IBM

IBM AIX Continuous Availability Features

Learn about AIX V6.1 and POWER6 advanced availability features

View sample programs that exploit storage protection keys

Harden your AIX system

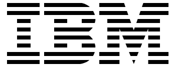
Octavian Lascu
Shawn Bodily
Matti Harvala
Anil K Singh
DoYoung Song
Frans Van Den Berg

Redpaper

ibm.com/redbooks

Archived

Archived



International Technical Support Organization

IBM AIX Continuous Availability Features

April 2008

Archived

REDP-4367-00

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

Archived

First Edition (April 2008)

This edition applies to Version 6, Release 1, of AIX Operating System (product number 5765-G62).

© Copyright International Business Machines Corporation 2008. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices vii
Trademarks viii

Preface ix
The team that wrote this paper ix
Become a published author xi
Comments welcome xii

Chapter 1. Introduction 1
1.1 Overview 2
1.2 Business continuity 2
1.3 Disaster recovery 3
1.4 High availability 3
1.5 Continuous operations 4
1.6 Continuous availability 4
1.6.1 Reliability 5
1.6.2 Availability 6
1.6.3 Serviceability 6
1.7 First Failure Data Capture 7
1.8 IBM AIX continuous availability strategies 8

Chapter 2. AIX continuous availability features 11
2.1 System availability 12
2.1.1 Dynamic Logical Partitioning 12
2.1.2 CPU Guard 12
2.1.3 CPU Sparing 12
2.1.4 Predictive CPU deallocation and dynamic processor deallocation 13
2.1.5 Processor recovery and alternate processor 13
2.1.6 Excessive interrupt disablement detection 13
2.1.7 Memory page deallocation 14
2.1.8 System Resource Controller 14
2.1.9 PCI hot plug management 15
2.1.10 Reliable Scalable Cluster Technology 15
2.1.11 Dual IBM Virtual I/O Server 17
2.1.12 Special Uncorrectable Error handling 18
2.1.13 Automated system hang recovery 19
2.1.14 Recovery framework 19
2.2 System reliability 19
2.2.1 Error checking 20
2.2.2 Extended Error Handling 20
2.2.3 Paging space verification 21
2.2.4 Storage keys 21
2.3 System serviceability 23
2.3.1 Advanced First Failure Data Capture features 23
2.3.2 Traditional system dump 23
2.3.3 Firmware-assisted system dump 24
2.3.4 Live dump and component dump 25
2.3.5 The dumpctrl command 26
2.3.6 Parallel dump 27

Archived

© Copyright IBM Corp. 2008. All rights reserved.

2.3.7 Minidump 27
2.3.8 Trace (system trace) 27
2.3.9 Component Trace facility 29
2.3.10 Lightweight Memory Trace (LMT) 30
2.3.11 ProbeVue 30
2.3.12 Error logging 30
2.3.13 The alog facility 31
2.3.14 syslog 32
2.3.15 Concurrent AIX Update 34
2.3.16 Core file control 35
2.4 Network tools 36
2.4.1 Virtual IP address support (VIPA) 37
2.4.2 Multipath IP routing 37
2.4.3 Dead gateway detection 37
2.4.4 EtherChannel 38
2.4.5 IEEE 802.3ad Link Aggregation 39
2.4.6 2-Port Adapter-based Ethernet failover. 40
2.4.7 Shared Ethernet failover 40
2.5 Storage tools 40
2.5.1 Hot swap disks 40
2.5.2 System backup (mksysb) 40
2.5.3 Alternate disk installation 41
2.5.4 The multibos utility 41
2.5.5 Network Installation Manager (NIM) 42
2.5.6 Logical Volume Manager-related options 42
2.5.7 Geographic Logical Volume Manager. 44
2.5.8 Journaled File System-related options 46
2.5.9 AIX storage device driver-related options 46
2.6 System and performance monitoring and tuning 48
2.6.1 Electronic Service Agent. 48
2.6.2 Other tools for monitoring a system 49
2.6.3 The topas command 49
2.6.4 Dynamic kernel tuning 50
2.7 Security 51
2.8 AIX mobility features 52
2.8.1 Live partition mobility 52
2.8.2 Live application mobility 52

Chapter 3. AIX advanced continuous availability tools and features. 55
3.1 AIX Reliability, Availability, and Serviceability component hierarchy 56
3.1.1 First Failure Data Capture feature. 56
3.2 Lightweight memory trace 57
3.2.1 LMT implementation 57
3.3 The snap command 61
3.4 Minidump facility 61
3.4.1 Minidump formatter 62
3.5 Live dump and component dump 64
3.5.1 Dump-aware components. 64
3.5.2 Performing a live dump 68
3.6 Concurrent AIX Update 71
3.6.1 Concurrent AIX Update terminology 71
3.6.2 Concurrent AIX Update commands and SMIT menus 72

Archived

3.7 Storage protection keys . . . 77
3.7.1 Storage protection keys overview . . . 77
3.7.2 System management support for storage keys. . . . 79
3.7.3 Kernel mode protection keys . . . 80
3.7.4 Degrees of storage key protection and porting considerations . . . 84
3.7.5 Protection gates . . . 87
3.7.6 Example using kernel keys . . . 90
3.7.7 User mode protection keys . . . 100
3.7.8 Kernel debugger commands . . . 108
3.7.9 Storage keys performance impact . . . 111
3.8 ProbeVue . . . 111
3.8.1 ProbeVue terminology . . . 113
3.8.2 Vue programming language . . . 113
3.8.3 The probevue command . . . 114
3.8.4 The probevctrl command . . . 114
3.8.5 Vue overview . . . 114
3.8.6 ProbeVue dynamic tracing example . . . 119
3.8.7 Other considerations for ProbeVue. . . . 125
3.9 Xmalloc debug enhancements in AIX V6.1 . . . 125
3.9.1 New features in xmalloc debug . . . 126
3.9.2 Enabling/disabling xmalloc RTEC and displaying current value . . . 126
3.9.3 Run-time error checking (RTEC) levels for XMDBG (alloc.xmdbg) . . . 127
3.9.4 XMDBG tunables affected by error check level . . . 131
3.9.5 XMDBG tunables not affected by error check level . . . 134
3.9.6 KDB commands for XMDBG. . . . 136
3.9.7 Heap registration for individual debug control . . . 137

Appendix A. AIX features availability . . . 139

Abbreviations and acronyms . . . 143

Related publications . . . 145
IBM Redbooks publications . . . 145
Other publications . . . 145
Online resources . . . 146
How to get IBM Redbooks publications . . . 146
Help from IBM . . . 146

Index . . . 147

Archived

Archived

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

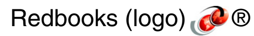
This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing 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.

Archived

© Copyright IBM Corp. 2008. All rights reserved.

Trademarks

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

Redbooks (logo)  ®
eServer™
pSeries®
z/OS®
AIX 5L™
AIX®
Blue Gene®
DB2®
Electronic Service Agent™
General Parallel File System™
GPFS™
HACMP™
IBM®
Power Architecture®
PowerPC®
POWER™
POWER Hypervisor™
POWER3™
POWER4™
POWER5™
POWER6™
Redbooks®
RS/6000®
System i™
System p™
System p5™
System z™
Workload Partitions Manager™

The following terms are trademarks of other companies:

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, or service names may be trademarks or service marks of others.

Archived

Preface

This IBM® Redpaper describes the continuous availability features of IBM AIX® Version 6, Release 1. It also addresses and defines the terms Reliability, Availability, and Serviceability (RAS) as used in an IT infrastructure. It touches on the global availability picture for an IT environment in order to better clarify and explain how AIX can improve that availability. The paper is intended for AIX specialists, whether customers, business partners, or IBM personnel, who are responsible for server availability.

A key goal of AIX development is to improve overall system serviceability by developing problem determination tools and techniques that have minimal impact on a live system. This document explains the new debugging tools and techniques, as well as the kernel facilities that work in conjunction with new hardware, that can help you provide continuous availability for your AIX systems.

The paper provides a broad description of the advanced continuous availability tools and features on AIX that help to capture software problems at the moment they appear with no need to recreate the failure. In addition to software problems, the AIX kernel works closely with advanced hardware features to identify and isolate failing hardware and replace hardware components dynamically without bringing down the system.

Among the tools discussed in this Redpaper are Dynamic Trace, Lightweight Memory Trace, Component Trace, Live dump and Component dump, Storage protection keys (kernel and user), Live Kernel update, and xmalloc debug.

The team that wrote this paper

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

Octavian Lascu is a Project Leader associated with the ITSO, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of IBM System p™ and Linux® clusters. His areas of expertise include High Performance Computing, Blue Gene® and Clusters. Before joining the ITSO, Octavian worked in 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 for IBM since 1992.

Shawn Bodily is a Certified Consulting IT Specialist with Advanced Technical Support (ATS) Americas based in Dallas, TX, and has worked for IBM for nine years. With 12 years of experience in AIX, his area of expertise for the last 10 years has been HACMP. Shawn has written and presented extensively on high availability for System p. He has co-authored two IBM Redbooks® publications.

Matti Harvala is an Advisory IT Specialist in Finland, and has worked for IBM for more than eight years. Matti has 10 years of experience supporting UNIX® systems, and holds a Bachelor of Science degree in Information Technology Engineering. His areas of expertise include AIX systems support, AIX Advanced Power Virtualization, NIM and DS 4000 family disk subsystems.

Anil K Singh is a Senior Software Engineer in India. He has six years of experience in testing and development, including more than three years in AIX. He holds a Bachelor of

© Copyright IBM Corp. 2008. All rights reserved.

Engineering degree in Computer Science. Anil's areas of expertise include programming and testing in kernel and user mode for TCP/IP, malloc subsystem, WPAR and WLM.

DoYoung Song is a Consulting IT specialist for pre-sales technical support in the IBM Technical Sales Support group in Seoul, and has worked for IBM for 17 years. He currently works as an IT Systems Architect. DoYoung has 15 years of experience in AIX, RS/6000®, and IBM System p. His areas of expertise include technologies and solutions on AIX and pSeries®, and System p high-end server systems, supporting IBM sales, IBM Business Partners, and clients with IT infrastructure and pre-sales consulting.

Frans Van Den Berg is an IT Specialist in the United Kingdom, and has worked at IBM for more than seven years. Frans has 11 years of experience working with UNIX. His areas of expertise include knowledge of a number of key operating systems, backup and recovery, security, database and software packages, SAN, storage and hardware skills.

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

Michael Lyons
IBM Austin

Susan Schreitmueller
IBM Dallas

Jay Kruemcke
IBM Austin

Michael S Wilcox
IBM Tulsa

Maria R Ward
IBM Austin

Saurabh Sharma
IBM Austin

Daniel Henderson, Brian Warner, Jim Mitchell
IBM Austin

Thierry Fauck
IBM France

Bernhard Buehler
IBM Germany

Donald Stence
IBM Austin

James Moody
IBM Austin

Grégoire Pichon
BULL AIX development, France

Jim Shaffer
IBM Austin, TX

Shajith Chandran
IBM India

Archived

Suresh Warrier
IBM Austin TX

Bruce Mealey
IBM Austin TX

Larry Brenner
IBM Austin TX

Mark Rogers
IBM Austin TX

Bruno Blanchard
IBM France

Steve Edwards
IBM UK

Brad Gough
IBM Australia

Hans Mozes
IBM Germany

The IBM Redbooks publication team for *IBM AIX V6.1 Differences Guide*
Rosa Fernandez
IBM France

Roman Aleksic
Zürcher Kantonalbank, Switzerland

Ismael Castillo
IBM Austin, TX

Armin Röll
IBM Germany

Nobuhiko Watanabe
IBM Japan Systems Engineering

Scott Vetter
International Technical Support Organization, Austin Center

Become a published author

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

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

Archived

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 papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Archived

1

Introduction

This chapter addresses some of the common concepts and defines the associated terms used in IT infrastructure and generally referred to as Reliability, Availability, and Serviceability (RAS). Although this paper covers the AIX V6, R1 operating system continuous availability features, it also introduces the global availability picture for an IT environment in order to better identify and understand what AIX brings to overall IT environment availability.

The chapter contains the following topics:

- Overview
- Business continuity
- Disaster recovery
- High availability
- Continuous operations
- Continuous availability
- First Failure Data Capture
- IBM AIX continuous availability strategies

© Copyright IBM Corp. 2008. All rights reserved.

1.1 Overview

In May 2007, IBM introduced the newest Power Architecture® technology-based line of servers incorporating the inventive IBM POWER6™ processor technology to deliver both outstanding performance and enhanced reliability, availability, and serviceability capabilities. This new line of servers enhances the IBM POWER5™ processor-based server family, with new capabilities designed to help ease administrative burdens and increase system utilization.

In addition, IBM virtualization technologies[1], available in the IBM System p and System i product families, enable individual servers to run dozens or even hundreds of mission-critical applications.

Today's enterprises can no longer afford planned or unplanned system outages. Even a few minutes of application downtime can result in financial losses, eroded customer confidence, damage to brand image, and public relations problems.

To better control and manage their IT infrastructure, enterprises have concentrated their IT operations in large (and on demand) data centers. These data centers must be resilient enough to handle the ups and downs of the global market, and must manage changes and threats with consistent availability, security and privacy, both around the clock and around the world. Most of the solutions are based on an integration of operating system clustering software, storage, and networking.

How a system, server or environment handles failures is characterized as its reliability, availability and serviceability. In today's world of e-business, the reliability, availability and serviceability of an operating system and the hardware on which it executes have assumed great importance.

Today's businesses require that IT systems be self-detecting, self-healing, and support 7x24x 365 operations. More and more IT systems are adopting fault tolerance through techniques such as redundancy and error correction, to achieve a high level of reliability, availability, and serviceability.

The reliability, availability and serviceability characteristics will be a significant market differentiator in the UNIX server space. This has resulted in UNIX servers attaining the reliability, availability and serviceability levels that were once considered to be available only on the mainframe systems.

More and more IT systems are adopting fault tolerance through redundancy, memory failure detection and correction methods, to achieve a high level of reliability, availability and serviceability.

The following sections discuss the concepts of continuous availability features in more detail.

1.2 Business continuity

The terms business continuity and disaster recovery are sometimes used interchangeably (as are business resumption and contingency planning). The following sections explain the definitions used in this paper.

[1] Virtualization features available for IBM System p and System i™ servers may depend on the type of hardware used, and may be subject to separate licensing.

Archived

Here, *business continuity* is defined as the ability to adapt and respond to risks, as well as opportunities, in order to maintain continuous business operations. However, business continuity solutions applied in one industry might not be applicable to a different industry, because they may have different sets of business continuity requirements and strategies.

Business continuity is implemented using a plan that follows a strategy that is defined according to the needs of the business. A total Business Continuity Plan has a much larger focus and includes items such as a crisis management plan, business impact analysis, human resources management, business recovery plan procedure, documentation and so on.

1.3 Disaster recovery

Here, *disaster recovery* is defined as the ability to recover a data center at a different site if a disaster destroys the primary site or otherwise renders it inoperable. The characteristics of a disaster recovery solution are that IT processing resumes at an alternate site, and on completely separate hardware.

Clarification:

- Disaster recovery is only one component of an overall business continuity plan.
- Business continuity planning forms the first level of planning *before* disaster recovery comes in to the plan.

Disaster recovery (DR) is a coordinated activity to enable the recovery of IT and business systems in the event of disaster. A DR plan covers both the hardware and software required to run critical business applications and the associated processes, and to (functionally) recover a complete site. The DR for IT operations employs additional equipment (in a physically different location) and the use of automatic or manual actions and methods to recover some or all of the affected business processes.

1.4 High availability

High availability is the attribute of a system which provides service during defined periods, at acceptable or agreed-upon levels, and masks *unplanned* outages from end users. It often consists of redundant hardware components, automated failure detection, recovery, bypass reconfiguration, testing, problem determination and change management procedures.

In addition, high availability is also the ability (and associated processes) to provide access to applications regardless of hardware, software, or system management issues. This is achieved through greatly reducing, or masking, *planned* downtime. Planned downtime often includes hardware upgrades, repairs, software updates, backups, testing, and development.

High availability solutions should eliminate single points of failure (SPOFs) through appropriate design, planning, selection of hardware, configuration of software, and carefully controlled change management discipline. High availability is fault resilience, but *not* fault tolerance.

Archived

1.5 Continuous operations

Continuous operations is an attribute of IT environments and systems which allows them to continuously operate and mask planned outages from end users. Continuous operations employs non-disruptive hardware, software, configuration and administrative changes.

Unplanned downtime is an unexpected outage and often is the result of administrator error, application software failure, operating system faults, hardware faults, or environmental disasters.

Generally, hardware component failure represents an extremely small proportion of overall system downtime. By far, the largest single contributor to system downtime is planned downtime. For example, shutting down a computer for the weekend is considered planned downtime. Stopping an application to take a level 0 (full) system backup is also considered planned downtime.

1.6 Continuous availability

Continuous availability is an attribute of a system which allows it to deliver non-disruptive service to end users 7 days a week, 24 hours a day by preventing both planned and unplanned outages.

> *Continuous availability (Elimination of downtime) = Continuous operations (Masking or elimination of planned downtime) + High availability (Masking or elimination of unplanned downtime)*

Most of today's solutions are based on an integration of the operating system with clustering software, storage, and networking. When a failure is detected, the integrated solution will trigger an event that will perform a predefined set of tasks required to reactivate the operating system, storage, network, and in many cases, the application on another set of servers and storage. This kind of functionality is defined as *IT continuous availability*.

The main goal in protecting an IT environment is to achieve continuous availability; that is, having no end-user observed downtime. Continuous availability is a collective term for those characteristics of a product which make it:

- Capable of performing its intended functions under stated conditions for a stated period of time (reliability)
- Ready to perform its function whenever requested (availability)
- Able to quickly determine the cause of an error and to provide a solution to eliminate the effects of the error (serviceability)

Continuous availability encompasses techniques for reducing the number of faults, minimizing the effects of faults when they occur, reducing the time for repair, and enabling the customer to resolve problems as quickly and seamlessly as possible.

AIX continuous availability encompasses all tools and techniques implemented at the operating system level that contribute to overall system availability. Figure 1-1 on page 5 illustrates AIX continuous availability.

Archived

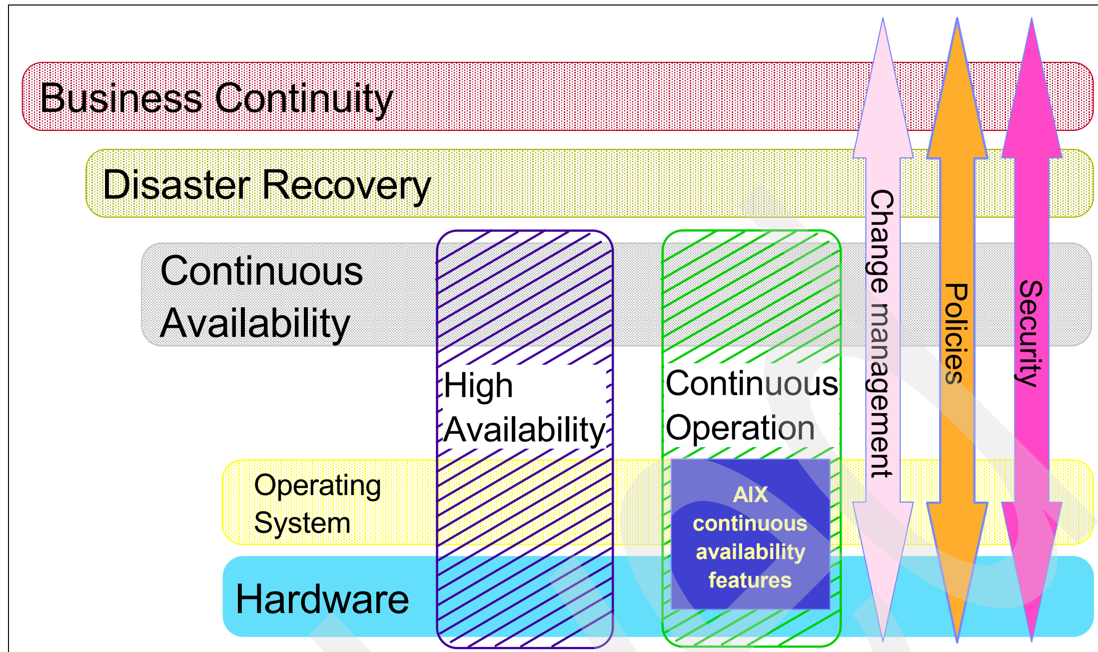


Figure 1-1 Positioning AIX continuous availability

1.6.1 Reliability

From a server hardware perspective, reliability is a collection of technologies (such as chipkill memory error detection/correction, dynamic configuration and so on) that enhance system reliability by identifying specific hardware errors and isolating the failing components.

Built-in system failure recovery methods enable cluster nodes to recover, without falling over to a backup node, when problems have been detected by a component within a node in the cluster. Built-in system failure recovery should be transparent and achieved without the loss or corruption of data. It should also be much faster compared to system or application failover recovery (failover to a backup server and recover). And because the workload does not shift from this node to another, no other node's performance or operation should be affected. Built-in system recovery should cover applications (monitoring and restart), disks, disk adapters, LAN adapters, power supplies (battery backups) and fans.

From a software perspective, reliability is the capability of a program to perform its intended functions under specified conditions for a defined period of time. Software reliability is achieved mainly in two ways: infrequent failures (built-in software reliability), and extensive recovery capabilities (self healing - availability).

IBM's fundamental focus on software quality is the primary driver of improvements in reducing the rate of software failures. As for recovery features, IBM-developed operating systems have historically mandated recovery processing in both the mainline program and in separate recovery routines as part of basic program design.

As IBM System p systems become larger, more and more customers expect mainframe levels of reliability. For some customers, this expectation derives from their prior experience with mainframe systems which were “downsized” to UNIX servers. For others, this is simply a consequence of having systems that support more users.

The cost associated with an outage grows every year, therefore avoiding outages becomes increasingly important. This leads to new design requirements for all AIX-related software.

For all operating system or application errors, recovery must be attempted. When an error occurs, it is not valid to simply give up and terminate processing. Instead, the operating system or application must at least try to keep the component affected by the error up and running. If that is not possible, the operating system or application should make every effort to capture the error data and automate system restart as quickly as possible.

The amount of effort put into the recovery should, of course, be proportional to the impact of a failure and the reasonableness of “trying again”. If actual recovery is not feasible, then the impact of the error should be reduced to the minimum appropriate level.

Today, many customers require that recovery processing be subject to a time limit and have concluded that rapid termination with quick restart or takeover by another application or system is preferable to delayed success. However, takeover strategies rely on redundancy that becomes more and more expensive as systems get larger, and in most cases the main reason for quick termination is to begin a lengthy takeover process as soon as possible. Thus, the focus is now shifting back towards core reliability, and that means quality and recovery features.

1.6.2 Availability

Today’s systems have hot plug capabilities for many subcomponents, from processors to input/output cards to memory. Also, clustering techniques, reconfigurable input/output data paths, mirrored disks, and hot swappable hardware should help to achieve a significant level of system availability.

From a software perspective, availability is the capability of a program to perform its function whenever it is needed. Availability is a basic customer requirement. Customers require a stable degree of certainty, and also require that schedules and user needs are met.

Availability gauges the percentage of time a system or program can be used by the customer for productive use. Availability is determined by the number of interruptions and the duration of the interruptions, and depends on characteristics and capabilities which include:

- The ability to change program or operating system parameters without rebuilding the kernel and restarting the system
- The ability to configure new devices without restarting the system
- The ability to install new software or update existing software without restarting the system
- The ability to monitor system resources and programs and cleanup or recover resources when failures occur
- The ability to maintain data integrity in spite of errors

The AIX operating system includes many availability characteristics and capabilities from which your overall environment will benefit.

1.6.3 Serviceability

Focus on serviceability is shifting from providing customer support remotely through conventional methods, such as phone and e-mail, to automated system problem reporting and correction, without user (or system administrator) intervention.

Hot swapping capabilities of some hardware components enhances the serviceability aspect. A service processor with advanced diagnostic and administrative tools further enhances the system serviceability. A System p server's service processor can call home in the service report, providing detailed information for IBM service to act upon. This automation not only

Archived

eases the burden placed on system administrators and IT support staff, but also enables rapid and precise collection of problem data.

On the software side, serviceability is the ability to diagnose and correct or recover from an error when it occurs. The most significant serviceability capabilities and enablers in AIX are referred to as the software service aids. The primary software service aids are error logging, system dump, and tracing.

With the advent of next generation UNIX servers from IBM, many hardware reliability-, availability-, and serviceability-related issues such as memory error detection, LPARs, hardware sensors and so on have been implemented. These features are supported with the relevant software in AIX. These abilities continue to establish AIX as the best UNIX operating system.

1.7 First Failure Data Capture

IBM has implemented a server design that builds in thousands of hardware error checker stations that capture and help to identify error conditions within the server. The IBM System p p5-595 server, for example, includes almost 80,000 checkers to help capture and identify error conditions. These are stored in over 29,000 Fault Isolation Register bits. Each of these checkers is viewed as a “diagnostic probe” into the server, and, when coupled with extensive diagnostic firmware routines, allows quick and accurate assessment of hardware error conditions at run-time.

Integrated hardware error detection and fault isolation is a key component of the System p and System i platform design strategy. It is for this reason that in 1997, IBM introduced First Failure Data Capture (FFDC) for IBM POWER™ servers. FFDC plays a critical role in delivering servers that can self-diagnose and self-heal. The system effectively traps hardware errors at system run time.

FFDC is a technique which ensures that when a fault is detected in a system (through error checkers or other types of detection methods), the root cause of the fault will be captured without the need to recreate the problem or run any sort of extended tracing or diagnostics program. For the vast majority of faults, an effective FFDC design means that the root cause can also be detected automatically without servicer intervention. The pertinent error data related to the fault is captured and saved for further analysis.

In hardware, FFDC data is collected in fault isolation registers based on the first event that had occurred. FFDC check stations are carefully positioned within the server logic and data paths to ensure that potential errors can be quickly identified and accurately tracked to an individual Field Replaceable Unit (FRU).

This proactive diagnostic strategy is a significant improvement over less accurate “reboot and diagnose” service approaches. Using projections based on IBM internal tracking information, it is possible to predict that high impact outages would occur two to three times more frequently without an FFDC capability.

In fact, without some type of pervasive method for problem diagnosis, even simple problems that occur intermittently can cause serious and prolonged outages. By using this proactive diagnostic approach, IBM no longer has to rely on an intermittent “reboot and retry” error detection strategy, but instead knows with some certainty which part is having problems.

This architecture is also the basis for IBM predictive failure analysis, because the Service Processor can now count and log intermittent component errors and can deallocate or take other corrective actions when an error threshold is reached.

IBM has tried to enhance FFDC features such that in most cases, failures in AIX will *not* result in recreate requests, also known as Second Failure Data Capture (SFDC), from AIX support to customers in order to solve the problem. In AIX, this service functionality focuses on gathering sufficient information upon a failure to allow for complete diagnosis without requiring failure reproduction. For example, Lightweight Memory Trace (LMT) support introduced with AIX V5.3 ML3 represents a significant advance in AIX first failure data capture capabilities, and provides service personnel with a powerful and valuable tool for diagnosing problems.

The Run-Time Error Checking (RTEC) facility provides service personnel with a method to manipulate debug capabilities that are already built into product binaries. RTEC provides service personnel with powerful first failure data capture and second failure data capture (SFDC) error detection features. This SFDC service functionality focuses on tools to enhance serviceability data gathering after an initial failure. The basic RTEC framework has been introduced in AIX V5.3 TL3, and extended with additional features in subsequent AIX releases.

1.8 IBM AIX continuous availability strategies

There are many market requirements for continuous availability to resolve typical customer pain points, including:

- Too many scheduled outages
- Service depends on problem recreation and intrusive problem determination
- System unavailability disrupts customer business
- Need for reliable protection of customer data

IBM has made AIX robust with respect to continuous availability characteristics, and this robustness makes IBM UNIX servers the best in the market. IBM's AIX continuous availability strategy has the following characteristics:

- Reduce the frequency and severity of AIX system outages, planned and unplanned
- Improve serviceability by enhancing AIX failure data capture tools.
- Provide enhancements to debug and problem analysis tools.
- Ensure that all necessary information involving unplanned outages is provided, to correct the problem with minimal customer effort
- Use of mainframe hardware features for operating system continuous availability brought to System p hardware
- Provide key error detection capabilities through hardware-assist
- Exploit other System p hardware aspects to continue transition to “stay-up” designs
- Use of “stay-up” designs for continuous availability
- Maintain operating system availability in the face of errors while minimizing application impacts
- Use of sophisticated and granular operating system error detection and recovery capabilities
- Maintain a strong tie between serviceability and availability
- Provide problem diagnosis from data captured at first failure without the need for further disruption
- Provide service aids that are non-disruptive to the customer environment

- Provide end-to-end and integrated continuous availability capabilities across the server environment and beyond the base operating system
- Provide operating system enablement and application and storage exploitation of the continuous availability environment

This paper explores and explains continuous availability features and enhancements available in AIX V5.3, as well as the new features in AIX V6.1.

