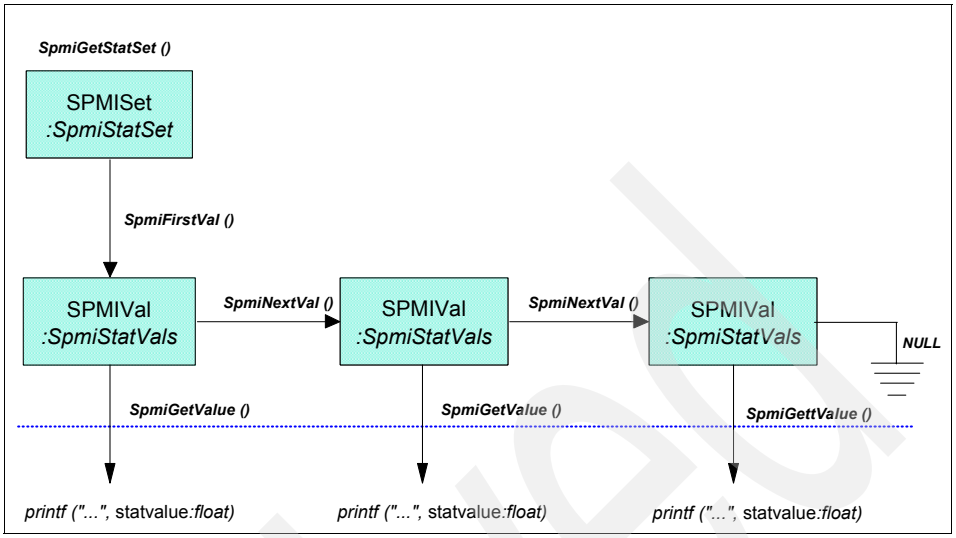

Figure 10-5 Traversing the data structure which is result of SpmiGetStatSet() subroutine

You can find complete source code of this example in Appendix A, in "spmi_dude.c" on page 679. Detailed execution results for this program are covered in next section.

10.2.5 SPMI examples

In this section we present three examples (programs) that use the SPMI API:

- "Hard-coded metrics" on page 632 uses a hard-coded array to store the hierarchical names of the metrics we want to collect statistics about.
- "Reading metrics from a file" on page 633 reads the metrics from a file.
- "Traversing and displaying the SPMI hierarchy" on page 635 traverses the SPMI hierarchy and displays all metrics.

Hard-coded metrics

This example uses the spmi_dude program given in Appendix A, in "spmi_dude.c" on page 679. This shows how the SPMI environment can be set up to collect and display statistics. Example 10-32 contains a sample output created by the **spmi_dude** program.

Example 10-32 Sample output from the spmi_dude program

```
#spmi_dude 1 10
swpq  runq pgspo 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 10-2 explains the values shown in the columns in the previous output for the **spmi_dude** program.

Table 10-2 Column explanation

Column	SPMI metric	SPMI description
wa	CPU/glwait	System-wide time waiting for I/O (percent)
id	CPU/glidle	System-wide time CPU is idle (percent)
sy	CPU/glkern	System-wide time executing in kernel mode (percent)
us	CPU/gluser	System-wide 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
%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/swpque	Average count of processes waiting to be paged in

Reading metrics from a file

The program in Appendix A, “spmi_file.c” on page 689 shows how to set up the SPMI environment to collect and display statistics after reading the SPMI metrics from a file. Example 10-33 displays a sample output created by the **spmi_file** program shown in the previous example.

Example 10-33 Sample output from the spmi_file program

```
# spmi_file|pr -t -2
IP/NetIF/en0/oerror      : 0        Mem/Virt/pgspgin         : 0
IP/NetIF/en0/ooctet_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/writech_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/swpque              : 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/glidle               : 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 (8 GB).

Example 10-34 shows the input file used with the **spmi_file** program to create the output presented in Example 10-33 on page 634.

Example 10-34 Sample input file SPMI_METRICS

```
CPU/gluser
CPU/glkern
CPU/glwait
CPU/glidle
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/writech_kb
IP/NetIF/en0/ipacket
IP/NetIF/en0/ioctet_kb
IP/NetIF/en0/ierror
IP/NetIF/en0/opacket
IP/NetIF/en0/ooctet_kb
IP/NetIF/en0/oerror
```

Traversing and displaying the SPMI hierarchy

The program in Appendix A, “Spmi_traverse.c” on page 691, shows how to set up the SPMI environment, and then how to traverse and display all metrics found

in the SPMI hierarchy. Example 10-35 shows the sample output created by the **spmi_traverse** program.

Example 10-35 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/glidle:Systemwide time CPU is idle (percent):Float/Quantity:0-100
CPU/gluticks:Systemwide CPU ticks executing in user mode:Long/Counter:0-100
CPU/glkticks: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 mknode 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 10-36 shows what a makefile would look like for all of the programs described above.

Example 10-36 Makefile

```
# nl 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.

10.3 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, which 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 10-39 on page 641.

The documentation for the subroutines can be found in the *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 1*, SC23-4913, and the *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155.

10.3.1 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 want 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 5.3 General Programming Concepts*, SC23-4896
- *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 1*, SC23-4913
- *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155

Also, refer to the following Web site:

http://www.austin.ibm.com/tech/monitor.html

10.3.2 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 shown in Example 10-37.

Example 10-37 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.

10.3.3 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 system-wide PM programmation.
pm_get_program	Retrieves system-wide PM settings.
pm_delete_program	Deletes previously established system-wide PM settings.
pm_start	Starts system-wide PM counting.
pm_stop	Stops system-wide PM counting.
pm_get_data	Returns system-wide PM data.
pm_reset_data	Resets system-wide PM data.

For a detailed description of the subroutines, read the *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 1*, SC23-4913.

10.3.4 PM API examples

A program using the PM API usually consists of three parts:

- Initialization
- Monitoring
- Reporting

Example 10-38 shows the basic layout of a program that uses the PM API.

Example 10-38 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 10-39 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 10-39 Sample pmapi_list.c program for displaying available events

```
#include <sys/types.h>
#include <pmapi.h>

main(int argc, char *argv[])
{
    static pm_info_t    pminfo;
    static pm_events_t  *pmeventp;
    static int          i,j,rc;

    if ((rc = pm_init(PM_VERIFIED|PM_UNVERIFIED|PM_CAVEAT, &pminfo)) > 0) {
        pm_error("pm_init", rc);
        exit(-1);
    }

    for (i = 0; i < pminfo.maxpmcs; i++) {
        pmeventp = pminfo.list_events[i];
        for (j = 0; j < pminfo.maxevents[i]; j++, pmeventp++) {
            printf("proc name  : %s\n",pminfo.proc_name);
            printf("event id   : %d\n",pmeventp->event_id);
            printf("status     : %c\n",pmeventp->status);
            printf("threshold  : %c\n",pmeventp->threshold);
            printf("short name : %s\n",pmeventp->short_name);
            printf("long name  : %s\n",pmeventp->long_name);
            printf("description: %s\n",pmeventp->description);
        }
    }
}
```

Example 10-40 shows the sample output from the **pmapi_list** program shown in Example 10-39 on page 641.

Example 10-40 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, see *AIX 5L Version 5.3 Performance Tools Guide and Reference*, SC23-4906. 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

10.3.5 PMAPI M:N pthreads support

AIX Version 5.3 start to support M:N threading Model. Under M:N threading model, M user threads are mapped to N kernel threads, with M typically being considerably bigger than N to allow large numbers of pthreads to run. Making PMAPI calls from a program running in this mode was previously not supported.

The PMAPI library has been updated by internal changes to handle the M:N thread model, as the current unchanged interfaces simply work in M:N mode.

The only significant change is for third party API callers, for example debuggers, where new interfaces with *pid*, *tid*, and *ptid* must be used.

10.4 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 5.3 Technical Reference: Base Operating System and Extensions, Volume 1,* SC23-4913, and *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 2,* SC23-4914.

10.4.1 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.

10.4.2 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 10-41 uses the `vario` structure to obtain information about the run-time operating system parameters.

Example 10-41 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 10-42 shows the output from the program in previous example.

Example 10-42 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 10-43 uses the vminfo structure to obtain information about certain VMM parameters.

Example 10-43 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)                    :
%lld\n",vminfo.pgexct);
        printf("vminfo.pgrclm (count of page reclaims)                  :
%lld\n",vminfo.pgrclm);
        printf("vminfo.lockexct (count of lockmisse)                    :
%lld\n",vminfo.lockexct);
        printf("vminfo.backtrks (count of backtracks)                   :
%lld\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 10-44 shows sample output from the previous program.

Example 10-44 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.numsios (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.pendiowts (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.memsizepgs (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 10-45 uses the pginfo structure to obtain information about active paging and swap devices.

Example 10-45 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);
            }
        }
    pclose(file);
}
main()
{
    swapqry_();
```

```
}
```

Example 10-46 shows the output from the program in Example 10-45 on page 652.