The goal is to provide a summarized, exact definition of all the enhancements (including those not visible directly by users, as well as the visible ones), complete with working scenarios of commands and the background information required to understand a topic.

The paper is intended for AIX specialists, whether customers, business partners, or IBM personnel, who are responsible for server availability.

Archived

Archived

2

AIX continuous availability features

This chapter explains the features and tools that exist in AIX to enhance the availability of the AIX operating system. It summarizes both new and existing AIX availability, reliability, and serviceability features and tools.

Today's IT industries can no longer afford system outages, whether planned or unplanned. Even a few minutes of application downtime can cause significant financial losses, erode client confidence, damage brand image, and create public relations problems.

The primary role of an operating system is to manage the physical resources of a computer system to optimize the performance of its applications. In addition, an operating system needs to handle changes in the amount of physical resources allocated to it in a smooth fashion and without any downtime. Endowing a computing system with this self-management feature often translates to the implementation of self-protecting, self-healing, self-optimizing, and self-configuring facilities and features.

Customers are looking for autonomic computing, in the ability of components and operating systems to adapt smoothly to changes in their environment. Some of the most prominent physical resources of an operating system are processors, physical memory, and I/O devices; how a system deals with the loss of any of these resources is an important feature in the making of a continuously available operating system. At the same time, the need to add and remove resources, as well as maintain systems with little or no impact to the application or database environment, and hence the business, are other important considerations.

© Copyright IBM Corp. 2008. All rights reserved.

2.1 System availability

AIX is built on architectural foundations that enable servers to continuously tune themselves, adapt to unexpected conditions, help prevent and recover from failures, and provide a safe environment for critical data and systems. Specific examples of these features of AIX include First Failure Data Capture (FFDC), automatic system recovery and I/O hang detection and recovery, self-optimizing disk management, dynamic partitioning for efficient resource utilization and the ability to automatically dial-up for service in anticipation of a system failure. AIX is designed to automate systems management and to maximize system availability.

2.1.1 Dynamic Logical Partitioning

The Dynamic Logical Partitioning (DLPAR) feature allows processor, memory, and I/O-slot resources to be added, deleted from, or moved between running partitions, without requiring any AIX instance to be rebooted. For more detailed information about DLPAR, refer to the IBM Redbooks publication, *Advanced POWER Virtualization on IBM System p5: Introduction and Configuration*, SG24-7940, which is located at the following site:

http://www.redbooks.ibm.com/redbooks/pdfs/sg247940.pdf

2.1.2 CPU Guard

For dynamic processor deallocation, the service processor performs a predictive failure analysis based on any recoverable processor errors that have been recorded. If these transient errors exceed a defined threshold, the event is logged and the processor is deallocated from the system while the operating system continues to run.

The original CPU Guard feature predicts the failure of a running CPU by monitoring certain types of transient errors and dynamically takes the CPU offline, but it does not provide a substitute CPU, so that a customer is left with less computing power. Additionally, the older feature will not allow an SMP system to operate with fewer than two processors.

The Dynamic CPU Guard feature, introduced in AIX 5.2, is an improved and dynamic version of the original CPU Guard that was available in earlier AIX versions. The key differences are that it utilizes DLPAR technologies and allows the operating system to function with only one processor. This feature, beginning with AIX 5.2, is enabled by default. Example 2-1 shows how to check this attribute.

Example 2-1 Dynamic CPU Guard

```
briley# lsattr -El sys0 |grep cpuguard
cpuguard        enable            CPU Guard                        True
```

If this feature is disabled, you can enable it by executing the **chdev** command as follows:

```
chdev -l sys0 -a cpuguard=enable
```

2.1.3 CPU Sparing

The Dynamic CPU Sparing feature allows the transparent substitution of a suspected defective CPU with a good unlicensed processor (part of a Capacity on Demand processor pool). This online switch is made seamlessly, so that applications and kernel extensions are not impacted.

Archived

The new processor autonomously replaces the defective one. Dynamic CPU Guard and Dynamic CPU Sparing work together to protect a customer's investments through their self-diagnosing and self-healing software. More information is available at the following site:

http://www.research.ibm.com/journal/sj/421/jann.html

2.1.4 Predictive CPU deallocation and dynamic processor deallocation

On these systems, AIX implements continuous hardware surveillance and regularly polls the firmware for hardware errors. When the number of processor errors hits a threshold and the firmware recognizes that there is a distinct probability that this system component will fail, then the firmware returns an error report. In all cases, the error is logged in the system error log. In addition, on multiprocessor systems, depending on the type of failure, AIX attempts to stop using the untrustworthy processor and deallocate it. More information is available at the following site:

http://www-05.ibm.com/cz/power6/files/zpravy/WhitePaper_POWER6_availabilty.PDF

2.1.5 Processor recovery and alternate processor

Although this is related mainly to System p hardware, AIX still plays a role in this process and coding in AIX allows the alternate processor recovery feature to deallocate and deconfigure a failing processor by moving the instruction stream over to and restarting it on a spare processor. These operations can be accomplished by the POWER Hypervisor™ and POWER6 hardware without application interruption, thus allowing processing to continue unimpeded. More information is available at the following site:

http://www-05.ibm.com/cz/power6/files/zpravy/WhitePaper_POWER6_availabilty.PDF

2.1.6 Excessive interrupt disablement detection

AIX V5.3 ML3 has introduced a new feature which can detect a period of excessive interrupt disablement on a CPU, and create an error log record to report it. This allows you to know if privileged code running on a system is unduly (and silently) impacting performance. It also helps to identify and improve such offending code paths before the problems manifest in ways that have proven very difficult to diagnose in the past.

This feature employs a kernel profiling approach to detect disabled code that runs for too long. The basic idea is to take advantage of the regularly scheduled clock “ticks” that generally occur every 10 milliseconds, using them to approximately measure continuously disabled stretches of CPU time individually on each logical processor in the configuration.

This approach will alert you to partially disabled code sequences by logging one or more hits within the offending code. It will alert you to fully disabled code sequences by logging the i_enable that terminates them.

You can turn excessive interrupt disablement off and on, respectively, by changing the proc.disa RAS component:

```
errctrl -c proc.disa errcheckoff
errctrl -c proc.disa errcheckon
```

Note that the preceding commands only affect the *current* boot. In AIX 6.1, the `-P` flag is introduced so that the setting can be changed persistently across reboots, for example:

```
errctrl -c proc.disa -P errcheckoff
```

Archived

On AIX 5.3, the only way to persistently disable component error checking (including excessive interrupt disablement detection) is to turn it off at the system level. On AIX 5.3 TL3 and TL4, this is done via:

```
errctrl errcheckoff -c all
```

On AIX 5.3 ML5 and later releases, it is done via:

```
errctrl -P errcheckoff
```

Additional detailed information about excessive interrupt disablement is available at the following site:

http://www-1.ibm.com/support/docview.wss?uid=isg3T1000678

2.1.7 Memory page deallocation

While a coincident single memory cell error in separate memory chips is a statistical rarity, POWER6 processor-based systems can contain these errors using a memory page deallocation scheme for partitions running AIX. If a memory fault is detected by the Service Processor at boot time, the affected memory will be marked as bad and will not be used on subsequent reboots. This is known as Memory Persistent Deallocation. If the service processor identifies faulty memory in a server that includes Capacity on Demand (CoD) memory, the POWER Hypervisor attempts to replace the faulty memory with available CoD memory.

In other cases, the POWER Hypervisor notifies the owning partition that the page should be deallocated. Where possible, the operating system moves any data currently contained in that memory area to another memory area and removes the pages associated with this error from its memory map, no longer addressing these pages. The operating system performs memory page deallocation without any user intervention and is transparent to end users and applications.

Additional detailed information about memory page deallocation is available at the following site:

http://www-05.ibm.com/cz/power6/files/zpravy/WhitePaper_POWER6_availabilty.PDF

2.1.8 System Resource Controller

The System Resource Controller (SRC) provides a set of commands and subroutines to make it easier for the system administrators to create and control subsystems. A *subsystem* is any program or process or set of programs or processes that is usually capable of operating independently or with a controlling system. A subsystem is designed as a unit to provide a designated function.

The SRC was designed to minimize the need for operator intervention. It provides a mechanism to control subsystem processes using a common command line and the C interface. This mechanism includes the following:

- Consistent user interface for start, stop, and status inquiries
- Logging of the abnormal termination of subsystems
- Notification program called at the abnormal system termination of related processes
- Tracing of a subsystem, a group of subsystems, or a subserver
- Support for control of operations on a remote system
- Refreshing of a subsystem (such as after a configuration data change)

Archived

The SRC is useful if you want a common way to start, stop, and collect status information about processes. You can use these options in SMIT via **smitty src.**

2.1.9 PCI hot plug management

PCI hot plug management consists of user (software) tools that allow you to manage hot plug connectors, also known as *dynamic reconfigurable connectors*, or *slots*. A connector defines the type of slot, for example, PCI. A slot has a unique identifier within the managed system.

Dynamic reconfiguration is the ability of the system to adapt to changes in the hardware and firmware configuration while it is still running. PCI Hot Plug Support for PCI Adapters is a specific subset of the dynamic reconfiguration function that provides the capability of adding, removing, and replacing PCI adapter cards while the host system is running and without interrupting other adapters in the system. You can also display information about PCI hot plug slots.

You can insert a new PCI hot plug adapter into an available PCI slot while the operating system is running. This can be another adapter of the same type that is currently installed, or of a different type of PCI adapter. New resources are made available to the operating system and applications without having to restart the operating system. The PCI hot plug manager interface can be accessed by executing the following command:

```
smitty devdrpci
```

Additional detailed information about PCI hot plug management is available in *AIX 5L System Management Guide: Operating System and Devices*, SC23-5204, which is downloadable from the following site:

http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.baseadmn/doc/baseadmndita/baseadmndita.pdf

2.1.10 Reliable Scalable Cluster Technology

Reliable Scalable Cluster Technology (RSCT) is a set of software components that together provide a comprehensive clustering environment for AIX and Linux. RSCT is the infrastructure used by a variety of IBM products to provide clusters with improved system availability, scalability, and ease of use. RSCT includes the following components:

- Resource Monitoring and Control (ctrmc) subsystem
- RSCT core resource managers (ctcas)
- RSCT cluster security services (ctsec)
- Topology Services subsystem (cthatsd)
- Group Services subsystem (cthagsd)

These components are explained in more detail here.

Resource Monitoring and Control subsystem

The Resource Monitoring and Control (RMC) subsystem is the scalable backbone of RSCT which provides a generalized framework for managing resources within a single system or a cluster. Its generalized framework is used by cluster management tools to monitor, query, modify, and control cluster resources.

RMC provides a single monitoring and management infrastructure for standalone servers (single operating system image), RSCT peer domains (where the infrastructure is used by the configuration resource manager), and management domains (where the infrastructure is used by the Hardware Management Console (HMC) and Cluster Systems Management (CSM)).

When used on a standalone server (single operating system), RMC enables you to monitor and manage the resources of that machine. However, when a group of machines that are each running RMC are clustered together (into management domains or peer domains), the RMC framework allows a process on any node to perform an operation on one or more resources on any other node in the domain.

A *resource* is the fundamental concept of the RMC architecture; it is an instance of a physical or logical entity that provides services to some other component of the system. Examples of resources include lv01 on node 10; Ethernet device en0 on node 14; IP address 9.117.7.21; and so on. A set of resources that have similar characteristics (in terms of services provided, configuration parameters, and so on) is called a *resource class*. The resources and resource class abstractions are defined by a resource manager.

RMC is part of standard AIX V5 and V6 installation, and provides comprehensive monitoring when configured and activated. The idea behind the RSCT/RMC implementation is to provide a high availability infrastructure for managing resources in a standalone system, as well as in a cluster (peer domain or management domain).

RMC can be configured to monitor any event that may occur on your system, and you can provide a response program (script or binary). For example, if a particular file system is always filling up, you can configure the RMC to raise an event when the file system grows to a specified utilization threshold. Your response program (script) might increase the size of the file system or archive old data, but after the user-specified response script is executed and the condition recovers (that is, file system utilization falls below a specified reset value), then the event is cleared and RMC returns to monitor mode.

A *resource manager* is a process that maps resource and resource class abstractions into actual calls and commands for one or more specific types of resources. A resource manager runs as a standalone daemon and contains definitions of all resource classes that the resource manager supports. These definitions include a description of all attributes, actions, and other characteristics of a resource class. RSCT provides a core set of resource managers for managing base resources on single systems and across clusters.

RSCT core resource managers

A *resource manager* is a software layer between a resource (a hardware or software entity that provides services to some other component) and RMC. As mentioned, a resource manager maps programmatic abstractions in RMC into the actual calls and commands of a resource.

RSCT provides a core set of resource managers. Resource managers provide low-level instrumentation and control, or act as a foundation for management applications.

These are the core resource managers of RSCT:

- Audit log resource manager
- Configuration resource manager
- Event resource manager
- File system resource manager
- Host resource manager
- Sensor resource manager

RSCT cluster security services

Cluster Security Services (ctsec) is the security infrastructure that is used by RMC to *authenticate* a node within the cluster, verifying that the node is who it says it is.

> **Note:** This is not to be confused with *authorization* (granting or denying access to resources), which is handled by RMC.

Cluster Security Services uses credential-based authentication that enables:

- A client process to present information to the server that owns the resource to be accessed in a way that cannot be imitated.
- A server process to clearly identify the client and the validity of the information.
- Credential-based authentication uses a third party that both the client and the server trust.

Group Services and Topology Services subsystems

Group Services and Topology Services, although included in RSCT, are not used in a management domain structure. These two components are used in high availability clusters for complete high availability and disaster recovery solutions, such as High-Availability Cluster Multi-Processing (HACMP), providing node and process coordination and node and network failure detection.

These services are often referred to as hats and hags: high availability Topology Services daemon (hatsd) and Group Services daemon (hagsd).

Additional detailed information about IBM Reliable Scalable Clustering Technology is available in *Reliable Scalable Cluster Technology: Administration Guide*, SA22-7889.

2.1.11 Dual IBM Virtual I/O Server

Dual IBM Virtual I/O Server (VIOS) provides the capability for a single physical I/O adapter to be used by multiple logical partitions of the same server, thus allowing consolidation of I/O resources and minimizing the number of I/O adapters required. The IBM Virtual I/O Server is the link between the virtual resources and physical resources. It is a specialized partition that owns the physical I/O resources, and runs in a special partition that cannot be used for execution of application code. It mainly provides two functions:

- Serves virtual SCSI devices to client partitions
- Provides a Shared Ethernet Adapter for VLANs

While redundancy can be built into the VIOS itself with the use of standard AIX tools like multipath I/O (MPIO) and AIX Logical Volume Manager (LVM) RAID Options for storage devices, and Ethernet link aggregation for network devices, the Virtual I/O Server must be available with respect to the client. Planned outages (such as software updates) and unplanned outages (such as hardware outages) challenge 24x7 availability. In case of a crash of the Virtual I/O Server, the client partitions will see I/O errors and not be able to access the adapters and devices that are hosted by the Virtual I/O Server.

The Virtual I/O Server itself can be made redundant by running a second instance in another partition. When running two instances of the Virtual I/O Server, you can use LVM mirroring, multipath I/O, Ethernet link aggregation, or multipath routing with dead gateway detection in the client partition to provide highly available access to virtual resources hosted in separate Virtual I/O Server partitions. Many configurations are possible; they depend on the available hardware resources as well as on your requirements.

For example, with the availability of MPIO on the client, each VIOS can present a virtual SCSI device that is physically connected to the same physical disk. This achieves redundancy for the VIOS itself and for any adapter, switch, or device that is used between the VIOS and the disk. With the use of logical volume mirroring on the client, each VIOS can present a virtual SCSI device that is physically connected to a different disk and then used in a normal AIX

Archived

mirrored volume group on the client. This achieves a potentially greater level of reliability by providing redundancy. Client volume group mirroring is also required when a VIOS logical volume is used as a virtual SCSI device on the Client. In this case, the virtual SCSI devices are associated with different SCSI disks, each controlled by one of the two VIOS.

As an example of network high availability, Shared Ethernet Adapter (SEA) failover offers Ethernet redundancy to the client at the virtual level. The client gets one standard virtual Ethernet adapter hosted by two VIO servers. The two Virtual I/O servers use a control channel to determine which of them is supplying the Ethernet service to the client. Through this active monitoring between the two VIOS, failure of either will result in the remaining VIOS taking control of the Ethernet service for the client. The client has no special protocol or software configured, and uses the virtual Ethernet adapter as though it was hosted by only one VIOS.

2.1.12 Special Uncorrectable Error handling

Although a rare occurrence, an uncorrectable data error can occur in memory or a cache, despite all precautions built into the server. In older generations of servers (prior to IBM POWER4™ processor-based offerings), this type of error would eventually result in a system crash. The IBM System p and System i offerings extend the POWER4 technology design and include techniques for handling these types of errors.

On these servers, when an uncorrectable error is identified at one of the many checkers strategically deployed throughout the system's central electronic complex, the detecting hardware modifies the ECC word associated with the data, creating a special ECC code. This code indicates that an uncorrectable error has been identified at the data source and that the data in the “standard” ECC word is no longer valid. The check hardware also signals the Service Processor and identifies the source of the error. The Service Processor then takes appropriate action to handle the error. This technique is called Special Uncorrectable Error (SUE) handling.

Simply detecting an error does not automatically cause termination of a system or partition. In many cases, an uncorrectable error will cause generation of a synchronous machine check interrupt. The machine check interrupt occurs when a processor tries to load the bad data. The firmware provides a pointer to the instruction that referred to the corrupt data, the system continues to operate normally, and the hardware observes the use of the data.

The system is designed to mitigate the problem using a number of approaches. For example, if the data is never actually used but is simply overwritten, then the error condition can safely be voided and the system will continue to operate normally.

For AIX V5.2 or greater, if the data is actually referenced for use by a process, then the operating system is informed of the error. The operating system will terminate only the specific user process associated with the corrupt data.

New with AIX V6.1

The POWER6 processor adds the ability to report the faulting memory address on a SUE Machine Check. This hardware characteristic, combined with the AIX V6.1 recovery framework, expands the cases in which AIX will recover from an SUE to include some instances when the error occurs in kernel mode.

Specifically, if an SUE occurs inside one of the copyin() and copyout() family of kernel services, these functions will return an error code and allow the system to continue operating (in contrast, on a POWER4 or POWER5 system, AIX would crash). The new SUE feature integrates the kernel mode handling of SUEs with the FRR recovery framework.

Note: The default kernel recovery framework setting is disabled. This means an affirmative action must be taken via SMIT or the **raso** command to enable recovery. When recovery is not enabled, the behavior will be the same as on AIX 5.3.

2.1.13 Automated system hang recovery

Automatic system hang recovery with error detection and fix capabilities are key features of the automated system management of AIX which can detect the condition that high priority processes are monopolizing system resources and prohibiting normal execution. AIX offers system administrators a variety of customizable solutions to remedy the system hang condition.

2.1.14 Recovery framework

Beginning with AIX V6.1, the kernel can recover from errors in selected routines, thus avoiding an unplanned system outage. The kernel recovery framework improves system availability. The framework allows continued system operation after some unexpected kernel errors.

Kernel recovery

Kernel recovery in AIX V6.1 is disabled by default. This is because the set of errors that can be recovered is limited in AIX V6.1, and kernel recovery, when enabled, requires an extra 4 K page of memory per thread. To enable, disable, or show kernel recovery state, use the SMIT path **Problem Determination** → **Kernel Recovery,** or use the **smitty krecovery** command.

You can show the current and next boot states, and also enable or disable the kernel recovery framework at the next boot. In order for the change to become fully active, you must run the **/usr/sbin/bosboot** command after changing the kernel recovery state, and then reboot the operating system.

During a kernel recovery action, the system might pause for a short time, generally less than two seconds. The following actions occur immediately after a kernel recovery action:

1. The system console displays the message saying that a kernel error recovery action has occurred.
2. AIX adds an entry into the error log.
3. AIX may generate a live dump.
4. You can send the error log data and live dump data to IBM for service (similar to sending data from a full system termination).

Note: Some functions might be lost after a kernel recovery, but the operating system remains in a stable state. If necessary, shut down and restart your system to restore the lost functions.

2.2 System reliability

Over the years the AIX operating system has included many reliability features inspired by IBM technology, and it now includes even more ground breaking technologies that add to AIX reliability. Some of these include kernel support for POWER6 storage keys, Concurrent AIX Update, dynamic tracing and enhanced software first failure data capture, just to mention a few new features.

2.2.1 Error checking

Run-Time Error Checking

The Run-Time Error Checking (RTEC) facility provides service personnel with a method to manipulate debug capabilities that are already built into product binaries. RTEC provides powerful first failure data capture and second failure data capture error detection features.

The basic RTEC framework is introduced in AIX V5.3 TL3, and has now been extended with additional features. RTEC features include the Consistency Checker and Xmalloc Debug features. Features are generally tunable with the **errctrl** command.

Some features also have attributes or commands specific to a given subsystem, such as the **sodebug** command associated with new socket debugging capabilities. The enhanced socket debugging facilities are described in the AIX publications, which can be found online at the following site:

http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp

Kernel stack overflow detection

Beginning with the AIX V5.3 TL5 package, the kernel provides enhanced logic to detect stack overflows. All running AIX code maintains an area of memory called a *stack*, which is used to store data necessary for the execution of the code. As the code runs, this stack grows and shrinks. It is possible for a stack to grow beyond its maximum size and overwrite other data.

These problems can be difficult to service. AIX V5.3 TL5 introduces an asynchronous run-time error checking capability to examine if certain kernel stacks have overflowed. The default action upon overflow detection is to log an entry in the AIX error log. The stack overflow run-time error checking feature is controlled by the ml.stack_overflow component.

AIX V6.1 improves kernel stack overflow detection so that some stacks are guarded with a synchronous overflow detection capability. Additionally, when the recovery framework is enabled, some kernel stack overflows that previously were fatal are now fully recoverable.

Kernel no-execute protection

Also introduced in the AIX V5.3 TL5 package, no-execute protection is set for various kernel data areas that should *never* be treated as executable code. This exploits the page-level execution enable/disable hardware feature. The benefit is immediate detection if erroneous device driver or kernel code inadvertently make a stray branch onto one of these pages. Previously the behavior would likely lead to a crash, but was undefined.

This enhancement improves kernel reliability and serviceability by catching attempts to execute invalid addresses immediately, before they have a chance to cause further damage or create a difficult-to-debug secondary failure. This feature is largely transparent to the user, because most of the data areas being protected should clearly be non-executable.

2.2.2 Extended Error Handling

In 2001, IBM introduced a methodology that uses a combination of system firmware and Extended Error Handling (EEH) device drivers that allow recovery from intermittent PCI bus errors. This approach works by recovering and resetting the adapter, thereby initiating system recovery for a permanent PCI bus error. Rather than failing immediately, the faulty device is “frozen” and restarted, preventing a machine check. POWER6 technology extends this capability to PCIe bus errors.

Archived

2.2.3 Paging space verification

Finding the root cause of system crashes, hangs or other symptoms when that root cause is data corruption can be difficult, because the symptoms can appear far downstream from where the corruption was observed. The page space verification design is intended to improve First Failure Data Capture (FFDC) of problems caused by paging space data corruption by checking that the data read in from paging space matches the data that was written out.

When a page is paged out, a checksum will be computed on the data in the page and saved in a pinned array associated with the paging device. If and when it is paged back in, a new checksum will be computed on the data that is read in from paging space and compared to the value in the array. If the values do not match, the kernel will log an error and halt (if the error occurred in system memory), or send an exception to the application (if it occurred in user memory).

Paging space verification can be enabled or disabled, on a per-paging space basis, by using the **mkps** and **chps** commands. The details of these commands can be found in their corresponding AIX **man** pages.

2.2.4 Storage keys

Most application programmers have experienced the inadvertent memory overlay problem where a piece of code accidentally wrote to a memory location that is not part of the component's memory domain. The new hardware feature, called storage protection keys, and referred to as *storage keys* in this paper, assists application programmers in locating these inadvertent memory overlays.

Memory overlays and addressing errors are among the most difficult problems to diagnose and service. The problem is compounded by growing software size and increased complexity. Under AIX, a large global address space is shared among a variety of software components. This creates a serviceability issue for both applications and the AIX kernel.

The AIX 64-bit kernel makes extensive use of a large flat address space by design. This is important in order to avoid costly MMU operations on POWER processors. Although this design does produce a significant performance advantage, it also adds reliability, availability and serviceability (RAS) costs. Large 64-bit applications, such as DB2®, use a global address space for similar reasons and also face issues with memory overlays.

Storage keys were introduced in PowerPC® architecture to provide memory isolation, while still permitting software to maintain a flat address space. The concept was adopted from the System z™ and IBM 390 systems. Storage keys allow an address space to be assigned context-specific protection. Access to the memory regions can be limited to prevent, and catch, illegal storage references.

A new CPU facility, Authority Mask Register (AMR), has been added to define the key set that the CPU has access to. The AMR is implemented as bit pairs vector indexed by key number, with distinct bits to control read and write access for each key. The key protection is in addition to the existing page protection bits. For any load or store process, the CPU retrieves the memory key assigned to the targeted page during the translation process. The key number is used to select the bit pair in the AMR that defines if an access is permitted.

A data storage interrupt occurs when this check fails. The AMR is a per-context register that can be updated efficiently. The TLB/ERAT contains storage key values for each virtual page. This allows AMR updates to be efficient, since they do not require TLB/ERAT invalidation.

The PowerPC hardware gives software a mechanism to efficiently change storage accessibility.

Storage-keys are exploited in both kernel-mode and user-mode APIs. In kernel-mode, storage-key support is known as *kernel keys*.

The APIs that manage hardware keys in user mode refer to the functionality as *user keys*. User key support is primarily being provided as a reliability, availability and serviceability (RAS) feature for applications. The first major application software to implement user keys is DB2. In DB2, user keys are used for two purposes. Their primary purpose is to protect the DB2 core from errors in user-defined functions (UDFs). The second use is as a debug tool to prevent and diagnose internal memory overlay errors. But this functionality is available to any application.

DB2 provides a UDF facility where customers can add extra code to the database. There are two modes that UDFs can run under, fenced and unfenced, as explained here:

- In fenced mode, UDFs are isolated from the database by execution under a separate process. Shared memory is used to communicate between the database and UDF process. Fenced mode does have a significant performance penalty, because a context switch is required to execute the UDF.
- An unfenced mode is also provided, where the UDF is loaded directly into the DB2 address space. Unfenced mode greatly improves performance, but introduces a significant RAS exposure.

Although DB2 recommends fenced mode, many customers use unfenced mode for improved performance. Use of user keys must provide significant isolation between the database and UDFs with low overhead.

User keys work with application programs. They are a virtualization of the PowerPC storage key hardware. User keys can be added and removed from a user space AMR, and a single user key can be assigned to an application's memory pages. Management and abstraction of user keys is left to application developers. The storage protection keys application programming interface (API) for user space applications is available in AIX V5.3 TL6 and is supported on all IBM System p POWER6 processor-based servers running this technology level.

Kernel keys are added to AIX as an important Reliability, Availability, and Serviceability (RAS) function. They provide a Reliability function by limiting the damage that one software component can do to other parts of the system. They will prevent kernel extensions from damaging core kernel components, and provide isolation between kernel extension classes.

Kernel keys will also help to provide significant Availability function by helping prevent error propagation—and this will be a key feature as AIX starts to implement kernel error recovery handlers. Serviceability is enhanced by detecting memory addressing errors closer to their origin. Kernel keys allow many random overlays to be detected when the error *occurs*, rather than when the corrupted memory is used.

With kernel key support, the AIX kernel introduces the concept of kernel domains and private memory access. *Kernel domains* are component data groups that are created to segregate sections of the kernel and kernel extensions from each other. Hardware protection of kernel memory domains is provided and enforced. Also, global storage heaps are separated and protected. This keeps heap corruption errors within kernel domains. There are also private memory keys that allow memory objects to be accessed only by authorized components. Besides the Reliability, Availability and Serviceability benefits, private memory keys are a tool to enforce data encapsulation.

Archived

Kernel keys are provided in AIX V6.1. Kernel keys will be a differentiator in the UNIX market place. It is expected that AIX will be the only UNIX operating system exploiting this type of memory protection.