Example 10-46 Sample output from the swapqry subroutine program

```
pagingspace                      : /dev/hd6
devno (device number)            : 655362
size (size in PAGESIZE blocks)   : 262144
free  (# of free PAGESIZE 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 10-47 uses the statstime structure to obtain statistics from the remote host specified in the `host` variable.

Example 10-47 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_swtch      : %d\n",statstime.v_swtch);
        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 and the inetd subsystem refreshed for the rstat subroutine to work:

```
rstatd sunrpc_udp udp wait root /usr/sbin/rpc.rstatd rstatd 100001 1-3
```

Example 10-48 shows the output from running the program in Example 10-47 on page 653.

Example 10-48 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_pgpgin     : 469012
v_pgpgout    : 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_swtch      : 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 procentry64 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 10-49 on page 656 uses the procsinfo structure to obtain information about processes.

Example 10-49 Using getprocs

```
#include <procinfo.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 %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] == '') 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 %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 10-50 shows the output from running the example program above.

Example 10-50 Sample output from the getprocs subroutine program

```
total # 65       cmd state   pid  ppid   uid  nice #thrd io/4k  size %real  io/b
             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 10-51 uses the wlm_info structure to obtain information about characteristics of the active WLM classes.

Example 10-51 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 *wlminfo;
    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 10-52 shows how the output of the example program above may look.

Example 10-52 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 10-53 uses the wlm_bio_dev_info_t structure to obtain information about WLM disk I/O statistics.

Example 10-53 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 10-54 shows what the output of the program above would look like.

Example 10-54 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.

10.4.3 Combined example

The dudestat.c program in Appendix A, “Source code” on page 665 illustrates how the different subroutines could be used together. Sample output of the dudestat program is shown in Example 10-55.

Example 10-55 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
```

A

Source code

This appendix contains source code that was used to create the examples for these sections of this book:

- The perfstat_dude.c program in 10.1, “The performance status (Perfstat) API” on page 584.
- The programs spmi_dude.c, spmi_data.c, spmi_file.c, and spmi_traverse.c in 10.2, “System Performance Measurement Interface” on page 620.
- The dudestat.c program in 10.4, “Miscellaneous performance monitoring subroutines” on page 644.