2.3 System serviceability

IBM has implemented system serviceability in AIX to make it easier to perform problem determination, corrective maintenance, or preventive maintenance on a system.

2.3.1 Advanced First Failure Data Capture features

First Failure Data Capture (FFDC) is a serviceability technique whereby a program that detects an error preserves all the data required for subsequent analysis and resolution of the problem. The intent is to eliminate the need to wait for or to force a second occurrence of the error to allow specially-applied traps or traces to gather the data required to diagnose the problem.

The AIX V5.3 TL3 package introduced these new First Failure Data Capture (FFDC) capabilities. The set of FFDC features is further expanded in AIX V5.3 TL5 and AIX V6.1. These features are described in the sections that follow, and include:

- Lightweight Memory Trace (LMT)
- Run-Time Error Checking (RTEC)
- Component Trace (CT)
- Live Dump

These features are enabled by default at levels that provide valuable FFDC information with minimal performance impacts. The advanced FFDC features can be individually manipulated. Additionally, a SMIT dialog has been provided as a convenient way to persistently (across reboots) disable or enable the features through a single command. To enable or disable all four advanced FFDC features, enter the following command:

```
smitty ffdc
```

This specifies whether the advanced memory tracing, live dump, and error checking facilities are enabled or disabled. Note that disabling these features reduces system Reliability, Availability, and Serviceability.

> **Note:** You must run the **/usr/sbin/bosboot** command after changing the state of the Advanced First Failure Data Capture Features, and then reboot the operating system in order for the changes to become fully active. Some changes will not fully take effect until the next boot.

2.3.2 Traditional system dump

The system dump facility provides a mechanism to capture a snapshot of the operating system state. A *system dump* collects the system's memory contents and provides information about the AIX kernel that can be used later for expert analysis. After the preserved image is written to the dump device, the system will be booted and can be returned to production if desired. The system generates a system dump when a severe error occurs, that is, at the time of the system crash.

Archived

System dumps can also be user-initiated by root users. System administrators and programmers can generate a dump and analyze its contents when debugging new applications. The system dump is typically submitted to IBM support for analysis, along with other system configuration information. Typically, an AIX system dump includes all of the information needed to determine the nature of the problem. The dump contains:

- Operating system (kernel) code and kernel data
- Kernel data about the current running application on each CPU
- Most of the kernel extensions code and data for all running kernel extensions

AIX V6.1 also provides additional features to reduce the size of dumps and the time needed to create a dump. It is possible to control which components participate in the system dump. It may be desirable to exclude some components from the system dump in order to decrease the dump size.

> **Note:** As of AIX V6.1, traditional system dumps are always compressed to further reduce size.

The **dumpctrl** command obtains information about which components are registered for a system dump. If a problem is being diagnosed, and multiple system dumps are needed, components that are not needed can be excluded from system dumps until the problem is solved. When the problem is solved, the system administrator should again enable all system dump components.

AIX V6.1 provides also a *live dump* capability to allow failure data to be dumped without taking down the entire system. A live dump will most likely involve just a few system components. For example, prior to AIX V6.1, if an inconsistency were detected by a kernel component such as a device driver, the usual approach is to bring down and dump the entire system.

System dumps can now be copied to DVD media. You can also use DVD as a primary or secondary dump device. Note that the **snap** command can use a DVD as source, as well as an output device.

2.3.3 Firmware-assisted system dump

Firmware-assisted dump improves the reliability of the traditional system dump by minimizing work done by the failing operating system.The idea is to freeze memory and reboot the system prior to dumping the memory to disk. *Firmware-assisted system dump* means an AIX dump assisted by firmware, which is taken when the partition is restarting. The firmware is involved to preserve memory across the reboot, and eventually to save parts of memory and processor registers.

> **Important:** In AIX V6.1, traditional dump remains the default method for performing a system dump in all configurations.
>
> Firmware-assisted system dump should only be used if you are directed to do so by IBM Service during problem determination.

Selective memory dump is a firmware-assisted system dump that is triggered by (or uses) the AIX instance. Full memory dump is a firmware-assisted system dump that dumps all partition memory without any interaction with the AIX instance that is failing. Both selective-memory dump and traditional system dump require interaction with the failing AIX instance to complete the dump.

Archived

As system memory increases, so does the time required to complete a dump. For this reason, a secondary goal of the firmware-assisted system dump feature is to reduce the overall system outage time by taking the dump in parallel with the system restart.

Note: Firmware-assisted system dump requires a POWER6 system with at least 4 GB of memory assigned to the partition. Firmware-assisted dump does not support all disk adapters. In addition, certain multi-path configurations are not supported.

In order to allow a firmware-assisted system dump, AIX must perform configuration operations both at startup and at the dump configuration:

1. AIX checks the memory size at system startup to determine whether there is enough available to enable firmware-assisted system dump.
2. AIX retrieves the firmware-assisted system dump property in the device tree to determine whether firmware-assisted system dump is supported by the platform.
3. AIX verifies that the administrator has enabled firmware-assisted dump.

If all conditions for a firmware-assisted system dump are validated, AIX reserves a scratch area and performs the firmware-assisted system dump configuration operations. The scratch area is not released unless the administrator *explicitly* reconfigures a traditional system dump configuration. Verification is not performed when a dynamic reconfiguration operation modifies the memory size.

AIX can switch from firmware-assisted system dump to traditional system dump at dump time because traditional system dump is always initialized. There are two cases of traditional system dump: a user-specified traditional system dump configuration, and a traditional system dump that is initialized just in case AIX cannot start a firmware-assisted system dump.

AIX can be configured to choose the type of dump between firmware-assisted system dump and traditional system dump. When the configuration of the dump type is changed from firmware-assisted system dump to traditional system, the new configuration is effective immediately. When the configuration of the dump type is changed from traditional system dump to firmware-assisted system dump, the new configuration is only effective *after* a reboot.

When firmware-assisted system dump is supported by the platform and by AIX, and is activated by the administrator, selective memory dump is the default dump configuration. Full memory dump is not allowed by default. In case of firmware-assisted system dump configuration, the administrator can configure AIX to allow a full memory dump. If allowed but not required, the full memory dump is only performed when AIX cannot initiate a selective memory dump.

The administrator can configure AIX to require a full memory dump even if AIX can initiate a selective memory dump. In both cases, where full memory dump is either allowed or required, the administrator can start a full memory dump from AIX or from the HMC menus.

2.3.4 Live dump and component dump

Component dump allows the user to request that one or more components (their associated memory and kernel data structures) be dumped. When combined with the live dump functionality, component dump functionality allows components to be dumped, without bringing down the entire system. When combined with the system dump functionality, this allows you to limit the size of the system dump.

Live dumps are small dumps that do not require a system restart. Only components that are registered for live dumps, and are requested to be included, are dumped. Use the **dumpctrl** command to obtain information about which components are registered for live dumps.

Live dumps can be initiated by software programs or by users with root user authority. Software programs use live dumps as part of recovery actions, or when the runtime error-checking value for the error disposition is ERROR_LIVE_DUMP. If you have root user authority, you can initiate live dumps when a subsystem does not respond or behaves erroneously. For more information about how to initiate and manage live dumps, see the **livedumpstart** and **dumpctrl** commands in AIX V6.1 command reference manuals, which are downloadable at the following site:

http://publib.boulder.ibm.com/infocenter/pseries/v6r1/index.jsp

Unlike system dumps, which are written to a dedicated dump device, live dumps are written to the file system. By default, live dumps are placed in the /var/adm/ras/livedump directory. The directory can be changed by using the **dumpctrl** command.

In AIX V6.1, only serialized live dumps are available. A *serialized* live dump causes a system to be frozen or suspended when data is being dumped. The freeze is done by stopping all processors, except the processor running the dump. When the system is frozen, the data is copied to the live dump heap in pinned kernel memory. The data is then written to the file system, but only after the system is unfrozen. Live dump usually freezes the system for no more than 100 ms.

The **heapsz** attribute (heap size) can be set to zero (0), meaning that at dump initialization time, the system calculates the live dump heap size based on the amount of real memory, which is the minimum of 16 MB or 1/64 the size of real memory (whichever is smaller).

Duplicate live dumps that occur rapidly are eliminated to prevent system overload and to save file system space. Eliminating duplicate dumps requires periodic (once every 5 minutes) scans of the live dump repository through a cron job. Duplicate elimination can be stopped via the **dumpctrl** command.

Each live dump has a data priority. A live dump of *info* priority is for informational purposes, and a live dump of *critical* priority is used to debug a problem. Info priority dumps can be deleted to make room for critical priority dumps.

You can enable or disable all live dumps by using the **dumpctrl ldmpon/ldmpoff** command, or by using the SMIT fastpath:

```
smitty livedump
```

> **Note:** Persistent options can be used to set state on further boots.

2.3.5 The dumpctrl command

There is also the need to modify component, live, and system dump properties from one command. To achieve this, the **dumpctrl** command has been implemented in AIX V6.1. The **dumpctrl** command is used to query and modify global system and live dump properties, as well as per-component system and live dump properties, where appropriate.

> **Note:** Live dumps are compressed and must be uncompressed with the **dmpumcompress** command.

2.3.6 Parallel dump

An AIX system generates a system dump when a severe unrecoverable error occurs. System dumps can also be user-initiated by users with root user authority. A system dump creates a picture of your system's memory contents. However, systems have an increasingly large amount of memory and CPUs, and larger systems experience longer dump times. For this reason a new feature, parallel dump, was introduced in AIX V5.3 TL5.

A new optimized compressed dump format has been introduced in AIX V5.3. The dump file extension for this new format is **.BZ**. In this new compressed dump file, the blocks are compressed and unordered; this unordered feature allows multiple processors to dump parallel sub-areas of the system. Parallel dumps are produced automatically when supported by the AIX release.

> **Important:** The new file format for parallel dump is *not* readable with **uncompress** and **zcat** commands.
>
> The new **dmpuncompress** command must be used instead.

In order to increase dump reliability, a new **-S** checking option, to be used with the **-L** option for the statistical information on the most recent dump, is also added to the **sysdumpdev** command. The **-S** option scans a specific dump device and sees whether it contains a valid compressed dump.

```
sysdumpdev -L -S <Device>
```

The dump must be from an AIX release with parallel dump support. This flag can be used only with the **-L** flag. Additional options can be found and modified in SMIT via **smitty dump.**

2.3.7 Minidump

The minidump, introduced in AIX V5.3 TL3, is a small compressed dump that is stored to NVRAM when the system crashes or a dump is initiated, and is then written to the error log on reboot. It can be used to see some of the system state and do some debugging if a full dump is not available. It can also be used to obtain a quick snapshot of a crash without having to transfer the entire dump from the crashed system.

Minidumps will show up as error log entries with a label of **MINIDUMP_LOG** and a description of **COMPRESSED MINIMAL DUMP**. To filter the minidump entries in the error log, you can use the **errpt** command with the **–J** flag (**errpt [–a] –J MINIDUMP_LOG**). Minidumps in the error log can be extracted, decompressed, and formatted using the **mdmprpt** command, as shown:

```
mdmprpt [-l seq_no] [-i filename] [-r]
```

More detailed information about minidump is available at the following site:

http://igets3.fishkill.ibm.com/DCF/isg/isgintra.nsf/all/T1000676?OpenDocument&Highlight=0,minidump

2.3.8 Trace (system trace)

The AIX trace facility is a powerful system observation tool. The trace facility captures a sequential flow of time-stamped system events, thus providing a fine level of detail of system activity. Events are shown in time sequence and in the context of other events.

Archived

Trace is a valuable tool for observing system and application execution. Where other tools provide high level statistics such as CPU utilization or I/O wait time, the trace facility is useful in expanding the information to understand the who, when, how, and why of an event.

Single thread trace

In prior versions of AIX, system trace would trace the entire system. In AIX V5.3, the system trace facility has been enhanced by new flags. This enables the trace to run only for specified processes, threads, or programs. The system trace can be used to trace the processor utilization register (PURR) to provide more accurate event timings in a shared processor partition environment.

Administrative control of the user trace buffers

Also, in previous versions of AIX, the trace buffer size for a regular user is restricted to a maximum of 1 MB. Version 5.3 allows the system group users to set the trace buffer size either through a new command, **trcctl**, or using a new SMIT menu called Manage Trace:

```
smitty trace → Manage Trace
```

To check the actual trace subsystem properties, use the **trcctl** command, as shown in Example 2-2.

Example 2-2 Trace characteristics

```
lpar15root:/root#trcctl

Default Buffer Size: 262144
Default Log File Size: 2621440
Default Log File: /var/adm/ras/trcfile
Non-Root User Buffer Size Maximum: 1048576
Default Components Directory: /var/adm/ras/trc_ct
Default LMT Log Dir: /var/adm/ras/mtrcdir
```

To list all trace event groups, you can use the following command:

```
trcevgrp -l
```

There are several trace-related tools which are used for various operating system components:

- CPU monitoring **tprof, curt, splat, trace, trcrpt**
- Memory monitoring **trace, trcrpt**
- I/O subsystem **trace, trcrpt**
- Network **iptrace, ipreport, trace, trcrpt**
- Processes and threads **tprof, pprof, trace, trcrpt**

Trace can be used in two ways: interactively, or asynchronously, as explained here:

- Interactively

  The following sequence of commands runs an interactive trace on the program myprog and ends the trace:

  ```
  trace -j30D,30E -o trace.file
  ->!myprog
  ->q
  ```

Archived

- Asynchronously

  The following sequence of commands runs an asynchronous trace on the program myprog and ends the trace:

  ```
  trace -a -j30D,30E
  myprog
  trcstop
  ```

- You can format the trace file with the following command:

  ```
  trcrpt -o output.file
  ```

Additional details about these trace options are available in *IBM AIX Version 6.1 Differences Guide*, SC27-7559.

POSIX trace

AIX Version 6 implements the POSIX trace system, which supports tracing of user applications via a standardized set of interfaces. The POSIX tracing facilities allow a process to select a set of trace event types, activate a trace stream of the selected trace events as they occur in the flow of execution, and retrieve the recorded trace events. Like system trace, POSIX trace is also dependent upon precompiled-in trace hooks in the application being instrumented.

Additional details about these trace options are available in *IBM AIX Version 6.1 Differences Guide*, SC27-7559.

Iptrace

The **`iptrace`** daemon provides interface-level packet tracing for Internet protocols. This daemon records Internet packets received from configured interfaces. Command flags provide a filter so that the daemon traces only packets that meet specific criteria. Packets are traced only between the local host on which the **`iptrace`** daemon is invoked and the remote host.

If the iptrace process was started from a command line without the System Resource Controller (SRC), it must be stopped with the **`kill -15`** command. The kernel extension loaded by the **`iptrace`** daemon remains active in memory if iptrace is stopped any other way.

The LogFile parameter specifies the name of a file to which the results of the **`iptrace`** command are sent. To format this file, run the **`ipreport`** command.

The **`ipreport`** command may display the message `TRACING DROPPED xxxx PACKETS`. This count of dropped packets indicates only the number of packets that the **`iptrace`** command was unable to grab because of a large packet whose size exceeded the socket-receive buffer size. The message does *not* mean that the packets are being dropped by the system.

2.3.9 Component Trace facility

The Component Trace (CT) facility allows the capture of information about a specific kernel component, kernel extension, or device driver. Component Trace is an important FFDC and SFDC tool available to the kernel, kernel extensions, and device drivers. The CT facility allows a component to capture trace events to aid in both debugging and system analysis, and provide focused trace data on larger server systems.

CT provides system trace information for specific system components. This information allows service personnel to access component state information through either in-memory trace buffers or through traditional AIX system trace. Component Trace is enabled by default.

Component Trace uses mechanisms similar to system trace. Existing TRCHKxx and TRCGEN macros can be replaced with Component Trace macros to trace into system trace buffers or memory trace mode private buffers. These macros are CT_HOOKx and CT_GEN, located in /usr/include/sys/ras_trace.h. Once recorded, Component Trace events can be retrieved by using the **ctctrl** command. Extraction using the **ctctrl** command is relevant only to in-memory tracing. Component Trace events can also be present in a system trace. The **trcrpt** command is used in both cases to process the events.

2.3.10 Lightweight Memory Trace (LMT)

Lightweight Memory Trace is an efficient, default-on per CPU, in-memory kernel trace. It is built upon the trace function that already exists in kernel subsystems, and is of most use for those who have AIX source-code access or a deep understanding of AIX internals.

LMT provides system trace information for First Failure Data Capture (FFDC). It is a constant kernel trace mechanism that records software events occurring during system operation. The system activates LMT at initialization, then tracing runs continuously. Recorded events are saved into per-processor memory trace buffers. There are two memory trace buffers for each processor—one to record common events, and one to record rare events. The memory trace buffers can be extracted from system dumps accessed on a live system by service personnel. The trace records look like traditional AIX system trace records. The extracted memory trace buffers can be viewed with the **trcrpt** command, with formatting as defined in the /etc/trcfmt file.

For further details about LMT, refer to 3.2, “Lightweight memory trace” on page 57.

2.3.11 ProbeVue

AIX V6.1 provides a new dynamic tracing facility that can help to debug complex system or application code. This dynamic tracing facility is introduced via a new tracing command, **probevue**, that allows a developer or system administrator to dynamically insert trace probe points in existing code without having to recompile the code. ProbeVue is described in detail in 3.8, “ProbeVue” on page 111. To show or change the ProbeVue configuration, use the following command:

```
smitty probevue
```

2.3.12 Error logging

Troubleshooting system problems is an important and challenging task for system administrators. AIX provides an error logging facility for the runtime recording of hardware and software failures in an error log. This error log can be used for informational purposes, or for fault detection and corrective actions.

The purpose of error logging is to collect and record data related to a failure so that it can be subsequently analyzed to determine the cause of the problem. The information recorded in the error log enables the customer and the service provider to rapidly isolate problems, retrieve failure data, and take corrective action.

Error logging is automatically started by the **rc.boot** script during system initialization. Error logging is automatically stopped by the **shutdown** script during system shutdown.

The error logging process begins when the AIX operating system module detects an error. The error-detecting segment of code then sends error information to either the **errsave** kernel

Archived

service and **errlast** kernel service for a pending system crash, or to the **errlog** subroutine to log an application error, where the information is, in turn, written to the /dev/error special file.

The **errlast** kernel service preserves the last error record in the NVRAM. Therefore, in the event of a system crash, the last logged error is not lost. This process then adds a time stamp to the collected data.

The **errdemon** daemon constantly checks the /dev/error file for new entries, and when new data is written, the daemon conducts a series of operations. Before an entry is written to the error log, the **errdemon** daemon compares the label sent by the kernel or application code to the contents of the error record template repository. If the label matches an item in the repository, the daemon collects additional data from other parts of the system.

To create an entry in the error log, the **errdemon** daemon retrieves the appropriate template from the repository, the resource name of the unit that detected the error, and detailed data. Also, if the error signifies a hardware-related problem and the Vital Product Data (VPD) hardware exists, the daemon retrieves the VPD from the Object Data Manager (ODM).

When you access the error log, either through SMIT or by using the **errpt** command, the error log is formatted according to the error template in the error template repository and presented in either a summary or detailed report.

Most entries in the error log are attributable to hardware and software problems, but informational messages can also be logged. The **errlogger** command allows the system administrator to record messages of up to 1024 bytes in the error log.

Whenever you perform a maintenance activity, such as clearing entries from the error log, replacing hardware, or applying a software fix, it is good practice to record this activity in the system error log; here is an example:

```
errlogger system hard disk '(hdisk0)' replaced.
```

This message will be listed as part of the error log.

2.3.13 The alog facility

The **alog** is a facility (or command) used to create and maintain fixed-size log files. The **alog** command can maintain and manage logs. It reads standard input, writes to standard output, and copies the output into a fixed-size file simultaneously. This file is treated as a circular log. If the file is full, new entries are written over the oldest existing entries.

The **alog** command works with log files that are specified on the command line, or with logs that are defined in the **alog** configuration database. Logs that are defined in the **alog** configuration database are identified by LogType. The File, Size, and Verbosity attributes for each defined LogType are stored in the **alog** configuration database with the LogType. The **alog** facility is used to log boot time messages, install logs, and so on.

You can use the **alog -L** command to display all log files that are defined for your system. The resulting list contains all logs that are viewable with the **alog** command. Information saved in BOS installation log files might help you determine the cause of installation problems. To view BOS installation log files, enter:

```
alog -o -f bosinstlog
```

All boot messages are collected in a boot log file, because at boot time there is no console available. Boot information is usually collected in /var/adm/ras/bootlog. It is good practice to check the bootlog file when you are investigating boot problems. The file will contain output generated by the **cfgmgr** command and **rc.boot**.

To display the boot log, use the following command:

```
alog -t boot -o | more
```

Alternatively, you can use the SMIT fastpath **smitty alog** menu shown in Example 2-3.

Example 2-3 SMIT alog menu

```
                           Alog

Move cursor to desired item and press Enter.

  Show an Alog File
  Change / Show Characteristics of an Alog File



F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Example 2-4 shows a list of available alog files, located in /var/adm/ras/.

Example 2-4 List of available alog files

```
                       Show an Alog File

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* +--------------------------------------------------------------------------+
  |                              Alog TYPE                                   |
  |                                                                          |
  | Move cursor to desired item and press Enter.                             |
  |                                                                          |
  |   boot                                                                   |
  |   bosinst                                                                |
  |   nim                                                                    |
  |   console                                                                |
  |   cfg                                                                    |
  |   lvmcfg                                                                 |
  |   lvmt                                                                   |
  |   dumpsymp                                                               |
  |                                                                          |
  | F1=Help                 F2=Refresh              F3=Cancel                |
F1| F8=Image                F10=Exit                Enter=Do                 |
F5| /=Find                  n=Find Next                                      |
F9+--------------------------------------------------------------------------+
```

2.3.14 syslog

The **syslogd** daemon logs the system messages from different software components (kernel, daemon processes, and system applications). This daemon uses a configuration file to determine where to send a system message, depending on the message's priority level and the facility that generated it. By default, syslogd reads the default configuration file /etc/syslog.conf, but by using the **-f** flag when starting syslogd, you can specify an alternate

configuration file. Whenever you change this configuration file, you need to refresh the syslogd subsystem, as follows:

```
refresh -s syslogd
```

The syslogd daemon reads a datagram socket and sends each message line to a destination described by the /etc/syslog.conf configuration file. The syslogd daemon reads the configuration file when it is activated and when it receives a hang-up signal.

Each message is one line. A message can contain a priority code marked by a digit enclosed in angle braces (< >) at the beginning of the line. Messages longer than 900 bytes may be truncated.

The /usr/include/sys/syslog.h include file defines the facility and priority codes used by the configuration file. Locally-written applications use the definitions contained in the syslog.h file to log messages using the syslogd daemon. For example, a configuration file that contains the following line is often used when a daemon process causes a problem:

```
daemon.debug /tmp/syslog.debug
```

This line indicates that a facility daemon should be controlled. All messages with the priority level debug and higher should be written to the file /tmp/syslog.debug.

Note: The file /tmp/syslog.debug must exist.

The daemon process that causes problems (in our example, the inetd) is started with option **-d** to provide debug information. This debug information is collected by the syslogd daemon, which writes the information to the log file /tmp/syslog.debug.

Logging can also improve security on your system by adding additional logging to the syslog.conf file. You can then increase system logging by adding extra logging for your system by editing inetd.conf and adding logging into each of the demons as required.

Example 2-5 shows a sample of syslog.conf settings.

Example 2-5 syslog settings

```
bianca:/etc/#vi syslog.conf
*.info          /var/adm/syslog/syslog.log
*.alert         /var/adm/syslog/syslog.log
*.notice        /var/adm/syslog/syslog.log
*.warning       /var/adm/syslog/syslog.log
*.err           /var/adm/syslog/syslog.log
*.crit          /var/adm/syslog/syslog.log rotate time 1d files 9
```

The last line in Example 2-5 indicates that this will create only 9 files and use a rotation on syslog. A file designated for storing syslog messages must exist; otherwise, the logging will not start. Remember to refresh syslog after any changes are made to the syslog configuration file.

Extra logging on demons controlled by inetd.conf can be configured as shown in Example 2-6 (ftpd is shown).

Example 2-6 Verbose logging for ftpd

```
bianca:/etc/#vi inetd.con

ftp     stream  tcp6    nowait  root    /usr/sbin/ftpd        ftpd -l
```

Logging can be added to many of the components in inetd.conf. Remember to refresh the inetd after you are done.

2.3.15 Concurrent AIX Update

Historically, servicing defects in the kernel has required some sort of interruption to system services in order to correct the problem. Typically, the interruption is in the form of a system reboot. This results in significant disruption to production systems, and has numerous implications to customers, including:

- The temporary loss of system availability and services
- An increased load on other systems
- A possible requirement of an unplanned maintenance window
- Monetary costs resulting from the loss of system availability

In addition, in some situations a diagnostic kernel is produced in order to analyze a problem, and multiple reboots become necessary. The Concurrent AIX Update feature for AIX V6.1 allows fixes to the base kernel and kernel extensions to be applied and simultaneously become fully operational on a running system. The system does *not* require any subsequent reboot to activate fixes. Concurrent AIX Update's ability to update a running system provides the following significant advantages:

- The ability to apply either preventive maintenance or corrective service fix without requiring a reboot
- The ability to reject an applied fix without requiring a reboot
- The ability to inject diagnostics or temporary circumventions without requiring a reboot
- Encouraging customer adoption of corrective service fixes, thereby decreasing outages for which fixes existed, but which would not have formerly been applied, due to the disruption to availability by the prior update scheme
- Improved system uptime
- Improved customer satisfaction
- A more convenient approach

Concurrent AIX Update enables activation and deactivation of IBM fixes to the kernel and kernel extensions. It accomplishes this by adding new capabilities to the interim fix packaging and installation tools, the system loader, and to the system process component.

> **Note:** The ability to apply or remove a fix without the requirement of a reboot is limited to Concurrent AIX Updates. Technological restrictions prevent some fixes from being made available as a Concurrent AIX Update. In such cases, those fixes may be made available as an interim fix (that is, "traditional" ifix).
>
> Traditional interim fixes for the kernel or kernel extensions still require a reboot of the operating system for both activation and removal.

Performing live updates on an operating system is a complicated task, and it places stringent demands on the operating system. There are many different approaches available for patching the operating system kernel. Concurrent AIX Update uses a method of functional redirection within the in-memory image of the operating system to accomplish patching-in of corrected code. After a fix for a problem has been determined, the corrected code is built, packaged, and tested according to a new process for Concurrent AIX Update. It is then provided to the customer, using the existing interim fix package format.

The package will contain one or more object files, and their corresponding executable modules. Patch object files have numerous restrictions, including (but not limited to) that no non-local data be modified, and that changes to multiple functions are only permitted if they

are valid when applied in a serial manner. The customer will manage Concurrent AIX Updates using the **emgr** command.

Installation requires specification of new options specific to Concurrent AIX Updates. Installation of a concurrent update will always perform an in-memory update. However, the customer can chose to additionally perform an on-disk update. Executable modules are provided for this purpose. The patch is then unpacked on the customer's system, and verified to be of correct version for the system. After verification, it is loaded into a reserved place within the kernel's allocated memory. The system loader, via new functionality, links concurrent update objects with the in-memory image of the operating system. Linkage performs symbol resolution and relocation of the patchset.

Defective functions that are to be patched have their first instruction saved aside and then subsequently replaced with a branch. The branch serves to redirect calls to a defective function to the corrected (patched) version of that function. To maintain system coherency, instruction replacements are collectively performed under special operation of the system.

After patching is successfully completed, the system is fully operational with corrected code, and no reboot is required. To remove a Concurrent AIX Update from the system, the saved-aside instructions are simply restored. Again, no reboot is required.

This method is suitable to the majority of kernel and kernel extension code, including interrupt handlers, locking code, and possibly even the concurrent update mechanism itself.

2.3.16 Core file control

A core file is created in the current directory when various errors occur. Errors such as memory-address violations, illegal instructions, bus errors, and user-generated quit signals, commonly cause this core dump. The core file that is created contains a memory image of the terminated process. If the faulty process is multi-threaded and the current core size ulimit is less than what is required to dump the data section, then only the faulting thread stack area is dumped from the data section.

Two new commands, **lscore** and **chcore**, have been introduced to check the settings for the corefile creation and change them, respectively. SMIT support has also been added; the fastpath is **smitty corepath**.

For AIX 5.3, the **chcore** command can change core file creation parameters, as shown:

```
chcore [ -R registry ] [ -c {on|off|default} ] [ -p {on|off|default} ] [ -l {path|
default] [ -n {on|off|default} ] [ username | -d ]
    -c {on|off|default} Setting for core compression.
    -d Changes the default setting for the system.
    -l path Directory path for stored corefiles.
    -n {on|off|default} Setting for core naming.
    -p {on|off|default} Setting for core location.
    -R registry Specifies the loadable I&A module.
```

New features have been added to control core files that will avoid key file systems being filled up by core files generated by faulty programs. AIX allows users to compress the core file and specify its name and destination directory.

The **chcore** command, as shown in Example 2-7, can be used to control core file parameters.

Example 2-7 Core file settings

```
lpar15root:/root#chcore -c on -p on -l /coredumps
lpar15root:/root#lscore
```

```
compression: on
path specification: on
corefile location: /coredumps
naming specification: off
```

In Example 2-7 on page 35, we have created a file system to store core files generated on the local system. This file system is mounted in the /coredumps directory. Next we created a core file for a program (`sleep 5000`), as shown in Example 2-8. We send the program to the background, so we get the process id, then we kill the program with Abort (**-6**) flag, and observe the core file.

Example 2-8 Core file example

```
lpar15root:/root#sleep 5000 &
[1]     397486
lpar15root:/root#kill -6 397486
lpar15root:/root#
[1] + IOT/Abort trap(coredump) sleep 5000 &
lpar15root:/root#ls -l /coredumps
total 16
-rw-------    1 root     system         7188 Oct 27 07:16 core.397486.27121636
drwxr-xr-x    2 root     system          256 Oct 27 07:07 lost+found
lpar15root:/root#lquerypv -h /coredumps/core.397486.27121636 6b0 64
000006B0   7FFFFFFF FFFFFFFF 7FFFFFFF FFFFFFFF  |................|
000006C0   00000000 00000FA0 7FFFFFFF FFFFFFFF  |................|
000006D0   00120000 28E1D420 00000000 00000003  |....(.. ........|
000006E0   736C6565 70000000 00000000 00000000  |sleep...........|
000006F0   00000000 00000000 00000000 00000000  |................|
00000700   00000000 00000000 00000000 0000002B  |...............+|
00000710   00000000 00000001 00000000 0000002B  |...............+|
```

2.4 Network tools

Networking is the means for a user to access system resources, application and data. Without network connectivity, the system is practically unavailable. AIX ensures maximum and efficient network performance and system availability to meet user needs by controlling computing resources in order to minimize network problems and facilitate system administration procedures.

This section covers the following network-related utilities currently available in AIX:

- Virtual IP address support
- Multipath IP routing
- Dead gateway detection
- EtherChannel
- IEEE 802.3ad Link Aggregation
- 2-Port Adapter-based Ethernet Failover
- Shared Ethernet Failover

Additional details and configuration information on these topics can be found in *AIX 5L Differences Guide Version 5.2 Edition,* SG24-5765, *AIX 5L Differences Guide Version 5.3 Edition,* SG24-7463, and *IBM AIX Version 6.1 Differences Guide*, SC27-7559.

2.4.1 Virtual IP address support (VIPA)

Prior to AIX V5.1, separate applications were required to provide high availability for a service IP address and its associated interface. If the network interface failed, then the application's TPCP/IP session was often lost, resulting in the loss of application availability.

To overcome this, support for virtual IP addresses (VIPA) on both IPv4 and IPv6 was introduced in AIX V5.1. VIPA allows the application to bind to a system-wide level virtual IP address, as opposed to a single network interface. VIPA is a virtual device often utilizing several network interfaces. VIPA can often mask underlying network interface failures by re-routing automatically to a different one. This allows continued connectivity and is transparent to the application and processes. VIPA also supports load balancing of traffic across the available connections.

Another advantage of choosing a virtual device (as opposed to defining aliases to real network interfaces) is that a virtual device can be brought up or down separately without having any effect on the real interfaces of a system. Furthermore, it is not possible to change the address of an alias (aliases can only be added and deleted), but the address of a virtual interface can be changed at any time.

Since its initial introduction in AIX V5.1, VIPA has been enhanced to make it friendlier, from a network administration perspective. It has also been enhanced so that failovers are completed faster, thus further improving availability.

2.4.2 Multipath IP routing

Prior to AIX V5.1, a new route could be added to the routing table only if it was different from the existing routes. The new route would have to be different by either destination, netmask, or group ID.

Also, previous AIX releases did not provide any mechanism to associate a specific interface with a route. When there were multiple interfaces on the same subnet, the same outgoing interface for all destinations accessible through that network was always chosen.

In order to configure a system for network traffic load balancing, it is desirable to have multiple routes so that the network subsystem routes network traffic to the same network segment by using different interfaces.

With the new multipath routing feature in AIX V6.1, routes no longer need to have a different destination, netmask, or group ID list. If there are several routes that equally qualify as a route to a destination, AIX will use a cyclic multiplexing mechanism (round-robin) to choose between them. The benefit of this feature is two-fold:

- It enables load balancing between two or more gateways.
- The feasibility of load balancing between two or more interfaces on the same network can be realized. The administrator would simply add several routes to the local network, one through each interface.

Multipath routing is often utilized together with dead gateway detection.

2.4.3 Dead gateway detection

The dead gateway detection (DGD) feature introduced in AIX V5.1 implements a mechanism for hosts to detect a dysfunctional gateway, adjust its routing table accordingly, and reroute

network traffic to an alternate backup route, if available. DGD is generally most useful for hosts that use static rather than dynamic routing.

AIX releases prior to AIX V5.1 do not permit you to configure multiple routes to the same destination. If a route's first-hop gateway failed to provide the required routing function, AIX continued to try to use the broken route and address the dysfunctional gateway. Even if there was another path to the destination which would have offered an alternative route, AIX did not have any means to identify and switch to the alternate route unless a change to the kernel routing table was explicitly initiated, either manually or by running a routing protocol program, such as `gated` or `routed`. Gateways on a network run routing protocols and communicate with one another. If one gateway goes down, the other gateways will detect it, and adjust their routing tables to use alternate routes (only the hosts continue to try to use the dead gateway).

The DGD feature in AIX V5.1 enables host systems to sense and isolate a dysfunctional gateway and adjust the routing table to make use of an alternate gateway without the aid of a running routing protocol program.

There are two modes for dead gateway detection:

- Passive dead gateway detection
- Active dead gateway detection

Passive dead gateway detection

Passive dead gateway detection will work without actively pinging the gateways known to a given system. Passive DGD will take action to use a backup route if a dysfunctional gateway has been detected.

The passive DGD mechanism depends on the protocols Transmission Control Protocol (TCP) and Address Resolution Protocol (ARP), which provide information about the state of the relevant gateways. If the protocols in use are unable to give feedback about the state of a gateway, a host will never know that a gateway is down and no action will be taken.

Passive dead gateway detection has low overhead and is recommended for use on any network that has redundant gateways. However, passive DGD is done on a best-effort basis only.

Active dead gateway detection

When no TCP traffic is being sent through a gateway, passive DGD will not sense a dysfunctional state of the particular gateway. The host has no mechanism to detect such a situation until TCP traffic is sent or the gateway's ARP entry times out, which may take up to 20 minutes. But this situation does not modify route costs. In other words, a gateway not forwarding packets is not considered dead. In such cases, active DGD becomes valuable to use.

Active DGD will ping gateways periodically, and if a gateway is found to be down, the routing table is changed to use alternate routes to bypass the dysfunctional gateway. Active dead gateway detection will be off by default and it is recommended to be used only on machines that provide critical services and have high availability requirements. Because active DGD imposes some extra network traffic, network sizing and performance issues have to receive careful consideration. This applies especially to environments with a large number of machines connected to a single network.

2.4.4 EtherChannel

EtherChannel is a network interface aggregation technology that allows you to produce a single large pipe by combining the bandwidth of multiple Ethernet adapters. In AIX V5.1, the

Archived

EtherChannel feature has been enhanced to support the detection of interface failures. This is called *network interface backup*.

EtherChannel is a trademark registered by Cisco Systems and is generally called *multi-port trunking* or *link aggregation*. If your Ethernet switch device supports this function, you can exploit the facility provided in AIX V5.1. In this case, you must configure your Ethernet switch to create a channel by aggregating a series of Ethernet ports.

EtherChannel allows for multiple adapters to be aggregated into one virtual adapter, which the system treats as a normal Ethernet adapter. The IP layer sees the adapters as a single interface with a shared MAC and IP address. The aggregated adapters can be a combination of any supported Ethernet adapter, although they must be connected to a switch that supports EtherChannel. All connections must be full-duplex and there must be a point-to-point connection between the two EtherChannel-enabled endpoints.

EtherChannel provides increased bandwidth, scalability, and redundancy. It provides aggregated bandwidth, with traffic being distributed over all adapters in the channel rather than just one. To increase bandwidth, the only requirement is to add more adapters to the EtherChannel, up to a maximum of eight physical devices.

If an adapter in the EtherChannel goes down, then traffic is transparently rerouted. Incoming packets are accepted over any of the interfaces available. The switch can choose how to distribute its inbound packets over the EtherChannel according to its own implementation, which in some installations is user-configurable. If all adapters in the channel fail, then the channel is unable to transmit or receive packets.

There are two policies for outbound traffic starting in AIX V5.2: standard and round robin:

- The standard policy is the default; this policy allocates the adapter to use on the basis of the hash of the destination IP addresses.
- The round robin policy allocates a packet to each adapter on a round robin basis in a constant loop.

AIX V5.2 also introduced the concept of configuring a backup adapter to the EtherChannel. The backup adapter's purpose is to take over the IP and MAC address of the channel in the event of a complete channel failure, which is constituted by failure of all adapters defined to the channel. It is only possible to have one backup adapter configured per EtherChannel.

2.4.5 IEEE 802.3ad Link Aggregation

IEEE 802.3ad is a standard way of doing link aggregation. Conceptually, it works the same as EtherChannel in that several Ethernet adapters are aggregated into a single virtual adapter, providing greater bandwidth and improved availability.

For example, ent0 and ent1 can be aggregated into an IEEE 802.3ad Link Aggregation called ent3; interface ent3 would then be configured with an IP address. The system considers these aggregated adapters as one adapter. Therefore, IP is configured over them as over any single Ethernet adapter. The link remains available if one of the underlying physical adapters loses connectivity.

Like EtherChannel, IEEE 802.3ad requires support in the switch. Unlike EtherChannel, however, the switch does not need to be configured manually to know which ports belong to the same aggregation.

Archived

2.4.6 2-Port Adapter-based Ethernet failover

The IBM 2-Port 10/100/1000 Base-TX Ethernet PCI-X Adapter, and the IBM 2-Port Gigabit Ethernet-SX PCI-X Adapter, provide an alternate failover capability. The failover configuration requires both ports to be connected to a switch module. One of the ports is configured as the primary, and the other is configured as the backup. Only the primary port is configured with an IP address. If the primary port loses its connection to the switch module, the backup port will take over seamlessly.

2.4.7 Shared Ethernet failover

The shared Ethernet failover needs two Virtual IO Server (VIOS) logical partitions (LPARs) of an IBM eServer™ p5 (POWER5) or p6 (Power6) node. A shared Ethernet adapter can be configured on both LPARs for the same networks. The shared adapter with higher priority is the primary. The backup is inactive when the primary is up, and it automatically becomes active when the primary fails.

2.5 Storage tools

Built into AIX is the powerful and flexible storage Logical Volume Manager (LVM). In addition, AIX also manages the hardware (devices) to access physical storage, and device drivers to manage data (file system). All these tools have been designed with availability and serviceability in mind. These tools include:

- Hot swap disks
- System backup (mksysb)
- Alternate disk installation
- Network Installation Manager (NIM)
- Logical Volume Manager (LVM)-related options
- Enhanced Journaled File System (JFS)-related options
- AIX storage device driver-related options

These tools are explained in more detail in the following sections.

2.5.1 Hot swap disks

Utilizing the combination of hardware features available in most POWER servers, storage subsystems and AIX, the *hot swap disk* feature provides the ability to replace a failed, or failing, hard drive dynamically without requiring any downtime. AIX provides many utilities such as **mkdev, cfgmgr,** and **rmdev** to support the disk replacement.

While this concept is not new by today's standards, it represents yet another ability that contributes to overall system availability. This ability can be used in conjunction with both PCI hot plug management (see 2.1.9, "PCI hot plug management" on page 15), and hot spare disks (see "Hot spare disks in a volume group" on page 43).

2.5.2 System backup (mksysb)

Although **mksysb** backup is not considered a true "availability" feature by normal standards, the tool is mentioned here as one of the best methods for convenient backup and recovery. Maintaining a backup of your system for recovery is strongly recommended. By using backup

Archived

and recovery tools, you can achieve very fast backup and, more importantly, system recovery of AIX systems. AIX simplifies the backup procedure in the following ways:

- Traditionally, backups have stored data on hard disks. AIX gives you the option of backing up your system to File (NIM server), tape drive, and CD/DVD. Compared to other backup media, CDs are portable, inexpensive, and highly reliable.
- The Network Installation Manager (NIM) server can be used to store and recover any other system.
- You can create a bootable root-volume group backup or user-volume group backup.
- In addition to system recovery, backups can be used to install additional systems with the same image as the system that was originally backed up (called *cloning*).
- You can create a customized installation CD for other machines.
- You can use **mksysb** AIX backups confidently on other IBM AIX-based machines without regard to hardware options.
- You do not have to restore an entire backup. You can list the contents of a system backup and choose to restore only selected files on a running system.

For details about system backup and recovery, refer to the AIX Documentation Web page:

```
http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.install/doc/insgdrf/backup_intro.htm
```

2.5.3 Alternate disk installation

Alternate disk installation and migration allows users a way to update the operating system to the next release, maintenance level, or technology level, without taking the machine down for an extended period of time. Advantages of using alternate disk migration over a conventional migration are reduced downtime, quick recovery from migration failures, and a high degree of flexibility and customization.

This feature also allows the ability to multi-boot from different AIX level images that could also contain different versions or updates to an application. This may be useful for periods of testing, or to even help create a test environment. This feature, originally introduced in AIX 4.1.4.0, utilizes the **alt_disk_inst** command. Starting in AIX V5.3, this feature has now been expanded into three commands:

```
alt_disk_copy
alt_disk_mksysb
alt_rootvg_op
```

These commands now offer additional granular options and greater flexibility than ever before.

Additional details about these commands are available in *AIX 5L Version 5.3 Commands Reference, Volume 1, a-c,* SC23-4888. This publication is also available online at the following site:

```
http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.cmds/doc/aixcmds1/aixcmds1.pdf
```

2.5.4 The multibos utility

Beginning with AIX V5.3 ML3, the **multibos** utility allows the root level administrator to create and maintain two bootable instances of the AIX Base Operating System (BOS) within the same root volume group (rootvg). This utility is provided primarily as an upgrade vehicle.

The **multibos** utility allows the administrator to access, install maintenance, update, and customize the standby instance of BOS (during setup or in subsequent customization

operations) without affecting production on the running instance. Migration to later releases of AIX will be supported when they are available.

The file systems /, /usr, /var, /opt, and /home, along with the boot logical volume, must exist privately in each instance of BOS. The administrator has the ability to share or keep private all other data in the rootvg. As a general rule, shared data should be limited to file systems and logical volumes containing data not affected by an upgrade or modification of private data.

When updating the non-running BOS instance, it is best to first update the running BOS instance with the latest available version of multibos (which is in the bos.rte.bosinst fileset).

Additional details on the **multibos** utility are available in the **man** pages and in *AIX 5L Version 5.3 Commands Reference, Volume 3, a-c,* SC23-4890. This publication is also available online at the following site:

`http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.cmds/doc/aixcmds3/aixcmds3.pdf`

2.5.5 Network Installation Manager (NIM)

The Network Installation Manager (NIM) allows you to centralize installation administration for multiple machines and schedule those installations to minimize disruptions and inconvenience.

- Network Installation Management can be used for centralized installation and software administration of your AIX systems.
- You can choose to install all networked machines at the same time, or stagger those installations.
- Within NIM, you can remain at your console while installing AIX on remote machines. You can even run typical installations unattended.
- You can install each machine with unique options, or install all machines with consistent options.
- You can make a system backup to a NIM server by using the **mksysb** command, and use that backup to install another machine (cloning), as follows:

  ```
  smitty nim
  ```

For details and examples about how to use NIM to enhance your system availability, refer to the IBM Redbooks publication *NIM from A to Z in AIX 5L*, SG24-7296.

2.5.6 Logical Volume Manager-related options

This section discusses Logical Volume Manager (LVM)-related options.

LVM RAID options

LVM supports three software level RAID options:

- RAID0 w/ LVM striping
- RAID1 - LVM mirroring 1:1 (two copies) or 1:1:1 (three copies)
- RAID10 (0+1) - LVM striping plus mirroring

For non-SAN based storage environments, it is quite common to utilize AIX LVM's ability to mirror data. This is especially true for the operating system disks (rootvg). By mirroring rootvg, this allows AIX to continue operating in the event of a rootvg disk failure. This feature,

Archived

when combined with hot spare disk and hot swap disks, allows for maintenance to take place without requiring any planned down time.

However, although LVM mirroring does increase storage availability, it is not intended to be a substitute for system backup. Additional detailed information about LVM mirroring is available in the **mirrorvg** man page and in *AIX 5L Version 5.3 System Management Concepts: Operating System and Devices Management,* SC23-5204.

Hot spare disks in a volume group

Beginning with AIX V5.1, the ability to designate hot spare disks for an LVM mirrored volume group was added. LVM hot spare allows automatic migration of partitions from a failing disk to another free disk previously assigned as a hot spare. The hot spare disk feature is an operating system equivalent to that of a hot spare disk when using a RAID storage solution that most storage administrators are already familiar. Hot spare disk concepts and policies are described in *AIX 5L Version 5.3 System Management Concepts: Operating System and Devices Management,* SC23-5204.

The chvg feature for sync

There is a feature in LVM to set the synchronization characteristics for the volume group specified by the VolumeGroup parameter that either permits (`y`) the automatic synchronization of stale partitions or prohibits **(n)** the automatic synchronization of stale partitions. This flag has no meaning for non-mirrored logical volumes.

Automatic synchronization is a recovery mechanism that will only be attempted after the LVM device driver logs LVM_SA_STALEPP in the errpt. A partition that becomes stale through any other path (for example, mklvcopy) will not be automatically resynced. This is always used in conjunction with the hot spare disk in a volume group feature.

To change the volume group characteristics, you can use the **smitty chvg** SMIT fastpath, or you can use the following commands (these are examples):

```
/usr/sbin/chvg -s'y' fransvg
/usr/sbin/chvg -h hotsparepolicy -s syncpolicy volumegroup
```

Advanced RAID support

Most common storage subsystems today are RAID arrays defined into multiple logical units (LUNs) that ultimately represent a disk definition to AIX. When additional space is required, you can choose to expand the size of a current LUN, as opposed to adding an additional LUN. This ultimately results in changing the size of a previously-defined disk.

In order to use this space, the disk must grow in size by dynamically adding additional physical partitions (PP). AIX V5.2 introduced support of dynamic volume expansion by updating the **chvg** command to include a new **-g** flag. More detailed information about this topic is available in the man page for **chvg** and in *AIX 5L Differences Guide Version 5.2 Edition,* SG24-5765.

Portability of volume groups

One useful attribute of LVM is a user's ability to take a disk or sets of disks that make up a volume group to another AIX system and introduce the information created on the first machine onto the second machine. This ability is provided through the Volume Group Descriptor Area (VGDA) and the logical volume control block (LVCB).

The design of LVM also allows for accidental duplication of volume group and logical volume names. If the volume group or logical volume names being imported already exist on the new machine, then LVM will generate a distinct volume group or logical volume name.

Quorum

A *quorum* is a vote of the number of Volume Group Descriptor Areas and Volume Group Status Areas (VGDA/VGSA) that are active. A quorum ensures data integrity of the VGDA/VGSA areas in the event of a disk failure. Each physical disk in a volume group has at least one VGDA/VGSA.

When a volume group is created onto a single disk, it initially has two VGDA/VGSA areas residing on the disk. If a volume group consists of two disks, one disk still has two VGDA/VGSA areas, but the other disk has one VGDA/VGSA. When the volume group is made up of three or more disks, then each disk is allocated just one VGDA/VGSA.

A quorum is lost when enough disks and their VGDA/VGSA areas are unreachable such that 51% (a majority) of VGDA/VGSA areas no longer exists. In a two-disk volume group, if the disk with only one VGDA/VGSA is lost, a quorum still exists because two of the three VGDA/VGSA areas still are reachable. If the disk with two VGDA/VGSA areas is lost, this statement is no longer true. The more disks that make up a volume group, the lower the chances of quorum being lost when one disk fails.

When a quorum is lost, the volume group varies itself off so that the disks are no longer accessible by the Logical Volume Manager (LVM). This prevents further disk I/O to that volume group so that data is not lost or assumed to be written when physical problems occur. Additionally, as a result of the vary off, the user is notified in the error log that a hardware error has occurred and service must be performed.

There are cases when it is desirable to continue operating the volume group even though a quorum is lost. In these cases, quorum checking may be turned off for the volume group. This type of volume group is referred to as a *nonquorum volume group*. The most common case for a nonquorum volume group is when the logical volumes have been mirrored.

When a disk is lost, the data is not lost if a copy of the logical volume resides on a disk that is not disabled and can be accessed. However, there can be instances in nonquorum volume groups, mirrored or nonmirrored, when the data (including copies) resides on the disk or disks that have become unavailable. In those instances, the data may not be accessible even though the volume group continues to be varied on.

Online quorum change

Starting in AIX 5.3 TL7, quorum changes are allowed online without having to varyoff and varyon the volume group. This also means no reboot is required when changing quorum for rootvg.

Scalable Volume Group

Scalable Volume Group (SVG) support was added in AIX 5.3 that will allow increasing the number of logical volumes and physical partitions online without requiring additional free partitions for metadata expansion. The SVG support also removed the 1016 PP per PV limitation that required factor changes when trying to extend a VG with a disk that has more than 1016 partitions. Refer to the mkvg (-S option) man page for more information.

2.5.7 Geographic Logical Volume Manager

The Geographic Logical Volume Manager (GLVM), added into AIX V5.3 ML3 in September 2005, is an AIX LVM extension for real time geographic data mirroring over standard TCP/IP networks. GLVM can help protect your business from a disaster by mirroring your mission-critical data to a remote disaster recovery site.

GLVM is built upon AIX's Logical Volume Manager, which allows you to create up to three total copies of data between two geographically distant locations. Because of its tight integration with LVM, users who are already familiar with LVM should find GLVM very easy to learn. Because it is operating system-based, this creates additional flexibility by allowing GLVM to be independent of the storage subsystems.

GLVM is synchronous data replication of physical volumes to *remote physical volumes*. The combination of both remote physical volumes with local physical volumes form *geographically mirrored volume groups*. These are managed by LVM very much like ordinary volume groups. A high level example of a GLVM environment is shown in Figure 2-1.

Figure 2-1 GLVM concepts example

Additional GLVM details, including installing and configuring, are available in the white paper *Using the Geographic LVM in AIX 5L,* which can be found online at the following site:

`http://www-03.ibm.com/systems/p/os/aix/whitepapers/pdf/aix_glvm.pdf`

GLVM by itself does not provide automated recovery in the event of a catastrophic site failure. However, it is a fully integrated and supported option in IBM's premier disaster recovery software solution for System p systems running AIX called HACMP/XD. More information about HACMP/XD options, including GLVM, is available in *Implementing High Availability Cluster Multi-Processing (HACMP) Cookbook*, SG24-6769.

Although GLVM does not have technical distance limitations for support, there are several factors which dictate what is realistically achievable. These factors include, but are not limited to, network bandwidth, I/O rates, and latency between the sites. More detailed information about this topic is available in the white paper *Optimizing Mirroring Performance using HACMP/XD for Geographic LVM*, which can be found online at the following site:

`http://www-304.ibm.com/jct03004c/systems/p/software/whitepapers/hacmp_xd_glvm.pdf`

Archived

2.5.8 Journaled File System-related options

AIX provides native support for Journaled File System (JFS) and JFS2. These file systems have built-in characteristics to maintain data availability and integrity. JFS and JFS2 work in conjunction with AIX LVM, as described in this section.

Enhanced Journaled File System (JFS2)

Enhanced Journaled File System (JFS2) supports the entire set of file system semantics. The file system uses database journaling techniques to maintain its structural consistency. This prevents damage to the file system when the file system is halted abnormally.

Each JFS2 resides on a separate logical volume. The operating system mounts JFS2 during initialization. This multiple file system configuration is useful for system management functions such as backup, restore, and repair. It isolates a part of the file tree to allow system administrators to work on a particular part of the file tree.

JFS2 file system shrink

Dynamically adding additional space to an existing filesystem can be easily done. However, *reducing* a filesystem often involved creating another smaller filesystem, copying the data over, and then removing the original filesystem.

AIX V5.3 introduced the ability to shrink a JFS2 filesystem dynamically by allowing the **`chfs`** command to recognize **`a -`** in the size attribute to be interpreted as a request to reduce the filesystem by the amount specified. More detailed information about shrinking a filesystem is available in *AIX 5L Differences Guide Version 5.3 Edition,* SG24-7463.

2.5.9 AIX storage device driver-related options

Today’s storage subsystems provide built-in performance and data protection. However, they must be matched with operating system drivers to fully exploit their characteristics.

Multipath I/O (MPIO)

Multipath I/O (MPIO) was a new feature introduced in AIX V5.2 that allows access to a single disk device (LUN) from multiple adapters along different (storage) paths. There are three main reasons to utilize MPIO:

- Improved performance
- Improved reliability and availability
- Easier device administration

MPIO supports standard AIX commands to be used to administer the MPIO devices.

A path control module (PCM) provides the path management functions. An MPIO-capable device driver can control more than one type of target device. A PCM can support one or more specific devices. Therefore, one device driver can be interfaced to multiple PCMs that control the I/O across the paths to each of the target devices.

The AIX PCM has a health-check capability that can be used for the following tasks:

- Check the paths and determine which paths are currently usable for sending I/O.
- Enable a path that was previously marked failed because of a temporary path fault (for example, when a cable to a device was removed and then reconnected).
- Check currently unused paths that would be used if a failover occurred (for example, when the algorithm attribute value is failover, the health check can test the alternate paths).

Archived

However, not all disk devices can be detected and configured by using the AIX default PCMs. The AIX default PCMs consist of two path control modules, one to manage disk devices and another to mange tape devices. If your device is not detected, check with the device vendor to determine if a PCM is available for your device.

AIX MPIO supports multiple IO routing policies, thereby increasing the system administrator's control over MPIO reliability and performance. Detailed information on MPIO including path management is available in *AIX 5L System Management Guide: Operating System and Devices Management,* SC23-5204, which is also online at the following site:

http://publib.boulder.ibm.com/infocenter/pseries/v5r3/topic/com.ibm.aix.baseadmn/doc/baseadmndita/baseadmndita.pdf

Dynamic tracking of fibre channel devices

Dynamic tracking was introduced in AIX V5.2 ML1. Prior to its introduction, performing SAN fabric changes such as moving cables between ports, adding interswitch links, and anything that caused N_Port ID to change, involved a service disruption. Often this would include unmounting filesystems, varying off volume groups, and even removing the device definition.

Dynamic tracking allows such changes to be performed without bringing the devices offline. Although this support is an integral component in contributing to overall system availability, devices that are only accessible by one path can still be affected during these changes. Applied logic dictates that a single path environment does not provide maximum availability.

Dynamic tracking of fibre channel devices is controlled by a new fscsi device attribute, *dyntrk*. The default setting for this attribute is no. To enable this feature, the fscsi attribute must be set to `yes`, as shown in Example 2-9.

Example 2-9 Enabling dynamic tracking

```
lizray /# 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      0x10500      Adapter SCSI ID                       False
sw_fc_class  3            FC Class for Fabric                   True

lizray /# chdev -l fscsi0 -a dyntrk=yes

lizray /# lsattr -El fscsi0
attach       switch       How this adapter is CONNECTED         False
dyntrk       yes          Dynamic Tracking of FC Devices        True
fc_err_recov fast_fail    FC Fabric Event Error RECOVERY Policy True
scsi_id      0x10500      Adapter SCSI ID                       False
sw_fc_class  3            FC Class for Fabric
True
```

Important: If child devices exist and are in the `available` state, this command will fail. These devices must either be removed or put in the `defined` state for successful execution.

Fast I/O failure for Fibre Channel devices

This feature allows the user to indicate that I/Os down a particular link be failed faster than they are currently. This may be useful in a multipath environment where customers want I/Os to fail over to another path relatively quickly.

Fast I/O failure is controlled by a new fscsi device attribute, *fc_err_recov*. The default setting for this attribute is *delayed_fail,* which is the I/O failure behavior that has existed in previous versions of AIX. To enable this feature, the fscsi attribute must be set to *fast_fail* utilizing the **chdev** command, as shown in Example 2-10:

Example 2-10 Enabling fast I/O failure

```
valkim /# 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      0x10500      Adapter SCSI ID                      False
sw_fc_class  3            FC Class for Fabric                  True

valkim /# chdev -l fscsi0 -a fc_err_recov=fast_fail

valkim /# lsattr -El fscsi0
attach       switch       How this adapter is CONNECTED        False
dyntrk       no           Dynamic Tracking of FC Devices       True
fc_err_recov fast_fail    FC Fabric Event Error RECOVERY Policy True
scsi_id      0x10500      Adapter SCSI ID                      False
sw_fc_class  3            FC Class for Fabric                  True
```

Important: If child devices exist and are in the `available` state, this command will fail. These devices must either be removed or put in the `defined` state for successful execution.

In single-path configurations, especially configurations with a single-path to a paging device, the default *delayed_fail* setting is the recommended setting.

In addition, dynamic tracking is often used in conjunction with fast I/O fail. Additional information about requirements and restrictions of fast I/O failure for fibre channel devices is available in *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts,* SC23-4900.

2.6 System and performance monitoring and tuning

Tools are available for monitoring and tuning the system to provide better performance and increased availability and resiliency, as described in the following sections.

2.6.1 Electronic Service Agent

Electronic Service Agent™ is a no-charge software tool that resides on your system to continuously monitor events and periodically send service information to IBM support on a user-definable timetable. This information may assist IBM support in diagnosing problems.

This tool tracks and captures service information, hardware error logs, and performance information. It automatically reports hardware error information to IBM support as long as the system is under an IBM maintenance agreement or within the IBM warranty period. Service information and performance information reporting do not require an IBM maintenance agreement or do not need to be within the IBM warranty period to be reported.

Previous Electronic Service Agent products were unique to the platform or operating system on which they were designed to run. Because the Electronic Service Agent products were

Archived

unique, each offered its own interface to manage and control the Electronic Service Agent and its functions. Since networks can have different platforms with different operating systems, administrators had to learn a different interface for each different platform and operating system in their network. Multiple interfaces added to the burden of administering the network and reporting problem service information to IBM support.

In contrast, Electronic Service Agent 6.1 installs on platforms running different operating systems. ESA 6.1 offers a consistent interface to reduce the burden of administering a network with different platforms and operating systems. Your network can have some clients running the Electronic Service Agent 6.1 product and other clients running the previous Electronic Service Agent product.

If you have a mixed network of clients running Electronic Service Agent 6.1 and previous Electronic Service Agent products, you need to refer to the information specific to each Electronic Service Agent product for instructions on installing and administering that product.

To access Electronic Service Agent user guides, go to the Electronic Services Web site and select **Electronic Service Agent** from the left navigation. In the contents pane, select **Reference Guides** > **Select a platform** > **Select an Operating System or Software**.

Alternatively, you can use the following SMIT fastpath:

```
smitty esa_main
```

Further information is available at the following site:

http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicbd/eicbd_aix.pdf

2.6.2 Other tools for monitoring a system

There are also other tools that are useful for system monitoring; here are a few examples:

- **vmstat** - Overall system statistics
- **netstat** - Network statistics
- **no** - Network tuning
- **sar** - Overall system statistics
- **iostat** - Disk and CPU statistics (needs to be enabled to collect statistics)
- **lsconf** - List and document the machine
- **filemon** - Find the busy filesystems and files
- **fileplace** - Check for scrambled files
- **lparstat** - Check on shared processor LPARs
- **perfpmr** - Report performance issues
- **lvmstat** - Check high-use disks
- **ioo** - Configures I/O tuning parameters
- **tuncheck** - Validates a tunable file with **tunchange**, **tundefault,tunrestore**, and **tunsave** commands

You can write shell scripts to perform data reduction on the command output, warn of performance problems, or record data on the status of a system when a problem is occurring. For example, a shell script can test the CPU idle percentage for zero (0), a saturated condition, and execute another shell script for when the CPU-saturated condition occurred.

2.6.3 The topas command

The **topas** command reports vital statistics about the activity on the local system, such as real memory size and the number of write system calls. This command uses the curses library to

display its output in a format suitable for viewing on an 80x25 character-based display, or in a window of at least the same size on a graphical display.

The **topas** command extracts and displays statistics from the system with a default interval of two seconds. The command offers the following alternate screens:

- Overall system statistics
- List of busiest processes
- WLM statistics
- List of hot physical disks
- Logical partition display
- Cross-Partition View (AIX V5.3 ML3 and higher)

SMIT panels are available for easier configuration and setup of the topas recording function and report generation; use this command:

```
smitty topas
```

> **Important:** Keep in mind that the incorrect use of commands to change or tune the AIX kernel can cause performance degradation or operating system failure.
>
> Before modifying any tunable parameter, you should first carefully read about all of the parameter's characteristics in the Tunable Parameters section of the product documentation in order to fully understand the parameter's purpose.
>
> Then ensure that the Diagnosis and Tuning sections for this parameter actually apply to your situation, and that changing the value of this parameter could help improve the performance of your system. If the Diagnosis and Tuning sections both contain only N/A, it is recommended that you do not change the parameter unless you are specifically directed to do so by IBM Software Support.

2.6.4 Dynamic kernel tuning

AIX offers dynamic kernel tuning without system reboots as one of its many standard features for excellent single-system availability. AIX provides the following significant dynamically tunable kernel parameters without system reboots:

- Scheduler and memory load control parameters
- Virtual Memory Manager, File System and Logical Volume Manager parameters
- Network option parameters
- NFS option parameters
- Input/Output parameters
- Reliability, Accessibility and Serviceability parameters

All six tuning commands (**schedo, vmo, no, nfso, ioo,** and **raso**) use a common syntax and are available to directly manipulate the tunable parameter values. SMIT panels and Web-based System Manager plug-ins are also available. These all provide options for displaying, modifying, saving, and resetting current and next boot values for all the kernel tuning parameters. To start the SMIT panels that manage AIX kernel tuning parameters, use the SMIT fast path **smitty tuning**.

You can make permanent kernel-tuning changes without having to edit any rc files. This is achieved by centralizing the reboot values for all tunable parameters in the /etc/tunables/nextboot stanza file. When a system is rebooted, the values in the /etc/tunables/nextboot file are automatically applied. For more information, refer to *IBM*

eServer Certification Study Guide - AIX 5L™ Performance and System Tuning, SG24-6184, which is available at the following site:

http://www.redbooks.ibm.com/redbooks/pdfs/sg246184.pdf

The raso command

The **raso** command is used to configure “selected” RAS tuning parameters. This command sets or displays the current or next-boot values to configure selected tuning parameters for the RAS tuning parameters it supports. The command can also be used to make permanent changes, or to defer changes until the next reboot.

The specified flag determines whether the **raso** command sets or displays a parameter. The **-o** flag can be used to display the current value of a parameter, or to set a new value for a parameter.

Here, we show the command syntax for the **raso** command:

```
Command Syntax
raso [ -p | -r ] [ -o Tunable [ = Newvalue ] ]
raso [ -p | -r ] [ -d Tunable ]
raso [ -p ] [ -r ] -D
raso [ -p ] [ -r ] [-F]-a
raso -h [ Tunable ]
raso [-F] -L [ Tunable ]
raso [-F] -x [ Tunable ]
```

> **Note:** Multiple **-o**, **-d**, **-x**, and **-L** flags can be specified.

As with all AIX tuning parameters, changing a raso parameter may impact the performance or reliability of your AIX LPAR or server; refer to *IBM System p5 Approaches to 24x7 Availability Including AIX 5L*, for more information about this topic, which is available at the following site:

http://www.redbooks.ibm.com/redbooks/pdfs/sg247196.pdf

We recommend that you do not change the parameter unless you are specifically directed to do so by IBM Software Support.

2.7 Security

The security features in AIX also contribute to system availability.

Role-Based Access Control

Role-Based Access Control (RBAC) improves security and manageability by allowing administrators to grant authorization for the management of specific AIX resources to users other than root by associating those resources with a role that is then associated with a particular system user. Role-Based Access Control can also be used to associate specific management privileges with programs, which can reduce the need to run those programs under the root user or via setuid.

AIX Security Expert LDAP integration

The AIX Security Expert provides clients with the capability to manage more than 300 system security settings from a single interface. The AIX Security Expert has been enhanced in AIX V6.1 with an option to store security templates directly in a Lightweight Directory Protocol (LDAP) directory, thus simplifying implementation of a consistent security policy across an entire enterprise.

For more detailed information about security features in AIX, refer to *AIX V6 Advanced Security Features Introduction and Configuration*, SG24-7430, which is available at the following site:

http://www.redbooks.ibm.com/redbooks/pdfs/sg247430.pdf

2.8 AIX mobility features

This section introduces the new AIX mobility features and explains their basic requirements.

- Live partition mobility
- Live application mobility

2.8.1 Live partition mobility

Live partition mobility allows you to move a logical partition from one physical server to another, with no application downtime. Partition mobility is a key continuous availability feature for enhancing Advanced Power Virtualization, and it will help you meet common business needs by keeping applications up and running.

Partition mobility basic requirements

Partition mobility requires at least two POWER6 Systems managed by the same Hardware Management Console (HMC). Both systems must have the Advanced Power Virtualization feature activated, and both systems need to be connected to the same HMC and private network. All logical partitions must be on the same open network with Resource Monitoring and Control (RMC) established to the HMC.

> **Note:** The partition to be moved must use *only* virtual devices. The virtual disks must be LUNs on external SAN storage. The LUNs must be accessible to the VIO Server on each system.

Partition mobility usage examples

Partition mobility can be used for evacuating a system before performing scheduled system maintenance. It is also useful for moving workloads across a pool of different resources as business needs shift, and for removing workloads from underutilized machines so that they can powered off to save energy costs.

For more information about this topic, refer to the IBM Redbooks publication *IBM System p Live Partition Mobility*, SG24-7460.

2.8.2 Live application mobility

Live application mobility is enabled by the IBM Workload Partitions Manager™ (WPAR Manager) for AIX product. WPAR Manager provides functionalities that make it possible to move a workload partition from one server to another. A workload partition (WPAR) is a software-created, virtualized operating system environment within a single AIX V6.1 image. Each workload partition is a secure and isolated environment for the application it hosts. To the application in a workload partition, it appears it is being executed in its own dedicated AIX instance.

Archived

The virtualization introduced with the workload partition (WPAR) allows WPAR *checkpoint* and WPAR *resume* functionalities. The functionality of checkpoint and resume of a workload partition on another system is called live application mobility, as explained here:

- Checkpoint consists of taking a real-time snapshot of the WPAR content and dumping it to disk. This file on disk is called a *statefile*. The snapshot will interrupt all processes to reach a quiescence point to allow the scan and capture of all resources in memory into the statefile.
- Resume consists of recreating a WPAR and reloading its content in memory on another system from the previous statefile, and then resuming its processing. From the user perspective, it is as if the application just paused for a moment.

Live application mobility requirements

> **Note:** Live application mobility is a completely separate technology from live partition mobility. However, they can coexist on a system that can match the prerequisites of both.

For live partition mobility, each participating logical partition and machine must be configured the same way for the workload partition. This includes:

- The same file systems needed by the application via NFS V3 or NFS V4.
- Similar network functionalities (usually the same subnet, with routing implications).
- Enough space to handle the data created during the relocation process. (Estimating the size is very difficult since it would include the virtual application size, some socket information, and possible file descriptor information.)
- The same operating system level and technology maintenance level (TL).

  > **Note:** In addition to operating system requirements, if not on shared file systems (NFS), then application binaries *must* be at the same level (identical).

For more information about workload partitions and live application mobility, refer to the IBM Redbooks publication *Introduction to Workload Partition Management in IBM AIX Version 6*, SG24-7431.

Archived

Archived

3

AIX advanced continuous availability tools and features

This chapter details selected advanced continuous availability tools and features on AIX. Some of these tools and features are new, announced with AIX V6.1. Others were recently introduced on AIX. And still others have been available since early AIX V5.3 technology levels.

The following tools and features are discussed here:

- RAS component hierarchy
- Lightweight memory trace (LMT)
- Snap
- Minidump and new dump facilities
- Concurrent updates
- POSIX trace
- Storage keys
- xmalloc debug enhancement
- ProbeVue
- Functional recovery routines (which handle kernel issues without bringing down the system)

© Copyright IBM Corp. 2008. All rights reserved.

3.1 AIX Reliability, Availability, and Serviceability component hierarchy

The advanced continuous availability tools and features can be assimilated as Reliability, Availability, and Serviceability (RAS) tools or facilities. These facilities were developed to enhance AIX capabilities for investigating application, component or subsystem failures and problems—without needing to perform a complete system dump or stop the application or system.

A RAS component hierarchy is used by some features. This divides the system into a resource hierarchy, and allows individual RAS commands to be directed to very specific parts of the system. The RAS features that exploit the RAS component hierarchy are runtime checking, component trace, and component dump. This grouping hierarchy is illustrated in Figure 3-1.

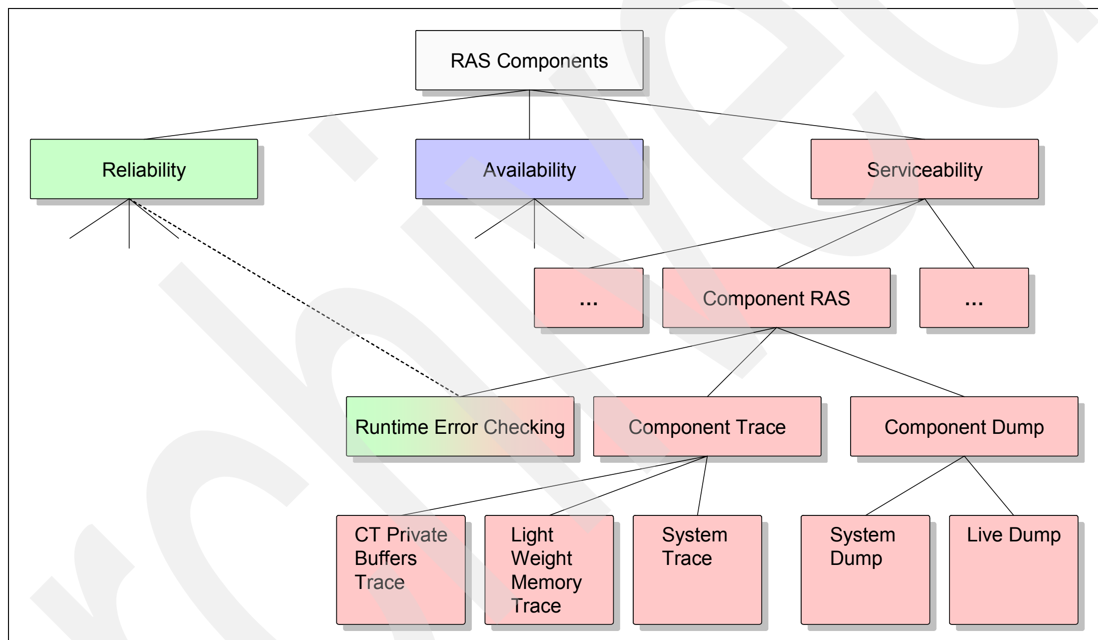


Figure 3-1 Component Reliability, Availability, and Serviceability

3.1.1 First Failure Data Capture feature

Lightweight memory trace is used to provide information for First Failure Data Capture (FFDC). It was introduced in AIX Version V5.3 ML3. The set of First Failure Data Capture features was further expanded in AIX V5.3 TL5 and AIX 6.1. They include:

- Lightweight memory trace (LMT)
- Run-time error checking (RTEC)
- Component Trace (CT)
- Live Dump

These features are enabled by default at levels that provide valuable First Failure Data Capture information with minimal performance impact. To enable or disable all four advanced First Failure Data Capture features, enter the following command:

```
/usr/lib/ras/ffdcctrl  -o ffdc=enabled -o bosboot=no
```

To access the SMIT FFDC menu, use the fastpath **smitty ffdc**. The menu is shown in Example 3-1.

Example 3-1 Advanced FFDC features

```
                        Advanced First Failure Data Capture Features

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

                                                        [Entry Fields]
  Advanced First Failure Data Capture Features       [enabled]          +
  Run bosboot automatically                          [no]               +




F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

For more information about FFDC, refer to *IBM eServer p5 590 and 595 System Handbook*, SG24-9119.

3.2 Lightweight memory trace

This section describes how to use lightweight memory trace (LMT). Lightweight memory trace is a serviceability feature. It is an important first failure data capture tool, and is most useful to those with AIX source-code access or a deep understanding of AIX internals.

3.2.1 LMT implementation

As mentioned, Lightweight memory trace (LMT) provides trace information for First Failure Data Capture (FFDC). It is a constant kernel trace mechanism that records software events occurring during system life. The system activates LMT at initialization, and then it runs continuously. Recorded events are saved to per-processor memory trace buffers.

There are two memory trace buffers for each processor: one to record common events, and one to record rare events. These buffers can be extracted from system dumps or accessed on a live system by service personnel using the mtrcsave command. The extracted memory trace buffers can be viewed by using the **trcrpt** command, with formatting as defined in the /etc/trcfmt file.

LMT has been carefully implemented such that it has negligible performance impacts. The impact on the throughput of a kernel-intensive benchmark is just 1%, and is much less for typical user workloads. LMT requires the consumption of a small amount of pinned kernel memory. The default amount of memory required for the trace buffers is calculated based on factors that influence trace record retention.

Lightweight memory trace differs from traditional AIX system trace in several ways. First, it is more efficient. Second, it is enabled by default, and has been explicitly tuned as a First Failure Data Capture mechanism. Unlike traditional AIX system trace, you cannot selectively record only certain AIX trace hook ids with LMT. With LMT, you either record *all* LMT-enabled hooks, or you record *none*.

Archived

This means that traditional AIX system trace is the preferred Second Failure Data Capture (SFDC) tool, because you can more precisely specify the exact trace hooks of interest, given knowledge gained from the initial failure.

Traditional AIX system trace also provides options that allow you to automatically write the trace information to a disk-based file (such as /var/adm/ras/trcfile). Lightweight memory trace provides no such option to automatically write the trace entries to disk when the memory trace buffer fills. When an LMT memory trace buffer fills, it “wraps”, meaning that the oldest trace record is overwritten.

The value of LMT derives from being able to view some history of what the system was doing prior to reaching the point where a failure is detected. As previously mentioned, each CPU has a memory trace buffer for common events, and a smaller memory trace buffer for rare events.

The intent is for the “common” buffer to have a 1- to 2-second retention (in other words, have enough space to record events occurring during the last 1 to 2 seconds, without wrapping). The “‘rare” buffer should have at least an hour's retention. This depends on workload, and on where developers place trace hook calls in the AIX kernel source and which parameters they trace.

Disabling and enabling lightweight memory trace

You can disable lightweight memory trace by using the following command:

```
/usr/bin/raso -r -o mtrc_enabled=0
```

You can enable lightweight memory trace by using the following command:

```
/usr/bin/raso -r -o mtrc_enabled=1
```

> **Note:** In either case, the boot image must be rebuilt (the **bosboot** command needs to be run), and the change does not take effect until the next reboot.

Lightweight Memory Trace memory consumption

The default amount of memory required for the memory trace buffers is automatically calculated based on factors that influence software trace record retention, with the target being sufficiently large buffers to meet the retention goals previously described.

There are several factors that may reduce the amount of memory automatically used. The behavior differs slightly between the 32-bit (unix_mp) and 64-bit (unix_64) kernels. For the 64-bit kernel, the default calculation is limited such that no more than 1/128 of system memory can be used by LMT, and no more than 256 MB by a single processor.

The 32-bit kernel uses the same default memory buffer size calculations, but further restricts the total memory allocated for LMT (all processors combined) to 16 MB. Table 3-1 presents some example LMT memory consumption.

Table 3-1 Lightweight Memory Trace memory consumption

Machine	Number of CPUs	System memory	Total LMT memory: 64-bit kernel	Total LMT memory: 32-bit kernel
POWER3™ (375 MHz CPU)	1	1 GB	8 MB	8 MB
POWER3 (375 MHz CPU)	2	4 GB	16 MB	16 MB

Machine	Number of CPUs	System memory	Total LMT memory: 64-bit kernel	Total LMT memory: 32-bit kernel
POWER5 (1656 MHz CPU, shared processor LPAR, 60% ent cap, simultaneous multi-threading	8 logical	16 GB	120 MB	16 MB
POWER5 (1656 MHz CPU)	16	64 GB	512 MB	16 MB

To determine the total amount of memory (in bytes) being used by LMT, enter the following shell command:

```
echo mtrc | kdb | grep mt_total_memory
```

The 64-bit kernel resizes the LMT trace buffers in response to dynamic reconfiguration events (for POWER4 and above systems). The 32-bit kernel does not resize; it will continue to use the buffer sizes calculated during system initialization.

Note: For either kernel, in the rare case that there is insufficient pinned memory to allocate an LMT buffer when a CPU is being added, the CPU allocation will fail. This can be identified by a CPU_ALLOC_ABORTED entry in the AIX error log, with detailed data showing an Abort Cause of 0000 0008 (LMT) and Abort Data of 0000 0000 0000 000C (ENOMEM).

Changing memory buffer sizes

For the 64-bit kernel, you can also use the **/usr/sbin/raso** command to increase or decrease the memory trace buffer sizes. This is done by changing the mtrc_commonbufsize and mtrc_rarebufsize tunable variables. These two variables are dynamic parameters, which means they can be changed without requiring a reboot.

For example, to change the per cpu rare buffer size to sixteen 4 K pages, for this boot as well as future boots, you would enter:

```
raso -p -o mtrc_rarebufsize=16
```

For more information about the memory trace buffer size tunables, refer to **raso** command documentation.

Note: Internally, LMT tracing is temporarily suspended during any 64-bit kernel buffer resize operation.

For the 32-bit kernel, the options are limited to accepting the default (automatically calculated) buffer sizes, or disabling LMT (to completely avoid buffer allocation).

Using lightweight memory trace

This section describes various commands which are available to make use of the information captured by lightweight memory trace. However, keep in mind that LMT is designed to be

used by IBM service personnel, so these commands (or their new LMT-related parameters) may not be documented in the standard AIX documentation. So, to help you recall specific syntax, each command displays a usage string if you enter <**command -?**>. This section provides an overview of the commands used to work with LMT.

The LMT memory trace buffers are included in an AIX system dump. You can manipulate them similarly to traditional AIX system trace buffers. The easiest method is to use the **trcdead** command to extract the LMT buffers from the dump. The new **-M** parameter extracts the buffers into files in the LMT log directory. The default LMT log directory is /var/adm/ras/mtrcdir.

If the dump has compression set (on), the dumpfile file needs to be uncompressed with the **dmpuncompress** command before running **kdb**, as shown here:

```
dmpuncompress dump_file_copy.BZ
```

For example, to extract LMT buffers from a dump image called dump_file_copy, you can use:

```
trcdead -M dump_file_copy
```

Each buffer is extracted into a unique file, with a control file for each buffer type. This is similar to the per CPU trace option in traditional AIX system trace. As an example, executing the previous command on a dump of a two-processor system would result in the creation of the following files:

```
ls /var/adm/ras/mtrcdir
   mtrccommon     mtrccommon-1   mtrcrare-0
   mtrccommon-0   mtrcrare       mtrcrare-1
```

The new **-M** parameter of the **trcrpt** command can then be used to format the contents of the extracted files. The trcrpt command allows you to look at common file and rare files separately. Beginning with AIX V5.3 TL5, both common and rare files can be merged together chronologically.

Continuing the previous example, to view the LMT files that were extracted from the dumpfile, you can use:

```
trcrpt -M common
```

and

```
trcrpt -M rare
```

These commands produce large amounts of output, so we recommend that you use the **-o** option to save the output to a file.

Other **trcrpt** parameters can be used in conjunction with the **-M** flag to qualify the displayed contents. As one example, you could use the following command to display only VMM trace event group hook ids that occurred on CPU 1:

```
trcrpt -D vmm -C 1 -M common
```

Example 3-2 shows the output of the **trcrpt** command displaying a VMM trace:

Example 3-2 Output of trcrpt command

```
# trcrpt -D vmm -C 1 -M common
ID   CPU    ELAPSED_SEC     DELTA_MSEC   APPL    SYSCALL KERNEL  INTERRUPT

1BE  1    141.427476136  141427.476136                  VMM pfendout:    V.S=0020.31250 ppage=32B20
                                                        client_segment commit in progress privatt

P_DEFAULT 4K
```

```
large modlist req (type 0)error=0000
```

Using the **trcrpt** command is the easiest and most flexible way to view lightweight memory trace records.

3.3 The snap command

The **snap** command gathers AIX system configuration information, error log data, kernel information and a system dump, if one exists. The information is compressed into a **pax file**. The file may then be written to a device such as tape or DVD, or transmitted to a remote system. The information gathered with the **snap** command might be required to identify and resolve system problems. Directory /tmp/ibmsupt contains **snap** command output.

> **Note:** The data collected with the **snap** command is designed to be used by IBM service personnel.

Snap usage examples

To gather all system configuration information, use:

```
snap -a
```

The output of this command is written to the /tmp/ibmsupt directory.

To create a pax image of all files contained in the /tmp/ibmsupt directory, use:

```
snap -c
```

To remove **snap** command output from the /tmp/ibmsupt directory, use:

```
snap -r
```

We recommend that you run the **snap** commands in the following order:

1. Remove old **snap** command output from /tmp/ibmsupt:
   `snap -r`
2. Gather all new system configuration information:
   `snap -a`
3. Create pax images of all files contained in the /tmp/ibmsupt directory:
   `snap -c`

The **snap** command has several flags that you can use to gather relevant system information. For examples, refer to the snap man pages:

```
man snap
```

3.4 Minidump facility

A system dump may not always complete successfully for various reasons. If a dump is not collected at system crash time, it is often difficult to determine the cause of the crash. To combat this, the minidump facility has been introduced in AIX V5.3 ML3.

A *minidump* is a small, compressed dump that is stored to NVRAM when the system crashes or a dump is initiated. It is written to the error log on reboot. The minidump can be used to observe the system state and perform some debugging if a full dump is not available. It can

Archived

also be used to obtain a snapshot of a crash, without having to transfer the entire dump from the crashed system.

Minidumps will show up as error log entries with a label of MINIDUMP_LOG and a description of COMPRESSED MINIMAL DUMP.

To view only the minidump entries in the error log, enter this command (see also Example 3-3):

```
errpt –J MINIDUMP_LOG
```

3.4.1 Minidump formatter

A new command **mdmprpt** provides functionalities to format minidumps. This command has the following usage:

```
mdmprpt [-i logfile] [-l seqno]  [-r]
```

When given a sequence number (seqno), the **mdmprpt** command will format the minidump with the given sequence number. Otherwise, it will format the most recent minidump in the error log, if any. It reads the corresponding minidump entry from the error log, and uncompresses the detail data. It then formats the data into a human-readable output which is sent to stdout.

The formatted output will start with the symptom string and other header information. Next it will display the last error log entry, the dump error information if any, and then each CPU, with an integrated stack (the local variables for each frame, if any, interleaved with the function name+offset output).

Note the following points:

- A different error log file can be supplied by using the **–i** flag.
- A different minidump from the most recent one can be displayed using the **–l** flag, followed by the sequence number of the error log entry where the minidump is.
- The **–r** flag is used to display the raw data that was saved to NVRAM and then extracted to the error log, without decompression or formatting. In general, this is only useful for debugging problems with minidump itself.

The **mdmprpt** command will use the uncompressed header of the minidump to correctly decompress and format the remainder. In the case of a user-initiated dump, the data gathered will be spread evenly across all CPUs, because the failure point (if any) is unknown.

The size of NVRAM is extremely limited, so the more CPUs on the system, the less data that will be gathered per CPU. In the case of a dump after a crash, the CPU that crashed will use up most of the space, and any that remains will be split among the remaining CPUs.

> **Note:** The **mdmprpt** command requires a *raw* error log file; it cannot handle output from the **errpt** command.
>
> Example 3-3 on page 62 displays partial output from a minidump. The **–g** flag will tell **snap** to gather the error log file.