perfstat_dump_all.c

Example A-1 shows how to combine all examples from 10.1.3, “Subroutines” on page 587 to access data provided by AIX 5.3 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 5.3 Perfstat API complete example

```
#include <stdio.h>
#include <stdlib.h>
#include <libperfstat.h>

cpu()
{
    perfstat_id_t   name;
    perfstat_cpu_t  *ub;
    int             ncpu,i;

    ncpu = perfstat_cpu (NULL,NULL,sizeof(perfstat_cpu_t),0);
    ub = malloc(sizeof(perfstat_cpu_t)*ncpu);

    strcpy(name.name,"");

    if (perfstat_cpu(&name,ub,sizeof(perfstat_cpu_t),ncpu) >= 0)
        for (i = 0; i < ncpu; i++) {
            printf("name    : %s\n",   ub[i].name);
            printf("\tuser    : %llu\n", ub[i].user);
            printf("\tsys     : %llu\n", ub[i].sys);
            printf("\tidle    : %llu\n", ub[i].idle);
            printf("\twait    : %llu\n", ub[i].wait);
            printf("\tpswitch : %llu\n", ub[i].pswitch);
            printf("\tsyscall : %llu\n", ub[i].syscall);
            printf("\tsysread : %llu\n", ub[i].sysread);
            printf("\tsyswrite: %llu\n", ub[i].syswrite);
            printf("\tsysfork : %llu\n", ub[i].sysfork);
            printf("\tsysexec : %llu\n", ub[i].sysexec);
            printf("\treadch  : %llu\n", ub[i].readch);
            printf("\twritech : %llu\n", ub[i].writech);
        }
}

cpu_total()
{
    perfstat_cpu_total_t    ub;

    if (perfstat_cpu_total((perfstat_id_t*)NULL, &ub,
sizeof(perfstat_cpu_total_t), 1) >= 0) {
        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 }

120 netinterface()
```

```
{
    perfstat_id_t           name;
    perfstat_netinterface_t *ub;
    int                     nnetinterface,i;

    nnetinterface = perfstat_netinterface (NULL,NULL,
sizeof(perfstat_netinterface_t), 0);
    ub = malloc(sizeof(perfstat_netinterface_t)*nnetinterface);

    strcpy(name.name,"");

    if (perfstat_netinterface (&name,ub, sizeof(perfstat_netinterface_t),
nnetinterface) >= 0)
        for (i = 0; i < nnetinterface; i++) {
            printf("name       : %s\n",     ub[i].name);
            printf("\tdescription: %s\n",   ub[i].description);
            printf("\ttype       : %u\n",   ub[i].type);
            printf("\tmtu        : %llu\n", ub[i].mtu);
            printf("\tipackets   : %llu\n", ub[i].ipackets);
            printf("\tibytes     : %llu\n", ub[i].ibytes);
            printf("\tierrors    : %llu\n", ub[i].ierrors);
            printf("\topackets   : %llu\n", ub[i].opackets);
            printf("\tobytes     : %llu\n", ub[i].obytes);
            printf("\toerrors    : %llu\n", ub[i].oerrors);
            printf("\tcollisions : %llu\n", ub[i].collisions);
        }
}

netinterface_total()
{
    perfstat_netinterface_total_t   ub;

    if (perfstat_netinterface_total ((perfstat_id_t*)NULL,&ub,
sizeof(perfstat_netinterface_total_t),1) >= 0) {
        printf("number    : %d\n", ub.number);
        printf("ipackets  : %llu\n", ub.ipackets);
        printf("ibytes    : %llu\n", ub.ibytes);
        printf("ierrors   : %llu\n", ub.ierrors);
        printf("opackets  : %llu\n", ub.opackets);
        printf("obytes    : %llu\n", ub.obytes);
        printf("oerrors   : %llu\n", ub.oerrors);
        printf("collisions: %llu\n", ub.collisions);
    }
}

main()
{
    cpu_total();
    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_tttime[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_tttime;
   static u_longlong_trunque;
   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("%3lld ", ub.runque-runque);
            printf("%3lld ", ub.swpque-swpque);
            printf("%6lld ", ub.sysfork-sysfork);
            printf("%6lld ", (ub.devintrs+ub.softintrs)-(devintrs+softintrs));
            printf("%6lld ", ub.syscall-syscall);
            printf("%6lld ", ub.pswitch-pswitch);
            printf("%3lld ", (ub.user-user)*100/ttime);
            printf("%3lld ", (ub.sys-sys)*100/ttime);
            printf("%3lld ", (ub.idle-idle)*100/ttime);
            printf("%3lld ", (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
%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_tierrors[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_tub;
   static u_longlong_tipackets;
   static u_longlong_tibytes;
   static u_longlong_tierrors;
   static u_longlong_topackets;
   static u_longlong_tobytes;
   static u_longlong_toerrors;
   static u_longlong_tcollisions;

   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                        Cpu
 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    pso     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 %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));
        /*
         * Finaly we get the next statistic in our data hierarchy.
         * And if this is NULL, then we have retreived 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   *statval1;
    float                 spmivalue;

/* 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 */
       statval1 = SpmiFirstVals(statset);
       if (statval1 == NULL) {
          printf("SpmiFirstVals Failed\n");
          if (SpmiErrno)
             printf("%s", SpmiErrmsg);
          must_exit();
       }

       /* traverse the set of statistics */
       while (statval1 != NULL) {
          /* value to be displayed */
          spmivalue = SpmiGetValue(statset, statval1);
          if (spmivalue < 0.0) {
             printf("SpmiGetValue Failed\n");
             if (SpmiErrno)
                printf("%s", SpmiErrmsg);
             must_exit();
          }
          printf("  %6.2f   ",spmivalue);

          statval1 = SpmiNextVals(statset, statval1);
       }  /* end while (statval1) */
       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()
{
    SpmiCxHdl   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 (!(parenthdl = 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 (!(cxhdl = SpmiPathGetCx("Real", parenthdl)))
    {
       printf("SpmiPathGetCx failed for Mem/Real context.\n");
       if (SpmiErrmsg)
          printf("%s", SpmiErrmsg);
       must_exit();
    }

    if (!SpmiPathAddSetStat(statset,"%free", cxhdl))
    {
       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 (!(cxhdl = SpmiPathGetCx("CPU/cpu0", NULL)))
    {
       printf("SpmiPathGetCx failed for CPU/cpu0 context.\n");
       if (SpmiErrno)
          printf("%s", SpmiErrmsg);
       must_exit();
    }

    if (!SpmiPathAddSetStat(statset,"idle", cxhdl))
```

```
    {
       printf("SpmiPathAddSetStat failed for CPU/cpu0/idle statistic.\n");
       if (SpmiErrno)
          printf("%s", SpmiErrmsg);
       must_exit();
    }

    if (!SpmiPathAddSetStat(statset,"kern", cxhdl))
    {
       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() registred 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);
   /*
    * Finaly we get the next statistic in our data hierarchy.
    * And if this is NULL, then we have retreived 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);
}
```

SPMI statistics in AIX 5.3

The following example is a execution result of above program. This list contains every statistic that is supported by SPMI API. You can refer to this list and decide which of those statistics will be monitored.

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 procsinfops[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();
}
```

B

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 3.7, “The trace, trcnm, and trcrpt commands” on page 147 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. We recommend 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 (lookuppn)
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 XMALLOC 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: cachefs 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)
```

Abbreviations and acronyms

AIO	Asynchronous I/O
AIX	Advanced Interactive Executive
API	Application Programming Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
CGI	Common Gateway Interface
CLI	Command Line Interface
CPU	Central Processing Unit
CSM	Cluster Systems Management
DASD	Direct Access Storage Device
DLPAR	Dynamic Logical Partition
DLPAR	Dynamic Logical Partition
DMA	Direct Memory Access
DNS	Domain Name Service
DPSA	Deferred Paging Space Algorithm
EPSA	Early Paging Space Algorithm
FC	Fibre Channel
FDDI	Fiber Distributed Data Interface
FIFO	First In First Out
GPFS	General Parallel File System
GUI	Graphical User Interface
HACMP/ES	High Availability Cluster Multi-Processing/Enhanced Scalability
HAEM	High Availability Event Management
HMC	Hardware Management Console
IBM	International Business Machines Corporation
IPC	Inter Process Communication
IPL	Initial Program Load
ISNO	Interface Specific Network Options
ITSO	International Technical Support Organization
JFS	Journaled File System
KEX	Kernel Extension
LAN	Local Area Network
LMBs	Logical Memory Blocks
LPAR	Logical Partition
LPP	Licensed Product Program
LPSA	Late Paging Space Algorithm
LUN	Logical Unit Number
LV	Logical Volume
LVD	Low Voltage Differential (Scsi)
LVDD	Logical Volume Device Driver
LVM	Logical Volume Manager
MCM	Multi-chip Module
MPIO	Multi-path I/O
MTU	Maximum Transfer Unit
NAS	Network Access Storage
NFS	Network File System

NFS	Network File System
NIM	Network Install Manager
NIS	Network Information Services
NLS	National Language Support
ODM	Object Data Manager
PCM	Path Control Module
PID	Process Id
PLM	Partition Load Manager
PMAPI	Performance Monitoring Application Programming Interface
PSSP	Parallel Systems Support Program
PTX	Performance Toolbox
RFC	Request For Comment
RMC	Resource Monitoring Control
RPC	Remote Procedure Call
RPM	Red Hat Package Manager
RSCT	Reliable Scalable Cluster Technology
SCSI	Small Computer System Interface
SDD	Subsystem Device Driver
SLIP	Serial Line Interface Protocol
SMIT	System Management Interface Tool
SMP	Symmetric Multi-processing
SMT	Simultaneous Multi-threading
SPLPAR	Shared Processor Logical Partition
SPMI	System Performance Monitoring Interface
SRC	System Resource Controller
VMM	Virtual Memory Manager
VP	Virtual Processor
WLM	Workload Manager
WLM	Workload Manager
WWN	World Wide Number

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 713. Note that some of the documents referenced here may be available in softcopy only.