Example 3-3 A minidump - partial output

```
# errpt -J MINIDUMP_LOG
IDENTIFIER TIMESTAMP  T C RESOURCE_NAME  DESCRIPTION
F48137AC   1002144807 U O minidump       COMPRESSED MINIMAL DUMP
```

```
# mdmprpt |more
MINIDUMP VERSION 4D32
***************************************************
64-bit Kernel, 10 Entries

Last Error Log Entry:
         Error ID: A924A5FC      Resource Name: SYSPROC
         Detail Data: 0000000600000000 0004A09800000004
        000000042F766172 2F61646D2F726173
        2F636F7265000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 0000000000000000
        0000000000000000 00000000736C6565
        7000000000000000 0000000000000000
        0000000000000000 0000000000007473
        7461727420314630 0A00000000000000
        0000000000000000 0000000000000020
        2068775F6672755F 69643A204E2F410A
        202068775F637075 5F69643A204E2F41
        0A00000000000000 0000000000000000
        00006E736C656570 203235340A3F3F0A
        736C656570203843 0A6D61696E203138
        340A5F5F73746172 742036430AFFDC00
        0100000000000000 0700000003353736
        3545363230300000 0000043532300000
        0000105350493200 00000023736C6565
        7000000002B00000 0006000000236E73
        6C65657000000000 2900000254

Symptom Information:
        Crash Location: [000000000042E80C] unknown
        Component: COMP  Exception Type: 267

Data From CPU #0
***************************************************
MST State:
        R0:  000000000018F3E0   R1:  F00000002FF46B80   R2:  0000000002C159B0
        R3:  00000000000110EE   R4:  0000000000000004   R5:  00000000FFFFFFFE
        R6:  0000000000010000   R7:  0000000000000100   R8:  0000000000000106
```

3.5 Live dump and component dump

Live dump and component dump capabilities are provided to allow failure data to be dumped without taking down the entire system. Listed here are the most frequent uses of these dumps:

- From the command line.

  The system administrator issues the **`livedumpstart`** command to dump data related to a failure.

- For recovery.

  A subsystem wants to dump data pertaining to the failure before recovering. (This ability is only available to the kernel and kernel extensions.)

The command **`dumpctrl`** can be used manage system dumps and live dumps. The command also has several flags and options; refer to the **`dumpctrl`** man pages (**`man dumpctrl`**) for more information.

The component dump allows the user, subsystem, or system administrator to request that one or more components be dumped. When combined with the live dump functionality, component dump allows components to be dumped, without bringing down the entire system.

When combined with the system dump functionality, component dump allows you to limit the size of the system dump or disaster dump.

Note that the main users of these functions are IBM service personnel. Examples of the live dump and component dump functions are provided here so that you can perform these dumps if requested.

3.5.1 Dump-aware components

Depending on system type and configuration, many different types of dump-aware components can exist. Later releases of AIX may add or delete components that exist on current releases. The following list describes a few important components.

In our case, our test system had the following dump-aware components:

- Inter-process communication (ipc)

  The ipc component currently has following subcomponents: semaphores data (sem); shared memory segments data (shm); message queues data (msg).

- Virtual memory manager (vmm)

  The vmm component currently has the following subcomponents: frame set data (frs); memory pool data (memp); paging device data (pdt); system control data (pfhdata); page size data (pst); interval data (vmint); kernel data (vmker); resource pool data (vmpool).

- Enhanced journaled file system (jfs2)

  The jfs2 component currently has the following subcomponents: file system data; log data.

- Logical file system (lfs)

  The lfs component currently has the following subcomponents: file system data; pile data; General Purpose Allocator Interface (gpai) data (gpai is a collector of fifo, vfs, and vnode data).

Archived

- Logical Volume Manager (lvm)

  The lvm component currently has the following subcomponents: rootvg data; other volume groups configured on the system.

- Process information (proc)

  The proc component currently has one subcomponent called watchdog. Watchdog is a kernel timer service.

- SCSI disk (scsidisk)

  This component represents the SCSI disk connected on the system, and it has subcomponents like hdisk0 and other disks, depending the configuration.

- Virtual SCSI (vscsi_initdd)

  This component represents the virtual SCSI connected on the system, and it has subcomponents like vscsi0 and other virtual SCSI devices, depending the configuration.

To see component-specific **live dump** attributes and system dump attributes on the system, you can use the **dumpctrl -qc** command; sample output is shown in Example 3-4.

Example 3-4 The output of dumpctrl -qc command

```
# dumpctrl -qc
-----------------------------------------------+------+-----------+------------
                                               | Have | Live Dump | System Dump
       Component Name                          |Alias |  /level   |  /level
-----------------------------------------------+------+-----------+------------
dump
     .livedump                                 | YES  |   OFF/3   |    ON/3
errlg                                          |  NO  |   OFF/3   |    ON/3
ipc
     .msg                                      | YES  |    ON/3   |   OFF/3
     .sem                                      | YES  |    ON/3   |   OFF/3
     .shm                                      | YES  |    ON/3   |   OFF/3
jfs2                                           |  NO  |    ON/3   |   OFF/3
     filesystem
          __1
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _admin_9
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _home_8
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _opt_11
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _tmp_5
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _usr_2
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
          _var_4
               .inode                          |  NO  |    ON/3   |   OFF/3
               .snapshot                       |  NO  |    ON/3   |   OFF/3
     log
          .A_3                                 |  NO  |    ON/3   |   OFF/3
```

Archived

```
lfs
     filesystem
          ._0                                      |  NO |    ON/3    |   OFF/3
          .__1                                     |  NO |    ON/3    |   OFF/3
          ._admin_9                                |  NO |    ON/3    |   OFF/3
          ._home_8                                 |  NO |    ON/3    |   OFF/3
          ._mksysb_16                              |  NO |    ON/3    |   OFF/3
          ._opt_11                                 |  NO |    ON/3    |   OFF/3
          ._proc_10                                |  NO |    ON/3    |   OFF/3
          ._tmp_5                                  |  NO |    ON/3    |   OFF/3
          ._usr_2                                  |  NO |    ON/3    |   OFF/3
          ._var_4                                  |  NO |    ON/3    |   OFF/3
     gpai
          .fifo                                    |  NO |    ON/3    |   OFF/3
          .vfs                                     |  NO |    ON/3    |   OFF/3
          .vnode                                   |  NO |    ON/3    |   OFF/3
     pile
          .NLC128                                  |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .NLC256                                  |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .NLC64                                   |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .PATH1024                                |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .PATH128                                 |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .PATH256                                 |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .PATH512                                 |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .PATH64                                  |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .bmIOBufPile                             |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .bmXBufPile                              |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .dioCache                                |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .dioPIOVPile                             |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .dioReq                                  |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .iCache                                  |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .j2SnapBufPool                           |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .j2VCBufferPool                          |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .logxFreePile                            |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
          .txLockPile                              |  NO |    ON/3    |   OFF/3
               .metadata                           |  NO |    ON/3    |   OFF/3
lvm                                                |  NO |    ON/3    |    ON/3
     .rootvg                                       |  NO |    ON/3    |    ON/3
          .metadata                                |  NO |    ON/3    |    ON/3
               .lvs                                |  NO |    ON/3    |    ON/3
                    .hd1                           |  NO |    ON/3    |    ON/3
                    .hd10opt                       |  NO |    ON/3    |    ON/3
                    .hd11admin                     |  NO |    ON/3    |    ON/3
```


Archived

```
                    .hd2                         |  NO |   ON/3    |    ON/3
                    .hd3                         |  NO |   ON/3    |    ON/3
                    .hd4                         |  NO |   ON/3    |    ON/3
                    .hd5                         |  NO |   ON/3    |    ON/3
                    .hd6                         |  NO |   ON/3    |    ON/3
                    .hd8                         |  NO |   ON/3    |    ON/3
                    .hd9var                      |  NO |   ON/3    |    ON/3
                    .rootvg_dalv                 |  NO |   ON/3    |    ON/3
                    .sysdump0                    |  NO |   ON/3    |    ON/3
               .pvs                              |  NO |   ON/3    |    ON/3
                    .hdisk0                      |  NO |   ON/3    |    ON/3
          .userdata                              |  NO |   ON/3    |    ON/3
               .lvs                              |  NO |   ON/3    |    ON/3
                    .hd1                         |  NO |   ON/3    |    ON/3
                    .hd10opt                     |  NO |   ON/3    |    ON/3
                    .hd11admin                   |  NO |   ON/3    |    ON/3
                    .hd2                         |  NO |   ON/3    |    ON/3
                    .hd3                         |  NO |   ON/3    |    ON/3
                    .hd4                         |  NO |   ON/3    |    ON/3
                    .hd5                         |  NO |   ON/3    |    ON/3
                    .hd6                         |  NO |   ON/3    |    ON/3
                    .hd8                         |  NO |   ON/3    |    ON/3
                    .hd9var                      |  NO |   ON/3    |    ON/3
                    .rootvg_dalv                 |  NO |   ON/3    |    ON/3
                    .sysdump0                    |  NO |   ON/3    |    ON/3
               .pvs                              |  NO |   ON/3    |    ON/3
                    .hdisk0                      |  NO |   ON/3    |    ON/3
proc
     .watchdog                                   |  NO |   ON/3    |   OFF/3
scsidiskdd                                       |  NO |   ON/3    |    ON/3
     .hdisk0                                     | YES |   ON/3    |    ON/3
vmm
     .frs                                        | YES |   ON/3    |   OFF/3
          .frs0                                  | YES |   ON/3    |   OFF/3
          .frs1                                  | YES |   ON/3    |   OFF/3
          .frs2                                  | YES |   ON/3    |   OFF/3
          .frs3                                  | YES |   ON/3    |   OFF/3
          .frs4                                  | YES |   ON/3    |   OFF/3
          .frs5                                  | YES |   ON/3    |   OFF/3
          .frs6                                  | YES |   ON/3    |   OFF/3
          .frs7                                  | YES |   ON/3    |   OFF/3
     .memp                                       | YES |   ON/3    |   OFF/3
          .memp0                                 | YES |   ON/3    |   OFF/3
          .memp1                                 | YES |   ON/3    |   OFF/3
     .pdt                                        | YES |   ON/3    |   OFF/3
          .pdt0                                  | YES |   ON/3    |   OFF/3
          .pdt80                                 | YES |   ON/3    |   OFF/3
          .pdt81                                 | YES |   ON/3    |   OFF/3
          .pdt82                                 | YES |   ON/3    |   OFF/3
          .pdt83                                 | YES |   ON/3    |   OFF/3
          .pdt84                                 | YES |   ON/3    |   OFF/3
          .pdt85                                 | YES |   ON/3    |   OFF/3
          .pdt86                                 | YES |   ON/3    |   OFF/3
          .pdt87                                 | YES |   ON/3    |   OFF/3
          .pdt88                                 | YES |   ON/3    |   OFF/3
          .pdt89                                 | YES |   ON/3    |   OFF/3
          .pdt8A                                 | YES |   ON/3    |   OFF/3
     .pfhdata                                    | YES |   ON/3    |   OFF/3
     .pst                                        | YES |   ON/3    |   OFF/3
     .vmint                                      | YES |   ON/3    |   OFF/3
```

```
    .vmker                                      | YES |   ON/3   |  OFF/3
    .vmpool                                     | YES |   ON/3   |  OFF/3
         .vmpool0                               | YES |   ON/3   |  OFF/3
         .vmpool1                               | YES |   ON/3   |  OFF/3
vscsi_initdd
    .vscsi0                                     | YES |   ON/3   |   ON/3
         .lun8100000000000000                   |  NO |   ON/3   |   ON/3
```

3.5.2 Performing a live dump

This section explains how to perform a live dump or component dump, and uses examples to clarify the tasks.

You can access the SMIT main menu of Component/Live Dump by using the **smitty livedump** command. The menu is shown in Example 3-5.

Example 3-5 Component / Live Dump Main Menu

```
                          Component / Live Dump

Move cursor to desired item and press Enter.

  Start a Live Dump
  List Components that are Dump Aware
  List Live Dumps in the Live Dump Repository
  Change/Show Global Live Dump Attributes
  Change/Show Dump Attributes for a Component
  Change Dump Attributes for multiple Components
  Refresh the Kernel's List of Live Dumps
  Display Persistent Customizations


F1=Help                       F2=Refresh                    F3=Cancel
F8=Image
F9=Shell                      F10=Exit                      Enter=Do
```

For example, if you are requested to run a live dump of virtual memory managers paging device data (vmm.pdt), you would use the **smitty livedump** menu and the **Start a Live Dump** submenu. The menu is shown in Example 3-6.

Example 3-6 Start a Live Dump submenu

```
                                Start a Live Dump

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

                                                     [Entry Fields]
  Component Path Names
  Component Logical Alias Names                   [@pdt]
  Component Type/Subtype Identifiers              []
  Pseudo-Component Identifiers                    []
  Error Code                                      []
  Symptom String                                  [any-description]
  File Name Prefix                                []
  Dump Priority                                   [critical]             +
  Generate an Error Log Entry                     [yes]                  +
```

```
  Dump Types                                         [serialized]           +
  Force this Dump                                    [yes]                  +
  Dump Subcomponents                                 [yes]                  +
  Dump Parent Components                             [no]                   +
  Obtain a Size Estimate                             [no]                   +
          (no dump is taken)
  Show Parameters for Components                     [no]                   +
          (no dump is taken)

F1=Help               F2=Refresh             F3=Cancel              F4=List
F5=Reset              F6=Command             F7=Edit                F8=Image
F9=Shell              F10=Exit               Enter=Do
```

Table 3-2 provides guidelines for filling in the menu fields.

Table 3-2 Guidelines for filling in Start a Live Dump submenu

Menu field	Explanation
Component Path Names	Can be found by output of dumpctrl. For instance, ipc.msg is a component path name. Not needed if any of Alias/Type/Pseudo-component is specified. Placing an at (@) sign in front of the component specifies that it is the failing component (to replace the nocomp in the file name).
Component Logical Alias Names	Can be found by output of dumpctrl for those components where Have Alias is listed as YES. For instance, msg is a valid Alias. Not needed if any of Path Name/Type/Pseudo-component is specified. Placing an at (@) sign before the alias specifies it as the failing component (to replace the nocomp in the file name).
Component Type/Subtype Identifiers	Specify type/subtypes from /usr/include/sys/ rasbase.h. Not all types or subtypes map to valid livedump components.
Pseudo-Component Identifiers	Built-in components not present in dumpctrl -qc. A list of them can be obtained by selecting the help for this topic. They require parameters to be specified with the identifiers. Details about the parameters are provided by an error message if you do not specify the parameters. Not needed if any of Path Name/Alias/Type is specified.
Error Code	Optional string to include inside dump.
Symptom String	Mandatory string that describes the nature of the live dump.
File Name Prefix	Prefix to append to a file name.
Dump Priority	Choice of critical/info. Critical dumps may delete info dumps if there is not enough room in the filesystem. Critical dumps can also be held in kernel memory.
Generate an Error Log Entry	Whether or not to generate an error log entry in the event of live dump completion.
Dump Types	Serialized is the only valid option in 6.1.
Force this Dump	Applies to whether or not duplicate elimination can potentially abort this live dump.
Dump Subcomponents	For any components specified by Path Name/Alias/Type, whether or not to dump their subcomponents. Subcomponents appear under and to the right of other components in dumpctrl -qc. (For instance, ipc.sem and ipc.msg are subcomponents of ipc.)
Dump Parent Components	For any components specified by Path Name/Alias/Type, whether or not to dump their parent components. (For instance, the parent of ipc.sem is ipc.)
Obtain a Size Estimate	Do not take a dump, just get an estimate for the compressed size. (No dump is taken)
Show Parameters for Components	If a component has help related to it, show that help information.

The output of Start a Live Dump is shown in Example 3-7.

Example 3-7 Output of executing Start a Live Dump menu

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

The dump is in file /var/adm/ras/livedump/pdt.200710111501.00.DZ.


F1=Help                F2=Refresh            F3=Cancel             F6=Command
F8=Image               F9=Shell              F10=Exit              /=Find
n=Find Next
```

Before the dump file can be analyzed, it needs to be uncompressed with the **dmpuncompress** command. Output from a **dmpuncompress** command is shown Example 3-8.

Example 3-8 Uncompress the dumpfile

```
# ls -l /var/adm/ras/livedump/pdt.200710111501.00.DZ
-rw-------    1 root     system         4768 Oct 11 10:24
/var/adm/ras/livedump/pdt.200710111501.00.DZ
# dmpuncompress /var/adm/ras/livedump/pdt.200710111501.00.DZ
 -- replaced with /var/adm/ras/livedump/pdt.200710111501.00
#
```

You can use the **kdb** command to analyze a live dump file. Output from a **kdb** command is shown in Example 3-9.

Example 3-9 Analyzing a live dump using the kdb command

```
# kdb /var/adm/ras/livedump/pdt.200710111501.00
# kdb /var/adm/ras/livedump/pdt.200710111501.00
/var/adm/ras/livedump/pdt.200710111501.00 mapped from @ 700000000000000 to @
70000000000cd88
Preserving 1675375 bytes of symbol table [/unix]
Component Names:
 1)  dump.livedump.header [3 entries]
 2)  dump.livedump.sysconfig [3 entries]
 3)  vmm.pdt [14 entries]
 4)  vmm.pdt.pdt8B [2 entries]
 5)  vmm.pdt.pdt8A [2 entries]
 6)  vmm.pdt.pdt89 [2 entries]
 7)  vmm.pdt.pdt88 [2 entries]
 8)  vmm.pdt.pdt87 [2 entries]
 9)  vmm.pdt.pdt86 [2 entries]
10)  vmm.pdt.pdt81 [2 entries]
11)  vmm.pdt.pdt82 [2 entries]
12)  vmm.pdt.pdt85 [2 entries]
13)  vmm.pdt.pdt84 [2 entries]
14)  vmm.pdt.pdt83 [2 entries]
15)  vmm.pdt.pdt80 [2 entries]
16)  vmm.pdt.pdt0 [2 entries]
17)  cu [2 entries]
Live Dump Header:
  Passes: 1
  Type: LDT_FORCE
```


Archived

```
  Priority: LDPP_CRITICAL
  Error Code: 0
  Failing Component: pdt
  Symptom: any-description
  Title:
         START              END <name>
0000000000001000 0000000003FB0000 start+000FD8
F00000002FF47600 F00000002FFDF940 __ublock+000000
000000002FF22FF4 000000002FF22FF8 environ+000000
000000002FF22FF8 000000002FF22FFC errno+000000
F100080700000000 F100080710000000 pvproc+000000
F100080710000000 F100080718000000 pvthread+000000
PFT:
PVT:
Unable to find kdb context
Dump analysis on CHRP_SMP_PCI POWER_PC POWER_6 machine with 2 available CPU(s)  (64-bit
registers)
Processing symbol table...
.......................done
cannot read vscsi_scsi_ptrs
(0)>
```

Using command line functions - example

Here, assume that I/O to the virtual disk hdisk0 is hanging. A live dump is taken to supplement any information recorded in the AIX error log. The live dump includes state information about hdisk0, state information about the underlying virtual SCSI adapter (vscsi0) and its LUN subcomponent, and the Component Trace buffer associated with vscsi0. hdisk1 is considered the failing component. The command is shown in Example 3-10.

Example 3-10   Command line example of livedumpstart command

```
# livedumpstart -L hdisk0 -l vscsi0+ -p comptrace:vscsi0,0 symptom="failing_io"
title="Sample Live Dump"
0453-142 The dump is in file /var/adm/ras/livedump/hdisk0.200710121432.00.DZ
```

3.6 Concurrent AIX Update

Concurrent AIX Update allows fixes to the base kernel and kernel extensions to be applied and simultaneously become fully operational on a running system. The system does not require any subsequent reboot to activate fixes. Concurrent AIX Update allows the following significant enhancements:

- The ability to apply preventive maintenance or corrective service fix without requiring a reboot.
- The ability to reject an applied fix without requiring a reboot.
- The ability to inject diagnostics or temporary circumventions without requiring a reboot.
- Encourages customer adoption of corrective services and thereby decreases outages for which fixes existed.
- Improves customer satisfaction.
- Improves system uptime.

3.6.1 Concurrent AIX Update terminology

Table 3-3 on page 72 lists Concurrent AIX Update terms and definitions.

Archived

Table 3-3 Concurrent update terminology

Term	Definition
Concurrent AIX Update	An update to the *in-memory image*, update of the base kernel and/kernel extensions that is immediately active or subsequently inactive without reboot. Synonyms: hotpatch, live update, online update.
Control file	A text file that contains all information required by **emgr** command to perform any update, whether concurrent or not.
Timestamp	The timestamp recorded within the XCOFF header of the module targeted for updating. It is used to validate that the ifix was created for the proper module level.
Description file	A text file derived from the emgr control file, and which contains only information relevant to Concurrent AIX Update. The **kpatch** command requires this file to apply a Concurrent AIX Update.
Fix or patch	Interchangeably used to refer to a software modification created to resolve a customer problem.
ifix	Interim fix. Package delivered for customer for temporary fixes. Synonym: efix
Kernel extension	Used to refer both kernel extensions and device drivers.
Module	An executable such as the kernel or a kernel extension.
Patchset	The complete set of patches composing a fix.
Target or target module	An executable that is to be patched (kernel or kernel extension).

3.6.2 Concurrent AIX Update commands and SMIT menus

The customer will manage Concurrent AIX Updates by using the **emgr** command. The **emgr** command has several flags; for additional information about the flags, refer to the man pages (use the **man emgr** command). You can also obtain limited help information by simply entering the **emgr** command.

Concurrent updates are packaged by using the **epkg** command. The **epkg** command also has multiple flags; for additional information about the flags refer to the man pages (use the **man epkg** command). You can also obtain limited help information by simply entering the **epkg** command.

The SMIT menus

You can access the **emgr** main menu by using the **smitty emgr** command. The menu is shown in Example 3-11.

Example 3-11 EFIX Management main menu

```
                             EFIX Management

Move cursor to desired item and press Enter.

  List EFIXES and Related Information
  Install EFIX Packages
  Remove Installed EFIXES
```

```
  Check Installed EFIXES

F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

From the List EFIXES and Related Information menu, you can select an emergency fix (EFIX) to be listed by the EFIX label, EFIX ID, or virtually unique ID (VUID). The menu is shown in Example 3-12.

Example 3-12 List EFIXES and Related Information menu

```
                          List EFIXES and Related Information

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

                                                       [Entry Fields]
  EFIX Label                                        [ALL]                    +

  VERBOSITY Level                                   [1]                      +
  DEBUG output?                                      no                      +



F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

From the Install EFIX Packages menu, you can specify the location of the emergency fix (EFIX) package to be installed or previewed. The EFIX package should be a file created with the **epkg** command. The menu is shown in Example 3-13.

Important: Concurrent update package files will have a name ending with *.epkg.Z. These files are compressed and should *not* be uncompressed before installation.

Example 3-13 Install EFIX Packages menu

```
                                 Install EFIX Packages

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

                                                       [Entry Fields]
  LOCATION of EFIX Package                          []                        /
    -OR-
  LOCATION of Package List File                     []                        /

  PREVIEW only? (install operation will NOT occur)   no                      +
  MOUNT Installation?                                no                      +
  EXTEND file systems if space needed?               yes                     +
  DEBUG output?                                      no                      +



F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```


Archived

We tested the installation of an efix called beta_patch.070909.epkg.Z. We placed the efix file in the /usr/emgrdata/efixdata directory for the installation. The output of the installation test is shown in Example 3-14.

Example 3-14 Output of EFIX installation menu

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[TOP]
+-----------------------------------------------------------------------------+
Efix Manager Initialization
+-----------------------------------------------------------------------------+
Initializing log /var/adm/ras/emgr.log ...
Efix package file is: /usr/emgrdata/efixdata/beta_patch.070909.epkg.Z
MD5 generating command is /usr/bin/csum
MD5 checksum is a7710363ad00a203247a9a7266f81583
Accessing efix metadata ...
Processing efix label "beta_patch" ...
Verifying efix control file ...
+-----------------------------------------------------------------------------+
Installp Prerequisite Verification
+-----------------------------------------------------------------------------+
No prerequisites specified.
Building file-to-package list ...

+-----------------------------------------------------------------------------+
Efix Attributes
+-----------------------------------------------------------------------------+
LABEL:            beta_patch
PACKAGING DATE:   Sun Sep  9 03:21:13 CDT 2007
ABSTRACT:         CU ifix for cu_kext
PACKAGER VERSION: 6
VUID:             00C6CC3C4C00090903091307
REBOOT REQUIRED:  no
BUILD BOOT IMAGE: no
PRE-REQUISITES:   no
SUPERSEDE:        no
PACKAGE LOCKS:    no
E2E PREREQS:      no
FIX TESTED:       no
ALTERNATE PATH:   None
EFIX FILES:       1

Install Scripts:
   PRE_INSTALL:   no
   POST_INSTALL:  no
   PRE_REMOVE:    no
   POST_REMOVE:   no

File Number:      1
   LOCATION:      /usr/lib/drivers/cu_kext
   FILE TYPE:     Concurrent Update
   INSTALLER:     installp (new)
   SIZE:          8
ACL:           root:system:755
   CKSUM:         60788
```

```
   PACKAGE:       None or Unknown
   MOUNT INST:    no

+-----------------------------------------------------------------------------+
Efix Description
+-----------------------------------------------------------------------------+
CU ifix - test ifix for /usr/lib/drivers/cu_kext

+-----------------------------------------------------------------------------+
Efix Lock Management
+-----------------------------------------------------------------------------+
Checking locks for file /usr/lib/drivers/cu_kext ...

All files have passed lock checks.

+-----------------------------------------------------------------------------+
Space Requirements
+-----------------------------------------------------------------------------+
Checking space requirements ...

Space statistics (in 512 byte-blocks):
File system: /usr, Free: 3310192, Required: 1250, Deficit: 0.
File system: /tmp, Free: 1041216, Required: 2486, Deficit: 0.

+-----------------------------------------------------------------------------+
Efix Installation Setup
+-----------------------------------------------------------------------------+
Unpacking efix package file ...
Initializing efix installation ...

+-----------------------------------------------------------------------------+
Efix State
+-----------------------------------------------------------------------------+
Setting efix state to: INSTALLING

+-----------------------------------------------------------------------------+
Efix File Installation
+-----------------------------------------------------------------------------+
Installing all efix files:
Installing efix file #1 (File: /usr/lib/drivers/cu_kext) ...

Total number of efix files installed is 1.
All efix files installed successfully.

+-----------------------------------------------------------------------------+
Package Locking
+-----------------------------------------------------------------------------+
Processing package locking for all files.
File 1: no lockable package for this file.

All package locks processed successfully.

+-----------------------------------------------------------------------------+
Reboot Processing
+-----------------------------------------------------------------------------+
Reboot is not required by this efix package.

+-----------------------------------------------------------------------------+
Efix State
+-----------------------------------------------------------------------------+
```

Archived

```
Setting efix state to: STABLE

+-----------------------------------------------------------------------------+
Operation Summary
+-----------------------------------------------------------------------------+
Log file is /var/adm/ras/emgr.log

EPKG NUMBER       LABEL             OPERATION             RESULT
===========       ==============    =================     ==============
1                 beta_patch        INSTALL               SUCCESS

Return Status = SUCCESS
```

With the Remove Installed EFIXES menu, you can select an emergency fix to be removed.The menu is shown in Example 3-15.

Example 3-15 Remove Installed EFIXES menu

```
                           Remove Installed EFIXES

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

                                                        [Entry Fields]
  EFIX Label                                         []                     +
    -OR-
  LOCATION of EFIX List File                         []                      /

  PREVIEW only? (remove operation will NOT occur)     yes                   +
  EXTEND file systems if space needed?                yes                   +
  DEBUG output?                                       no                    +


F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

We also tested the removal of the efix. Part of the output from the test is shown in Example 3-16.

Example 3-16 Partial output of Remove Installed EFIXES menu

```
                              COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[TOP]
+-----------------------------------------------------------------------------+
Efix Manager Initialization
+-----------------------------------------------------------------------------+
Initializing log /var/adm/ras/emgr.log ...
Accessing efix metadata ...
Processing efix label "beta_patch" ...

+-----------------------------------------------------------------------------+
Efix Attributes
+-----------------------------------------------------------------------------+
```


Archived

```
LABEL:           beta_patch
INSTALL DATE:    10/11/07 15:36:45
[MORE...87]

F1=Help            F2=Refresh         F3=Cancel          F6=Command
F8=Image           F9=Shell           F10=Exit           /=Find
n=Find Next
```

From the Check Installed EFIXES menu, you can select the fixes to be checked. The menu is shown in Example 3-17.

Example 3-17 Check Installed EFIXES menu

```
                              Check Installed EFIXES

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

                                                     [Entry Fields]
  EFIX Label                                        []                     +
    -OR-
  LOCATION of EFIX List File                        []                      /

  VERIFICATION Level                                [1]                    +
  DEBUG output?                                      no                    +



F1=Help            F2=Refresh         F3=Cancel          F4=List
F5=Reset           F6=Command         F7=Edit            F8=Image
F9=Shell           F10=Exit           Enter=Do
```

3.7 Storage protection keys

Storage protection keys are designed to alleviate software issues that can appear because of memory overlays and addressing errors. In AIX, a large global address space is shared among a variety of software components and products.