- *Developing and Porting C and C++ Applications on AIX*, SG24-5674
- *Advanced POWER Virtualization on IBM @server p5 Servers Architecture and Performance Considerations*, SG24-5768
- *Advanced POWER Virtualization on IBM @server p5 Servers: Introduction and Basic Configuration*, SG24-7940
- *Auditing and Accounting on AIX*, SG24-6020
- *Accounting and Auditing on AIX 5L*, SG24-6396
- *Introduction to pSeries Provisioning*, SG24-6389
- *AIX Logical Volume Manager from A to Z, Introduction and Concepts*, SG24-5432
- *A Practical Guide for Resource Monitoring and Control (RMC)*, SG24-6615
- *A Comparison of Workload Management and Partitioning*, TIPS0426
- *AIX 5L Workload Manager (WLM)*, SG24-5977
- *Understanding IBM @server pSeries Performance and Sizing*, SG24-4810
- *RS/6000 SP System Performance Tuning Update,* SG24-5340
- *AIX 5L Performance Tools Handbook*, SG24-6039
- *AIX 5L Differences Guide Version 5.3 Edition*, SG24-5765
- *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155

Other publications

These publications are also relevant as further information sources:

- *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 1*, SC23-4913
- *AIX 5L Version 5.3 Technical Reference: Base Operating System and Extensions, Volume 2*, SC23-4914
- *IBM Reliable Scalable Cluster Technology Administration Guide*, SA22-7889
- *IBM Reliable Scalable Cluster Technology for AIX 5L Technical Reference*, SA22-7890
- *AIX 5L Version 5.3 Performance Tools Guide and Reference*, SC23-4906
- *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905
- *Performance Toolbox Version 2 and 3 Guide and Reference*, SC23-2625
- *AIX 5L Version 5.3 System Management Concepts: Operating System and Devices*, SC23-4908
- *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*, SC23-4910
- *AIX 5L Version 5.3 Commands Reference, Volume 1*, SC23-4888
- *AIX 5L Version 5.3 Commands Reference, Volume 2*, SC23-4889
- *AIX 5L Version 5.3 Commands Reference, Volume 3*, SC23-4890
- *AIX 5L Version 5.3 Commands Reference, Volume 4*, SC23-4891
- *AIX 5L Version 5.3 Commands Reference, Volume 5*, SC23-4892
- *AIX 5L Version 5.3 Commands Reference, Volume 6*, SC23-4893
- *AIX 5L Version 5.3 System User's Guide: Operating System and Devices*, SC23-4911
- *AIX 5L Version 5.3 General Programming Concepts*, SC23-4896
- *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *Event Management Programming Guide and Reference*, SA22-7354
- *AIX 5L Version 5.3 Files Reference*, SC23-4895
- *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts*, SC23-4900

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM High Performance Computing Toolkit Web page

  http://www.alphaworks.ibm.com/tech/hpmtoolkit

- RS/6000 SP System Performance Tuning Update

  http://www.rs6000.ibm.com/support/sp/perf

- Request For Comment (RFC) Web site

  http://www.rfc-editor.org/

- IBM network management products

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

- Performance problem determination tool collection

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

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

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

AIX 5L Practical Performance Tools and Tuning Guide

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

AIX 5L Practical Performance Tools and Tuning Guide

Updated performance information for IBM @server p5 and AIX 5L V5.3

New tools for @server p5 with SMT and Micro-Partitioning

Practical performance problem determination examples

This IBM Redbook incorporates the latest AIX 5L performance and tuning tools. It is a comprehensive guide about the performance monitoring and tuning tools that are provided with AIX 5L Version 5.3, and it is the ultimate guide for system administrators and support professionals who want to efficiently use the AIX performance monitoring and tuning tools and understand how to interpret the statistics.

The usage of each tool is explained along with the measurements it takes and the statistics it produces. This redbook contains a large number of usage and output examples for each of the tools, pointing out the relevant statistics to look for when analyzing an AIX system's performance from a practical point of view. It also explains the performance API available with AIX 5L and gives examples about how to create your own performance tools.

This redbook also contains an overview of the graphical AIX performance tools available with AIX 5L and the AIX Performance Toolbox Version 3.0.

This redbook is a rework of the very popular redbook *AIX 5L Performance Tools Handbook*, SG24-6039, published in 2003.

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

SG24-6478-00 ISBN 0738491799