This section explains storage protection keys and describes how kernel programmers or application programmers use them. The storage protection key concept was adopted from the z/OS® and 390 systems, so AIX can confidently rely on this feature and its robustness.

3.7.1 Storage protection keys overview

Storage protection keys and their support by the AIX kernel are new capabilities introduced with AIX Version 5.3 (user keys) and Version 6.1 (user and kernel keys) running on POWER6 hardware. In this paper, storage protection keys are also called *storage keys* or simply *keys*. Keys provide a context-sensitive storage protection mechanism for virtual memory pages.

Software might use keys to protect multiple classes of data individually and to control access to data on a per-context basis. This differs from the older page protection mechanism, which is global in nature.

Storage keys are available in both kernel-mode and user-mode application binary interfaces (ABIs). In kernel-mode ABIs, storage key support is known as *kernel keys*. In user space

Archived

mode, storage keys are called *user keys*. A kernel extension may have to deal with both types of keys.

A memory protection domain generally uses multiple storage protection keys to achieve additional protection. AIX Version 6.1 divides the system into four memory protection domains, as described here:

Kernel public This term refers to kernel data that is available without restriction to the kernel and its extensions, such as stack, bss, data, and areas allocated from the kernel or pinned storage heaps.

Kernel private This term refers to data that is largely private within the AIX kernel proper, such as the structures representing a process.

Kernel extension This term refers to data that is used primarily by kernel extensions, such as file system buf structures.

User This term refers to data in an application address space that might be using key protection to control access to its own data.

One purpose of the various domains (except for kernel public) is to protect data in a domain from coding accidents in another domain. To a limited extent, you can also protect data within a domain from other subcomponents of that domain. When coding storage protection into a kernel extension, you can achieve some or all of the following RAS benefits:

- Protect data in user space from accidental overlay by your extension
- Respect private user key protection used by an application to protect its own private data
- Protect kernel private data from accidental overlay by your extension
- Protect your private kernel extension data from accidental overlay by the kernel, by other kernel extensions, and even by subcomponents of your own kernel extension

Note: Storage protection keys are *not* meant to be used as a security mechanism. Keys are used following a set of voluntary protocols by which cooperating subsystem designers can better detect, and subsequently repair, programming errors.

Design considerations

The AIX 64-bit kernel makes extensive use of a large flat address space by design. It produces a significant performance advantage, but also adds Reliability, Accessibility and Serviceability (RAS) costs.

Storage keys were introduced into PowerPC architecture to provide memory isolation while still permitting software to maintain a flat address space. Large 64-bit applications, such as DB2, use a global address space for similar reasons and also face issues with memory overlays.

Storage keys allow an address space to be assigned context-specific protection. Access to the memory regions can be limited to prevent and catch illegal storage references. PowerPC hardware assigns protection keys to virtual pages. The keys are identifiers of a memory object class. These keys are kept in memory translation tables. They are also cached in translation look-aside buffers (TLBs) and in effect to real TLBs (ERATs).

A new CPU special purpose register known as the Authority Mask Register (AMR) has been added to define the keyset that the CPU has access to. The AMR is implemented as a bit pairs vector indexed by key number, with distinct bits to control read and write access for each key. The key protection is in addition to the existing page protection mechanism.

Archived

For any load/store, the CPU retrieves the memory key assigned to the targeted page during the translation process. The key number is used to select the bit pair in the AMR that defines if an access is permitted. A data storage interrupt results when this check fails.

The AMR is a per-context register that can be updated efficiently. The TLB/ERAT contains storage key values for each virtual page. This allows AMR updates to be efficient, because they do not require TLB/ERAT invalidation. The POWER hardware enables a mechanism that software can use to efficiently change storage accessibility.

Ideally, each storage key would correspond to a hardware key. However, due to the limited number of hardware keys with current Power Architecture, more than one kernel key is frequently mapped to a given hardware key. This key mapping or level of indirection may change in the future as architecture supports more hardware keys.

Another advantage that indirection provides is that key assignments can be changed on a system to provide an exclusive software-to-hardware mapping for a select kernel key. This is an important feature for testing fine granularity keys. It could also be used as an SFDC tool. Kernel keys provide a formalized and abstracted API to map kernel memory classes to a limited number of hardware storage keys.

For each kernel component, data object (virtual pages) accessibility is determined. Then component entry points and exit points may be wrapped with “protection gates”. Protection gates change the AMR as components are entered and exited, thus controlling what storage can be accessed by a component.

At configuration time, the module initializes its kernel keyset to contain the keys required for its runtime execution. The kernel keyset is then converted to a hardware keyset. When the module is entered, the protection gates set the AMR to the required access authority efficiently by using a hardware keyset computed at configuration time.

User keys work in application programs. The keys are a virtualization of the PowerPC storage hardware keys. Access rights can be added and removed from a user space AMR, and an user key can be assigned as appropriate to an application’s memory pages. Management and abstraction of user keys is left to application developers.

3.7.2 System management support for storage keys

You can use **smitty skeyctl** to disable (keys are enabled by default) Kernel keys, as shown in Example 3-18.

Note: You may want to disable kernel keys if one of your kernel extensions is causing key protection errors but you need to be able to run the system even though the error has not been fixed.

Example 3-18 Storage protection keys SMIT menu

```
                              Storage Protection Keys

Move cursor to desired item and press Enter.

  Change/Show Kernel Storage Protection Keys State


F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Example 3-19 shows the SMIT menu for the current storage key setting.

Example 3-19 SMIT menu to check current storage key setting

```
            Change/Show Kernel Storage Protection Keys State

Move cursor to desired item and press Enter.

  Show Kernel Storage Protection Keys State
  Change Next Boot Kernel Storage Protection Keys State


F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

The current storage key setting for our test system is shown in Example 3-20.

Example 3-20 Current storage key setting

```
                         COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

Current Kernel Storage Protection Keys State is enabled.
Next Boot Kernel Storage Protection Keys State is default.


F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

You can change the next boot storage key setting by using the second option of the SMIT menu shown in Example 3-19.

Note: Although you may be able to use the command line for checking and altering storage key settings, this is not supported for direct use. Only SMIT menus are supported.

3.7.3 Kernel mode protection keys

Kernel keys provide a Reliability function by limiting the damage that a software component can do to other parts of the system. The keys will prevent kernel extensions from damaging core kernel components, and provide isolation between kernel extension classes. Kernel keys will also help provide a significant Availability function by helping to prevent error propagation.

Note: Kernel and kernel extensions are not necessarily protected from other kernel extensions. Kernel and kernel extensions are only protected (to the extent possible) from key-safe extensions. For details about protection degrees, refer to 3.7.4, “Degrees of storage key protection and porting considerations” on page 84, which describes key-safe and key-unsafe kernel extensions.

Furthermore, key-unsafe extensions’ access rights are also somewhat limited. In particular, certain kernel parts appear only as read-only to the key-unsafe kernel extensions.

Serviceability is enhanced by detecting memory addressing errors closer to their origin. Kernel keys allow many random overlays to be detected when the error occurs, rather than when the corrupted memory is used.

With kernel key support, the AIX kernel introduces kernel domains and private memory access. Kernel domains are component data groups that are created to segregate sections of the kernel and kernel extensions from each other. Hardware protection of kernel memory domains is provided and enforced. Also, global storage heaps are separated and protected. This keeps heap corruption errors within kernel domains.

There are also private memory keys that allow memory objects to be accessed only by authorized components. In addition to RAS benefits, private memory keys are a tool to enforce data encapsulation. There is a static kernel key-to-storage key mapping function set by the kernel at boot time. This mapping function is dependent on the number of storage keys that are present in the system.

Note: Kernel keys are *not* intended to provide a security function. There is no infrastructure provided to authorize access to memory. The goal is to detect and prevent accidental memory overlays. The data protection kernel keys provide can be circumvented, but this is by design. Kernel code, with the appropriate protection gate, can still access any memory for compatibility reasons.

Analogy

To understand the concept of kernel storage keys, consider a simple analogy. Assume there is a large house (the kernel) with many rooms (components and device drivers) and many members (kernel processes and kernel execution path), and each member has keys only for a few other selected rooms in addition to its own room (keyset).

Therefore, members having a key for a room are considered safe to be allowed inside. Every time a member wants to enter a room, the member needs to see whether its keyset contains a key for that room.

If the member does not have the corresponding key, it can either create a key which will permit the member to enter the room (which means it will add the key to its keyset). Or the member can try to enter without a key. If the member tries to enter without a key, an alarm will trip (cause a DSI/Kernel crash) and everything will come to a halt because one member (a component or kernel execution path) tried to intrude into an unauthorized room.

Kernel keys

The kernel's data is classified into kernel keys according to intended use. A *kernel key* is a software key that allows the kernel to create data protection classes, regardless of the number of hardware keys available. A kernel keyset is a representation of a collection of kernel keys and the desired read or write access to them. Remember, several kernel keys

might share a given hardware key. Most kernel keys are for use only within the kernel (the full list can be found in sys/skeys.h). Table 3-4 shows the kernel keys that are likely to be useful to kernel extension developers.

Table 3-4 Useful kernel keys

Key name	Description
KKEY_PUBLIC	This kernel key is always necessary for access to a program's stack, bss, and data regions. Data allocated from the pinned_heap and the kernel_heap is also public.
KKEY_BLOCK_DEV	This kernel key is required for block device drivers. Their buf structs must be either public or in this key.
KKEY_COMMO	This kernel key is required for communication drivers. CDLI structures must be either public or in this key.
KKEY_NETM	This kernel key is required for network and other drivers to reference memory allocated by net_malloc.
KKEY_USB	This kernel key is required for USB device drivers.
KKEY_GRAPHICS	This kernel key is required for graphics device drivers.
KKEY_DMA	This kernel key is required for DMA information (DMA handles and EEH handles).
KKEY_TRB	This kernel key is required for timer services (struct trb).
KKEY_IOMAP	This kernel key is required for access to I/O-mapped segments.
KKEY_FILE_SYSTEM	This kernel key is required to access vnodes and gnodes (vnop callers).

Note: Table 3-4 shows a preliminary list of kernel keys. As kernel keys are added to components, additional kernel keys will be defined.

The current full list can be found in /usr/include/sys/skeys.h.

Kernel keysets

Note: Not all keys in the kernel key list are currently enforced. However, it is always safe to include them in a keyset.

Because the full list of keys might evolve over time, the only safe way to pick up the set of keys necessary for a typical kernel extension is to use one of the predefined kernel keysets, as shown the following list.

KKEYSET_KERNEXT The minimal set of keys needed by a kernel extension.
KKEYSET_COMMO Keys needed for a communications or network driver.
KKEYSET_BLOCK Keys needed for a block device driver.
KKEYSET_GRAPHICS Keys needed for a graphics device driver.
KKEYSET_USB Keys needed for a USB device driver.

See sys/skeys.h for a complete list of the predefined kernel keysets. These keysets provide read and write access to the data protected by their keys. If you want simply read access to keys, those sets are named by appending _READ (as in KKEYSET_KERNEXT_READ).

Archived

It is acceptable (though not required) to remove unwanted keys from copies of the sets mentioned. For example, KKEY_TRB might be unnecessary because your code does not use timer services. However, building a keyset from scratch by explicitly adding just the kernel keys you need is *not* recommended. The kernel's use of keys is likely to evolve over time, which could make your keyset insufficient in the future. Any new kernel keys defined that kernel extensions might need will be added to the basic predefined keysets mentioned, thus ensuring that you will hold these new keys automatically.

Hardware and operating system considerations

- AIX kernel - kernel key exploitation is provided with the AIX V6.1 release. There is no retrofit enablement.
- AIX V6.1 will continue to function on supported legacy hardware that does not support storage keys.
- Kernel keys are only available with the 64-bit kernel.
- Kernel keys are enabled by default on key-capable machines. This is the normal operational mode.
- A reboot is required in order to enable or disable kernel keys.

Kernel key-to-hardware key mapping

The PowerPC architecture allows implementations to provide 2 to 32 hardware keys. The initial POWER6 implementation provides support for 8 keys.

For all supported hardware configurations, the ability exists to force one kernel key to be mapped exclusively to one hardware key. Kernel mappings can be expected to become more fine-grained as the number of available hardware keys increases. These mappings will be updated as new keys are defined.

The hardware key number (a small integer) is associated with a virtual page through the page translation mechanism. The AMR special purpose register is part of the executing context; see Figure 3-2. The 32-bit-pair keyset identifies the keys that the kernel has at any point of execution. If a keyset does not have the key to access a page, it will result in a DSI/system crash.

Archived

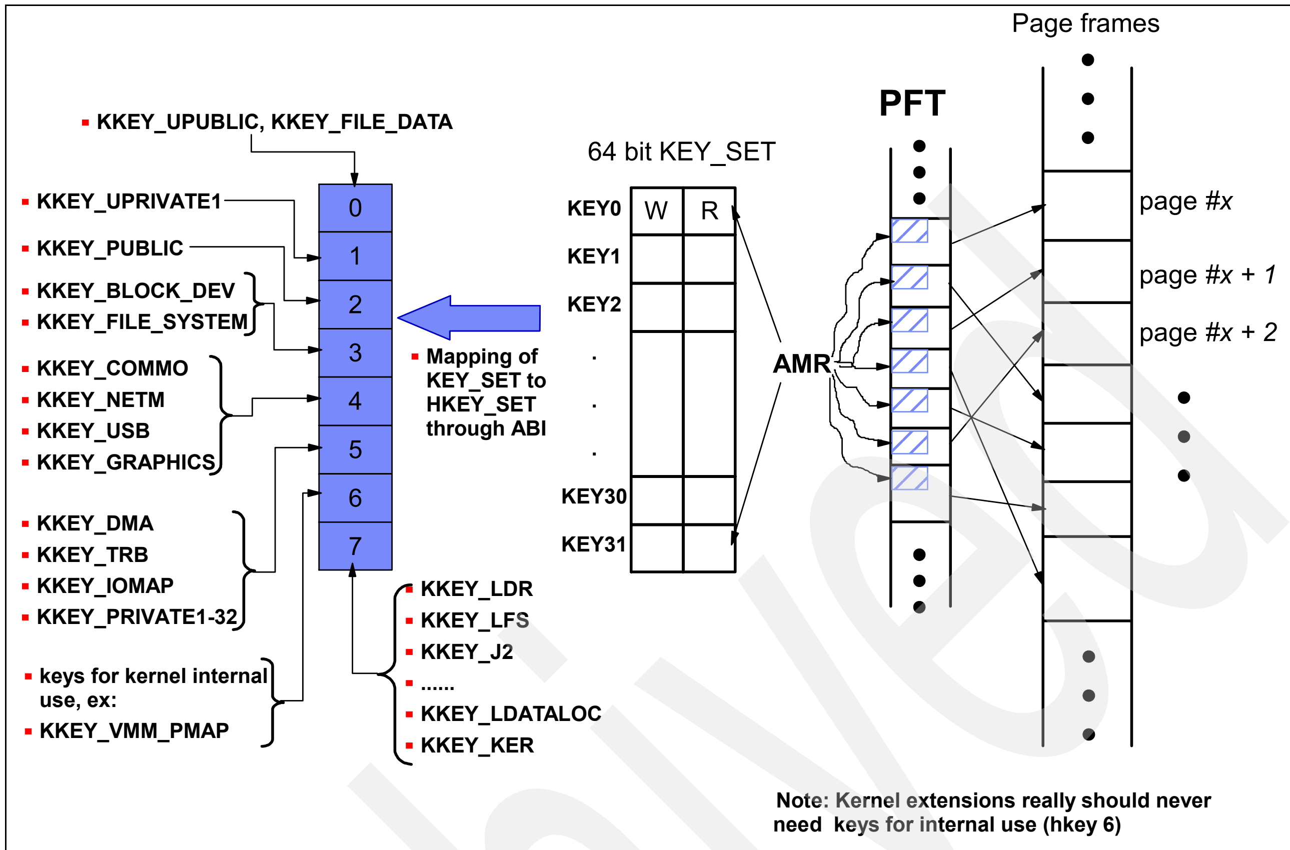


Figure 3-2 PFT entry with AMR for a typical kernel process/execution path on P6 hardware

For P6 hardware, there are only eight available hardware keys, so each keyset will be mapped to a 16-bit AMR. Each bit-pair in AMR may have more than one key mapped to it. For example, if key 4 is set in AMR, that means at least one of KKEY_COMMO, KKEY_NETM, KKEY_USB, and KKEY_GRAPHICS has been added to the hardware keyset.

Two base kernel domains are provided. Hardware key 6 is used for critical kernel functions. Hardware key 7 for all other base kernel keys. Hardware key 5 is used for kernel extension private data keys. Hardware keys 3 to 5 are used for kernel extension domains. Two keys are dedicated for user mode kernel keys. KKEY_UPRIVATE1 is allocated by default for potential user mode.

3.7.4 Degrees of storage key protection and porting considerations

A kernel extension might support storage protection keys to varying degrees, depending on its unique requirements. These degrees are explored in the next section.

Key-unsafe kernel extension

A key-unsafe kernel extension does not contain any explicit support for storage protection keys. Extensions in this class are older code written without regard to storage key protection, needing largely unrestricted access to all memory.

It is the kernel's responsibility to ensure that legacy code continues to function as it did on prior AIX releases and on hardware without storage key support, even though such code might access kernel private data.

Archived

To continue running in a key-protected environment, legacy kernel extensions receive special support from the kernel. Any extension converted to use keys is still in an environment with a mixture of key-aware and key-unsafe functions. A key-aware kernel extension might call a service that is in a key-unsafe kernel extension.

When a key-unsafe function is called, the kernel must, in effect, transparently insert special glue code into the call stack between the calling function and the key-unsafe called function. This is done automatically, but it is worth understanding the mechanism because the inserted glue code is visible in stack callback traces.

When legacy code is called, either directly by calling an exported function or indirectly by using a function pointer, the kernel must:

1. Save the caller's current key access rights (held in the AMR).
2. Save the caller's link register (LR).
3. Replace the current AMR value with one granting broad data access rights.
4. Proceed to call the key-unsafe function, with the link register set, so that the called function returns to the next step.
5. Restore the original caller's AMR and LR values.
6. Return to the original caller.

The AMR update must be performed transparently, thus the new AMR stack had to be developed. The new resource is also called a *context stack*. The current context stack pointer is maintained in the active kmstsave structure, which holds the machine state for the thread or interrupt context. Use the **`mst`** kernel debugger command to display this information. The context stack is automatically pinned for key-unsafe kernel processes. The setjmpx and longjmpx kernel services maintain the AMR and the context stack pointer.

When a context stack frame needs to be logically inserted between standard stack frames, the affected function (actually, the function's traceback table) is flagged with an indicator. The debugger recognizes this and is able to provide you with a complete display for the stack trace. The inserted routine is named hkey_legacy_gate. A similar mechanism is applied at many of the exported entry points into the kernel, where you might observe the use of kernel_add_gate and kernel_replace_gate.

This processing adds overhead when an exported key-unsafe function is called, but only when the called function is external to the calling module. Exported functions are represented by function descriptors that are modified by the loader to enable the AMR changing service to front-end exported services. Intramodule calls do not rely on function descriptors for direct calls, and thus are not affected.

All indirect function pointer calls in a key-aware kernel extension go through special kernel-resident glue code that performs the automatic AMR manipulations as described. If you call out this way to key-unsafe functions, the glue code recognizes the situation and takes care of it for you. Hence, a key-aware kernel extension must be compiled with the **`-q noinlglue`** option for glue code.

Key-safe kernel extension

A key-safe kernel extension manages its access to memory, respecting the boundaries of the kernel and user domains. It does not directly reference either kernel private data structures or user space addresses. It also does not attempt to protect its own private data (see “Key-protected kernel extension” on page 86). To become key-safe, the extension must explicitly select the existing memory domains which it intends to access. This protects the rest of the system from errors in the key-safe module.

Note: Kernel keys and keysets, which are described later in this section, are defined here:

- A *kernel key* is a virtual key that is assigned by the kernel. Kernel programs can use more virtual keys than exist in the hardware, because many kernel keys can share a single hardware key.

  The kernel data is classified into keys according to function. You can use kernel keys to define a large number of virtual storage protection keys. Most kernel keys are used only within the kernel. The sys/skeys.h file contains a list of all keys.
- A *kernel keyset* is a grouping of kernel keys based on usage scope (see "Kernel keysets" on page 82). The sys/skeys.h file contains a list of all predefined keysets.

To make a kernel extension key-safe, follow these steps:

1. Decide which exported kernel keyset, if any, should be the basis for your module's keyset.
2. Optionally, remove any unnecessary keys from your copy of this kernel keyset.
3. Convert the kernel keyset to a hardware keyset.
4. Place add or replace (protection) gates (see 3.7.5, "Protection gates" on page 87) at or near all entry points (except driver initialization) as needed to gain the specific data access rights required by each entry point. You will no longer have direct access to user space by default, and you might need to address that issue at a later time.
5. Place your restore gates at or near exit points.
6. Link your extension with the new `–b` RAS flag to identify it to the system as RAS-aware.
7. Do *not* specify inline pointer glue (`-q inlglue`), as previously mentioned.
   Note that if you are compiling with the v9 or later xlC compiler, you must specify `–q noinlglue` because the default has changed.

Your initialization or configuration entry point cannot start off with a protection gate whose underlying hardware keyset it must first compute. Only after setting up the necessary hardware keysets can you implement your protection gates.

The computation of these keysets should be done only once (for example, when the first instance of an adapter is created). These are global resources used by all instances of the driver. Until you can use your new protection gates, you must be sure to reference only data that is unprotected, such as your stack, bss, and data regions.

If this is particularly difficult for some reason, you can statically initialize your hardware keyset to HKEYSET_GLOBAL. That initial value allows your protection gates to work even before you have constructed your kernel and hardware keysets, although they would grant the code following them global access to memory until after the hardware keysets have been properly initialized. If your extension accepts and queues data for future asynchronous access, you might also need to use HKEYSET_GLOBAL, but only if this data is allowed to be arbitrarily key-protected by your callers. Use of the global keyset should be strictly minimized.

If you want to be certain that a hardware keyset is not used unexpectedly, statically initialize it to HKEYSET_INVALID. A replace gate with this hardware keyset would revoke all access to memory and cause a DSI almost immediately.

Your protection gates protect kernel data and the data of other modules from many of the accidental overlays that might originate in your extension. It should not be necessary to change any of the logic of your module to become key safe. But your module's own data remains unprotected. The next step is to protect your kernel extension.

Key-protected kernel extension

A key-protected kernel extension goes beyond key safe; it identifies and protects its own private data, as well as data in other domains, from accidental access. This protection can be

Archived

achieved by using a private kernel key, or by taking advantage of a shared key that you are already using.

A kernel extension that is either key-safe or key-protected is called “key-aware”. To make a kernel extension key-aware, you must understand the kernel’s use of keys. To make a kernel extension key-protected, you must also define its private or semi-private data and how it uses keys to protect that data. A semi-private key might be used to share data among several related kernel extensions. A private key would be for the exclusive use of a single extension.

Making a kernel extension fully key-protected adds more steps to the port. You now must also follow these steps:

1. Analyze your private data, and decide which of your structures can be key-protected. You might decide that your internal data objects can be partitioned into multiple classes, according to the internal subsystems that reference them, and use more than one private key to achieve this.
2. Consider that data allocated for you by special services might require you to hold specific keys.
3. Construct hardware keysets as necessary for your protection gates.
4. Consider using read-only access rights for extra protection. For example, you might switch to read-only access for private data being made available to an untrusted function.
5. Allocate one or more private kernel keys to protect your private data.
6. Construct a heap (preferably, or use another method for allocating storage) protected by each kernel key you allocate, and substitute this heap (or heaps) consistently in your existing xmalloc and xmfree calls.
   When substituting, pay particular attention that you replace use of kernel_heap with a pageable heap, and the use of pinned_heap with a pinned one. Also be careful to always free allocated storage back to the heap from which it was allocated. You can use malloc and free as a shorthand for xmalloc and xmfree from the kernel_heap, so be sure to also check for these.
7. Understand the key requirements of the services you call. Some services might only work if you pass them public data.

You need to collect individual global variables into a single structure that can be xmalloced with key protection. Only the pointer to the structure and the hardware keyset necessary to access the structure need to remain public.

The private key or keys you allocate share hardware keys with other kernel keys, and perhaps even with each other. This affects the granularity of protection that you can achieve, but it does not affect how you design and write your code. So, write your code with only its kernel keys in mind, and do not concern yourself with mapping kernel keys to hardware keys for testing purposes. Using multiple keys requires additional protection gates, which might not be justifiable in performance-sensitive areas.

3.7.5 Protection gates

The mechanism described in 3.7.4, “Degrees of storage key protection and porting considerations” on page 84 that grants broad data access rights to a key-unsafe kernel extension is a type of protection gate called an *implicit protection gate*.

If you make a kernel extension key-aware, you must add explicit protection gates, typically at all entry and exit points of your module. The gates exist to ensure that your module has access to the data it requires, and does not have access to data it does not require.

Without a gate at (or near) an entry point, code would run with whichever keys the caller happened to hold. This is something that should not be left to chance. Part of making a kernel

extension key-aware is determining the storage protection key requirements of its various entry points and controlling them with explicit protection gates.

There are two kinds of gates to choose from at an entry point:

Add gate This kind of gate allows you to augment the caller's keyset with your own. Choose an add gate for a service where your caller passes pointers to data that could be in an arbitrary key.

Because this data might be protected with a key that is not known to you, it is important that you retain the caller's keys to ensure you can reference the data, while perhaps adding additional keys so that you can also reference any private data that you need.

Replace gate With this kind of gate, you switch to your own defined set of keys. Choose a replace gate at an entry point that stands on its own (typically a callback that you have registered with the device switch table, for example). Your parameters are implicit, and you need to pick up the known keys that are necessary to reference this data.

The replace gate is also important in relinquishing the caller's keys; typically the kernel is your caller in these situations, and retaining access to kernel internal data would be inappropriate. The predefined kernel keysets previously described should form the basis of the typical replace gate.

In both cases, the protection gate service returns the original AMR value, so you can restore it at your exit points.

If a directly called service is not passed any parameters pointing to data that might be in an arbitrary key, a replace gate should be used in preference to an add gate, because it is a stronger form of protection. Generally, calls within your module do not need gates, unless you want to change protection domains within your module as part of a multiple keys component design.

Protection gates might be placed anywhere in your program flow, but it is often simplest to identify and place gates at all the externally visible entry points into your module. However, there is one common exception: you can defer the gate briefly while taking advantage of your caller's keys to copy potentially private data being passed into public storage, and then switch to your own keyset with a replace gate.

This technique yields stronger storage protection than the simpler add gate at the entry point. When using this approach, you must restore the caller's keys before public data can be copied back through a parameter pointer. If you need both the caller's keys and your own keys simultaneously, you *must* use an add gate.

To identify the entry points of your kernel extension, be sure to consider the following typical entry points:

- Device switch table callbacks, such as:
  - open
  - close
  - read
  - write
  - ioctl
  - strategy
  - select
  - print

  - dump
  - mpx
  - revoke

  The config entry point for a device driver typically does not have a protection gate, but it includes the initialization necessary for protection gates, heaps, and so on for subsequent use by other entry points. The entry point configuring device instances would typically have protection gates.

- Struct ndd callbacks used in network drivers, such as:
  - ndd_open
  - ndd_close
  - ndd_output
  - ndd_ctl
  - nd_receive
  - nd_status
  - ndd_trace
- Trb (timer event) handler
- Watchdog handler
- Enhanced I/O error handling (EER) handlers
- Interrupt handler (INTCLASSx, iodone, and offlevel)
- Environmental and power warning (EPOW) handler
- Exported system calls
- Dynamic reconfiguration (DR) and high-availability (HA) handlers
- Shutdown notification handler
- RAS callback
- Dump callback (for example, as set up using dmp_add or dmp_ctl(DMPCTL_ADD,…))
- Streams callback functions
- Process state change notification (proch) handlers
- Function pointers passed outside of your module

You generally need protection gates only to set up access rights to non-parameter private data that your function references. It is the responsibility of called programs to ensure the ability to reference any of their own private data. When your caller's access rights are known to be sufficient, protection gates are not needed.

Memory allocation

Issues related to heap changes and associated APIs are beyond the scope of this document. For information about this topic, refer to the AIX Information Center for 6.1 at the following site:

http://publib.boulder.ibm.com/infocenter/pseries/v6r1/index.jsp

Initial Authority Mask Register (AMR) value

When a key-safe kernel extension’s configuration entry point is called, only the KKEY_PUBLIC kernel key is active. This is set by the sysconfig() system call when calling an module config entry point or a registered dd_config entry point. It is the first call to a module’s config entry point that initializes hardware keysets for the module’s other entry points.

A configuration entry point can execute an explicit protection gate after its keysets are initialized. Future calls to other extension interfaces use protection gates (if the developer

Archived

added a protection gate for the interface) and the hardware keyset established by module configuration. Future calls to a configuration entry point can execute an explicit protection gate, but this requires logic in the module configuration point to differentiate the first and subsequent calls.

When a new (key-safe) thread/k-proc is created, it starts execution at an initialization function passed to kthread_start()/initp(). For key-safe extensions, the kernel calls this entry point with a keyset that contains only the KKEY_PUBLIC kernel key. The k-thread/k-proc initialization function is an entry point, so it is the callee's responsibility to add a protection gate if another keyset is required.

In these two cases, the kernel is required to load the legacy keyset before calling an initialization function contained in a key-unsafe extension.

Multiple page sizes

Significant work has been done to exploit medium (64 K) and large (16 M) pages in the kernel. Medium pages continue to be usable with kernel keys. Kernel heaps can continue to be backed by medium-size pages when kernel keys are enabled. There will be a heap per hardware key, and that will increase the kernel's working set. Code setting storage keys on all kernel memory must be aware of the page size.

3.7.6 Example using kernel keys

This example tries to load a kernel extension that has the ability to use kernel protection keys. It will provide a system call kkey_test() that will be called by user program myprog.c (shown in Example 3-24 on page 93). The make file for this program is shown in Example 3-21.

> **Note:** The kernel must be built using `bosboot -aD` to include the kernel debugger. Without this, you will not see the kernel printfs, and the dsi will not pop you into kdb, but will just take a dump.

When system call kkey_test() is called with parameter=0, it tries to access private heap with KKEY_VMM in its protection gate (as shown in Example 3-28 on page 96).

When system call kkey_test() is called with parameter>0, it tries to access private heap without KKEY_VMM in its protection gate (as shown in Example 3-29 on page 96).

Example 3-21 Make file for kernel key example

```
CC=/usr/vac/bin/cc
LD=/usr/bin/ld
LIB= -bI:kkey_set.exp
UTIL=.

all: myprog service kkey_set64

kkey_set64: kkey_set.c kkey_set.exp
   $(CC) -q64 -D_KERNEL -D_KERNSYS -D_64BIT_KERNEL -D__64BIT__ -o kkey_set64.o -c
kkey_set.c
   $(LD) -b64 -o kkey_set64 kkey_set64.o -e kkey_test_init -bE:kkey_set.exp
-bI:/usr/lib/kernex.exp  -lcsys

service: service.c
   $(CC) -o service service.c
```

```
myprog: myprog.c
   $(CC) -q64 -o myprog myprog.c $(LIB)
clean:
   rm -f *.o myprog service kkey_set64
```

The kernel extension will create a private heap protected by key KKEY_VMM. Then kernel extension will try to access it with and without KKEY_VMM in its keyset.

When trying access without KKEY_VMM, it should cause a DSI with exception EXCEPT_SKEY; see Example 3-22.

Example 3-22 Kernel extension code, system call is kkey_test(): kkey_set.c

```
#include <sys/types.h>
#include <sys/err_rec.h>
#include <sys/malloc.h>
#include <sys/libcsys.h>
#include <sys/kerrno.h>
#include <sys/skeys.h>
#include <sys/vmuser.h>

kkey_test_init(int cmd, struct uio * uio)
{
   printf("Kernel Extension kkey_set loaded successfully\n");
   return 0;
}

int kkey_test(int mode)
{
   kerrno_t rc;
   kkey_t kkey=KKEY_VMM;
   heapattr_t heapattr;
   heapaddr_t my_heap=HPA_INVALID_HEAP;
      kkeyset_t myset=KKEYSET_INVALID;
   hkeyset_t hwset, hwset1, oldhwset, oldhwset1;
   printf("\n");
   if ((rc=kkeyset_create( &myset))!=0){
      printf("kkeyset_create() failed\n");
      return -1;
   }
   hwset1=hkeyset_get();
   printf("Current hardware keyset       =%016lx\n",hwset1);
   /*
    * Add keyset KKEYSET_KERNEXT to our keyset.
    * Remember KKEYSET_KERNEXT= {KKEY_PUBLIC, KKEY_BLOCK_DEV, KKEY_COMMO,
    * KKEY_USB, KKEY_GRPAPHICS, KKEY_FILESYSTEM, KKEY_DMA, KKEY_TRB, KKEY_MBUF,
    * KKEY_IOMAP}
    */
   if ((rc=kkeyset_add_set(myset, KKEYSET_KERNEXT))!=0){
      printf("kkseyset_add_set() failed\n");
      return -1;
   }

   if ((rc=kkeyset_to_hkeyset(myset, &hwset))!=0){
      printf("kkeyset_to_hkeyset() failed: rc=%lx\n",rc);
      return -1;
```

Archived

```
}
printf("hardware keyset after KERNEXT =%016lx\n",hwset);
/*
 * Add kkey=KKEY_VMM to the current keyset. This is the key we will be
 * using to protect our memory pages
 */
if ((rc=kkeyset_add_key(myset, kkey ,KA_RW))!=0){
   printf("kkeyset_add_key() failed\n");
   return -1;
}

if ((rc=kkeyset_to_hkeyset(myset, &hwset))!=0){
   printf("kkeyset_tohkeyset() failed: rc=%lx\n",rc);
   return -1;
}
printf("hardware keyset after KKEY_VMM=%016lx\n",hwset);
/*
 * Create a heap protected by the key KKEY_VMM
 */
bzero(&heapattr, sizeof(heapattr));
heapattr.hpa_eyec=EYEC_HEAPATTR;
heapattr.hpa_version=HPA_VERSION;
heapattr.hpa_flags=HPA_PINNED|HPA_SHARED;
heapattr.hpa_debug_level=HPA_DEFAULT_DEBUG;
/*
 * The heap will be protected by key==heapattr.hpa_kkey=KKEY_VMM
 * So other extensions/components should have KKEY_VMM in their keyset in
 * order to access it.
 */
heapattr.hpa_kkey=kkey;
if ((rc=heap_create(&heapattr, &my_heap))!=0 ){
   printf("heap_create() failed\n");
   return -1;

}
/*
 * Add current keyset={KKEYSET_KERNEXT, KKEY_VMM} to the current kernel
 * extension/system-call. This will be done through the help of a Protection
 * Gate. If you dont do this you will not be able to access the private heap
 * created in line#75 as thats protected by key KKEY_VMM
 */
oldhwset=hkeyset_replace(hwset);
/*
 * Assign a page from our private kernel heap which is protected by
 * keyset={KKEYSET_KERNEXT, KKEY_VMM}.
 */
caddr_t page=xmalloc(4096, 12, my_heap);
if (page==NULL){
   printf("xmalloc() failed");
   return -1;
}
/* Test KA_READ access on heap. Since we have the key in our keyset (from
 * line#52) so we will be able to access it. In case we haven't it would have
 * caused a DSI or machine crash.
 */
```

Archived

```
   *(page+10)=10;
   if (mode>0){
   /*
    * Remove KKEY_VMM from current keyset. After this operation current keysey=
    * {KKEYSET_KERNEXT}
    */
      if ((rc=kkeyset_remove_key(myset, KKEY_VMM, KA_RW))!=0){
         printf("kkeyset_remove_key() failed\n");
         return -1;
      }
      if ((rc=kkeyset_to_hkeyset(myset, &hwset))!=0){
         printf("kkeyset_to_hkeyset() failed: rc=%lx\n",rc);
         return -1;
      }
   /*
    * Set current keyset= {KKEYSET_KERNEXT} using the Protection Gate to current
    * kernel code path
    */
      oldhwset1=hkeyset_replace(hwset);
      printf("hardware keyset w/o KKEY_VMM  =%lx\n",hwset);
   /*
    * Try accessing the private heap using the current keyset={KKEYSET_KERNEXT}.
    * As the heap is protected by {KKEY_VMM} so this should
    * result in a DSI or system crash (as there is no default exception handler
    * defined for storage key exception- EXCEPT_SKEY
    */
      *(page+10)=10;
      printf("never reaches here\n");
   }
   /*
    * Restore the hardware keyset to the value before module entry
    */
   hkeyset_restore(hwset1);
   /*
    * Free the space occupied by data structures for kernel keyset
    */
   xmfree(page, my_heap);
   heap_destroy (my_heap,0);
   kkeyset_delete(myset);
   return 0;
}
```

The export file for kernel extension is shown in Example 3-23.

Example 3-23 Export file for kernel extension:kkey_set.exp

```
#!/unix
kkey_test syscall64
```

Sample code myprog.c is shown in Example 3-24.

Example 3-24 User program for accessing kernel extension: myprog.c

```
main(int argc, char** argv)
{
   int param=0;
```

```
   param=atoi(*(argv+1));
   printf("Testing Kernel Storage key\n");
   /*
    * If argument passed is > 0 then Kernel extension
    * will perform accessing a page for which it doesnt
    * have a key. This will result in a DSI and hence
    * machine will crash.
    * Else kernel extension will access the page with
    * required storage key and everything will go fine.
    */
   return (kkey_test(param));
}
```

Sample code for a tool used for loading and unloading the kernel extension is shown in Example 3-25.

Example 3-25 Tool for loading and unloading kernel extension: service.c

```
#include <sys/types.h>
#include <sys/sysconfig.h>
#include <errno.h>

#define BINSIZE 256
#define LIBSIZE 256

#define ADD_LIBPATH()                     \
      strcpy(g_binpath,*(argv+2));      \
      g_cfg_load.path=g_binpath;        \
      if (argc<4)                       \
         g_cfg_load.libpath=NULL;      \
      else{                             \
         strcpy(g_libpath,*(argv+3)); \
         g_cfg_load.libpath=g_libpath;\
      }                                 \
      g_cfg_load.kmid = 0;

struct cfg_load g_cfg_load;
char g_binpath[BINSIZE];
char g_libpath[LIBSIZE];

main(int argc, char** argv)
{
   if (argc<3){
      printf("Usage: service --load|--unload Kernel_Entension [Libpath]\n");
      return 1;
   }

   if (!(strcmp("--load",*(argv+1)))){
      ADD_LIBPATH();
   /*
    * Load the kernel extension
    */
      if (sysconfig(SYS_SINGLELOAD, &g_cfg_load, sizeof(struct cfg_load))==\\
         CONF_SUCC){
         printf("[%s] loaded successfully kernel \\
                mid=[%d]\n",g_binpath,g_cfg_load.kmid);
```

```
            return 0;
         }else{
            printf("[%s] not loaded with ERROR=%d\n",g_binpath,errno);
            return 1;
         }
      }
      if (!(strcmp("--unload",*(argv+1)))){
         ADD_LIBPATH();
         if (sysconfig(SYS_QUERYLOAD, &g_cfg_load, sizeof(struct \\
              cfg_load))||g_cfg_load.kmid){
            printf("[%s] found with mid=[%d]\n",g_binpath,g_cfg_load.kmid);
      /*
       * Unloaded the loaded kernel extension
       */
            if (sysconfig(SYS_KULOAD, &g_cfg_load, sizeof(struct cfg_load))== 0){
               printf("[%s] unloaded successfully\n",g_binpath);
               return 0;
            }else{
               printf("[%s] not unloaded with error=[%d]\n",g_binpath,errno);
               return 1;
            }
         }else{
            printf("[%s] currently not loaded, nothing to unload\n",g_binpath);
            return 1;
         }
      }
      /*
       * Invalid second Argument
       */
      printf("Usage: service --load|--unload Kernel_Entension [Libpath]\n");
      return 1;
   }
```

Try to load the kernel extension and call it first with a protection gate, and then without an appropriate protection gate (that is, without KKEY_VMM) by using KKEY_VMM.

Note: The second iteration will cause the kernel to crash.

To view the output of the kernel extension (system call), you need console access. All output will go to the hardware console when you use printf() in kernel mode. Compile the program using the command shown in Example 3-26.

Example 3-26 Compile the program

```
# make
        /usr/vac/bin/cc -q64 -o myprog myprog.c -bI:kkey_set.exp
        /usr/vac/bin/cc -o service service.c
        /usr/vac/bin/cc -q64 -D_KERNEL -D_KERNSYS -D_64BIT_KERNEL -D__64BIT__ -o
kkey_set64.o -c kkey_set.c
"kkey_set.c": 1506-312 (W) Compiler internal name __64BIT__ has been defined as a
macro.
        ld -b64 -o kkey_set64 kkey_set64.o -e kkey_test_init -bE:kkey_set.exp
-bI:/usr/lib/kernex.exp  -lcsys
Target "all" is up to date.
```

Archived

Load the kernel extension so that a system call can use it form user mode, as shown in Example 3-27.

Example 3-27 Loading kernel extension using tool "service"

```
# ./service --load kkey_set64
Preserving 2378 bytes of symbol table [kkey_set64]
[kkey_set64] loaded successfully kernel mid=[546676736]
```

Run myprog without any argument, which means it will run the kernel extension (syscall) with the appropriate protection gate {{KKEYSET_KERNEXT}+KKEY_VMM}. Because the private heap of the kernel extension has been protected by KKEY_VMM, you can access it and the system call will return without any DSI/crash.

Execute the user-level program that will call the kernel system call with protection gate enabled with KKEY_VMM, as shown in Example 3-28.

Example 3-28 First iteration with KKEY_VMM

```
#./myprog
Testing Kernel Storage key

Current hardware keyset       =0008000000000000
hardware keyset after KERNEXT =F00F000000000000
hardware keyset after KKEY_VMM=F00C000000000000
```

Now execute the user-level program that will call the kernel system call with a protection gate without KKEY_VMM key in its keyset (see Example 3-29 on page 96). This will cause the kernel to crash. Running myprog with argument>2 will do that.

Because the private heap of the kernel extension has not been protected by KKEY_VMM, a DSI will result and the kernel will crash. Keep in mind that exception type EXCEPT_SKEY cannot be caught with setjmpx(), so the kernel programmer cannot catch EXCEPT_SKEY using setjmpx().

> **Note:** There are kernel services provided for catching storage-key exceptions. However, their use is *not* recommended and they are provided only for testing purposes.

Example 3-29 Second iteration without KKEY_VMM

```
# ./myprog 1

Current hardware keyset       =0008000000000000
hardware keyset after KERNEXT =F00F000000000000
hardware keyset after KKEY_VMM=F00C000000000000
hardware keyset w/o KKEY_VMM  =F00F000000000000
Data Storage Interrupt - PROC
.kkey_test+0002C8       stb    r3,A(r4)
r3=000000000000000A,A(r4)=F10006C00134A00A
KDB(6)>
```

When you are debugging in kernel mode (see Example 3-30), you can see the value of the AMR (current hardware key or protection gate), the current process which caused the exception, and the exception type. Refer to 3.7.8, "Kernel debugger commands" on page 108

for more information about KDB commands relevant for debugging storage-key exceptions and related issues.

Example 3-30 KDB debugging

```
KDB(6)> f
pvthread+80B100 STACK:
[F1000000908E34C8]kkey_test+0002C8 (0000000100000001)
[00014D70].hkey_legacy_gate+00004C ()
[00003820]ovlya_addr_sc_flih_main+000100 ()
[kdb_get_virtual_doubleword] no real storage @ FFFFFFFF3FFFE60

KDB(6)> dr amr
amr   : F00F000000000000
  hkey  2  RW  PUBLIC
  hkey  3  RW  BLOCK_DEV LVM RAMDISK FILE_SYSTEM NFS CACHEFS AUTOFS KRB5 ...
  hkey  4  RW  COMMO NETM IPSEC USB GRAPHICS
  hkey  5  RW  DMA PCI VDEV TRB IOMAP PRIVATE1 PRIVATE17 PRIVATE9 PRIVATE25 ...

KDB(6)> dk 1 @r4
Protection for sid 4000006000000, pno 00C00134, raddr 81C7A000
  Hardware Storage Key...  7
  Page Protect key.......  0
  No-Execute.............  0

KDB(6)> p
              SLOT NAME     STATE      PID    PPID          ADSPACE  CL #THS

pvproc+100AC00 16427*myprog   ACTIVE 002B186 00301C8 00000000A9857590   0 0001

NAME....... myprog
STATE...... stat  :07  .... xstat :0000
FLAGS...... flag  :00200001 LOAD EXECED
........... flag2 :00000001 64BIT
........... flag3 :00000000
........... atomic :00000000
........... secflag:0001 ROOT
LINKS...... child      :0000000000000000
........... siblings   :0000000000000000
........... uidinfo    :000000000256BA78
........... ganchor    :F10008070100AC00 <pvproc+100AC00>
THREAD..... threadlist :F10008071080B100 <pvthread+80B100>
DISPATCH... synch      :FFFFFFFFFFFFFFFF
AACCT...... projid      :00000000  ........... sprojid     :00000000
........... subproj     :0000000000000000
........... file id     :0000000000000000 0000000000000000 00000000
........... kcid       :00000000
........... flags       :0000
(6)> more (^C to quit) ?

KDB(6)> mst
Machine State Save Area
iar   : F1000000908E34C8  msr   : 8000000000009032  cr    : 80243422
lr    : F1000000908E34BC  ctr   : 00000000007CD2A0  xer   : 20000001
mq    : FFFFFFFF  asr   : FFFFFFFFFFFFFFFF  amr   : F00F000000000000
r0  : F1000000908E34BC  r1  : F00000002FF47480  r2  : F1000000908E4228
```

```
r3  : 000000000000000A  r4  : F10006C00134A000  r5  : F00000002FF47600
r6  : 8000000000009032  r7  : 8000000000009032  r8  : 0000000000000000
r9  : 00000000020BB1C0  r10 : F00F000000000000  r11 : F00000002FF47600
r12 : F00000002FFCCFE0  r13 : F100060801392400  r14 : 0000000000000002
r15 : 0FFFFFFFFFFFF418  r16 : 0FFFFFFFFFFFF430  r17 : 0800200140000000
r18 : 0FFFFFFFFFFFFED0  r19 : 09FFFFFFF000B940  r20 : 0000000110922550
r21 : 0000000000000000  r22 : 0000000000002AF3  r23 : 0000000110008158
r24 : 0000000110008158  r25 : 0000000000000000  r26 : 0000000110009778
r27 : 0000000110009768  r28 : FFFFFFFFFFFFFFFF  r29 : 0000000110009758
r30 : F1000000908E3788  r31 : F1000000908E4000

prev      0000000000000000 stackfix  0000000000000000 int_ticks 0000
cfar      0000000000014E74
kjmpbuf   0000000000000000 excbranch 0000000000000000 no_pfault 00
intpri    0B               backt     00               flags     00
hw_fru_id 00000001         hw_cpu_id 00000003
(6)> more (^C to quit) ?
fpscr     0000000000000000 fpscrx    00000000         fpowner   01
fpeu      01               fpinfo    00               alloc     F000
o_iar     F1000000908E34C8 o_toc     F1000000908E4228
o_arg1    000000000000000A o_vaddr   F10006C00134A00A
krlockp   0000000000000000 rmgrwa    F100041580132E20
amrstackhigh  F00000002FFCCFF0 amrstacklow   F00000002FFCC000
amrstackcur   F00000002FFCCFE0 amrstackfix   0000000000000000
kstackhigh    0000000000000000 kstacksize    00000000
frrstart  700DFEED00000000 frrend    700DFEED00000000
frrcur    700DFEED00000000 frrstatic 0000 kjmpfrroff 0000
frrovcnt  0000 frrbarrcnt 0000 frrmask 00 callrmgr 00
Except :
excp_type 0000010E  EXCEPT_SKEY
 orgea F10006C00134A00A dsisr 0000000002200000  bit set: DSISR_ST DSISR_SKEY
 vmh   0000000006C00510 curea F10006C00134A00A pftyp 4000000000000106
KDB(6)> vmlog
Most recent VMM errorlog entry
Error id              =  DSI_PROC
Exception DSISR/ISISR  =  0000000002200000
Exception srval       =  0000000006C00510
Exception virt addr    =  F10006C00134A00A
Exception value       =  0000010E  EXCEPT_SKEY
KDB(0)>
```

Kernel key API

The following basic types represent kernel keys, kernel keysets, and hardware keysets. They are declared in <sys/skeys.h>.

kkey_t (uint)
: This holds the value of a single kernel key. This does not contain information about read/write access.

kkeyset_t (struct kkeyset *)
: This identifies a kernel key group with read/write access flags for each key in the set. This data structure is pageable, so it can only be referenced in the process environment.

hkeyset_t (unsigned long long)
: This is derived from a kernel keyset. It contains the hardware keyset

that allows access to the kernel keyset it was derived from. It is implemented as a raw AMR value.

The following APIs are available for managing kernel and hardware keysets:

kerrno_t kkeyset_create(kkeyset_t *set)
This creates a kernel keyset.

kerrno_t kkeyset_delete(kkeyset_t set)
This deletes a kernel keyset.

kerrno_t kkeyset_add_key(kkeyset_t set, kkey_t key, unsigned long flags)
This adds a kernel key to a kernel keyset.

kerrno_t kkeyset_add_set(kkeyset_t set, kkeyset_t addset)
This adds a kernel keyset to an existing kernel keyset.

kerrno_t kkeyset_remove_key(kkeyset_t set, kkey_t key, unsigned long flags)
This removes a kernel key- from an existing kernel keyset.

kerrno_t kkeyset_remove_keyset(kkeyset_t set, kkeyset_t removeset)
This removes members of one kernel keyset from an existing kernel keyset.

kerrno_t kkeyset_to_hkeyset(kkeyset_t kkeyset, hkeyset_t *hkey)
This computes the hardware key (AMR value) that provides memory access specified by the inputted kernel keyset.

hkeyset_t hkeyset_add(hkeyset_t keyset)
This updates the protection domain by adding the hardware-keyset specified by keyset to the currently addressable hardware-keyset. The previous hardware-keyset is returned.

hkeyset_t hkeyset_replace(hkeyset_t keyset)
This updates the protection domain by loading the set specified by keyset as the currently addressable storage set. The previous hardware-keyset is returned.

void hkeyset_restore(hkeyset_t keyset)
This updates the protection domain by loading the set specified by keyset as the currently addressable storage set. No return value is provided by this function. Because this service is slightly more efficient than hkeyset_replace(), it can be used to load a hardware-keyset when the previous keyset does not need to be restored.

hkeyset_t hkeyset_get()
This reads the current hardware-key-set without altering it.

Hardware keysets can also be statically assigned several predefined values. This is often useful to deal with use of a hardware keyset before a component can initialize it.

HKEYSET_INVALID
This keyset is invalid. When used, it will cause a storage-key exception on the next data reference.

HKEYSET_GLOBAL
This keyset allows access to all kernel keys. It is implemented as an all zeroes AMR value.

3.7.7 User mode protection keys

User keys are intended for providing heuristic data protection in order to detect and prevent accidental overlays of memory in an application. User key support is primarily being provided as a RAS feature which could be used, for example, for DB2. Their primary use is to protect the DB2 core from errors in UDFs (user-defined functions). Their secondary use is as a debugging tool to prevent and diagnose internal memory overlay errors.

To understand how a user key is useful, take a brief look at DB2. DB2 provides a UDF facility where customers can add extra code to the database. There are two modes UDFs can run in: fenced and unfenced, as explained here:

- Fenced mode

  UDFs are isolated from the database by execution under a separate process. Shared memory is used to communicate between the database and UDF process. Fenced mode has a significant performance penalty because a context switch is required to execute the UDF.

- Unfenced mode

  The UDF is loaded directly into the DB2 address space. Unfenced mode greatly improves performance, but introduces a significant RAS exposure.

Although DB2 recommends using fenced mode, many customers use unfenced mode for performance reasons. It is expected that only “private” user data can be protected with user keys. There are still exposures in system data, such as library data that is likely to remain unprotected.

Hardware keys are separated into a user mode pool and a kernel mode pool for several reasons. First, an important feature of kernel keys is to prevent accidental kernel references to user data. If the same hardware key is used for both kernel and user data, then kernel components that run with that hardware key can store to user space. This is avoided.

Separating the hardware keys simplifies user memory access services such as copyin(). Because the hardware keys are separated, the settings for kernel mode access and user mode access can be contained in a single AMR. This avoids a costly requirement to alter the AMR between source and destination buffer accesses.

When user keys are disabled, sysconf(_SC_AIX_UKEYS) returns zero (0), indicating that the feature is not available. Applications that discover the feature is not available should not call other user key-related services. These services fail if called when user keys are disabled.

Support for user keys is provided for both 32-bit and 64-bit APIs. In 32-bit mode, compiler support for long long is required to use user keys. User key APIs will be provided in an AIX V5.3 update, as well as in AIX V6.1.

Applications that use these APIs have an operating system load time requisite. To avoid this requisite, it is recommended that the application conditionally load a wrapper module that makes reference to the new APIs. The wrapper module is only loaded when it is determined that user keys are available; that is, when the configured number of user keys is greater than zero.

Types of user keys

AIX will provide a default number of user keys. A sysconf (_SC_AIX_UKEYS) call can be used by applications to query the number of user keys. For POWER6, two user keys are available.

Archived

This is accomplished by exclusively dedicating hardware keys for use by applications. The primary user key UKEY_PUBLIC is the default storage-key for user data. Access to this key cannot be disabled in user-mode.

The UKEY values are an abstraction of storage keys. These key values are the same across all applications. For example, if one process sets a shared page to UKEY_PRIVATE1, all processes need UKEY_PRIVATE1 authority to access that page.

The sysconf() service can be used to determine if user keys are available without load time dependencies. Applications must use ukey_enable() to enable user keys before user key APIs can be used. All user memory pages are initialized to be in UKEY_PUBLIC. Applications have the option to alter the user key for specific data pages that should not be publicly accessible. User keys may not be altered on mapped files. The application must have write authority to shared memory to alter the user key.

Other considerations for user keys application programming

Like kernel key support, user key support is intended as a Reliability and Serviceability feature. It is *not* intended to provide a security function. A system call permits applications to modify the AMR to allow and disallow access to user data regions, with no authority checking.

The kernel manages its own AMR when user keys are in use. When the kernel performs loads or stores on behalf of an application, it respects the user mode AMR that was active when the request was initiated. The user key values are shared among threads in a multithreaded process, but a user mode AMR is maintained per thread. Kernel context switches preserve the AMR. Threaded applications are prevented from running M:N mode with user keys enabled by the ukey_enable() system call and pthread_create().

The user mode AMR is inherited by fork(), and it is reset to its default by exec(). The default user mode value enables only UKEY_PUBLIC (read and write access). A system call, ukeyset_activate() is available to modify the user mode AMR. Applications cannot disable access to UKEY_PUBLIC. Preventing this key from being disabled allows memory that is “unknown” to an application to always be accessible. For example, the TOC or data used by an external key-unsafe library is normally set to UKEY_PUBLIC.

The ucontext_t structure is extended to allow the virtualized user mode AMR to be saved and restored. The sigcontext structure is not changed. The jmp_buf structure is not extended to contain an AMR, so callers of setjmp(), _setjmp(), and sig_setjmp() must perform explicit AMR management. A ukey_setjmp() API is provided that is a front-end to setjmp() and manages the user mode AMR.

The user mode AMR is reset to contain only UKEY_PUBLIC when signals are delivered and the interrupted AMR is saved in the ucontext_t structure. Signal handlers that access storage that is not mapped UKEY_PUBLIC are responsible for establishing their user mode AMR.

Hardware and operating system considerations

- AIX APIs for application user keys will be made available in AIX 5.3 TL6 running on POWER6 hardware. AIX V5.3 does not support kernel keys.
- In AIX V5.3 TL6, application interfaces that exploit user keys only function with the 64-bit kernel. When the 32-bit kernel is running, these user keys APIs fail.
- User keys are considered an optional platform feature. APIs are present to query if user keys are supported at runtime, and how many user keys are available.
- User keys are available in 32-bit and 64-bit user-mode APIs.
- User keys can be used by threaded applications in 1:1 mode. They are not supported in M:N mode.

- The APIs provided by the system are low level. They allow management of the AMR and page protection. Address space management is left to applications.
- When storage keys are available on a platform, a P6 system will default to two user keys available to applications. It is possible to reconfigure the number of user keys, but that requires an operating system reboot. Changing the default number of user keys is not expected to be a customer action. This is only done in a lab environment, or as an SFDC action by IBM service.

Note: User keys are provided as a RAS feature. They are only intended to prevent accidental accesses to memory and are *not* intended to provide data security.

Example of user keys for two processes using a shared memory object

To demonstrate how user keys can be used, we developed a short C program. This program creates two processes sharing a memory page, and that page will have a private key assigned to it.

The parent process will have UKEY_PRIVATE1=UK_RW access. The child process will have UKEY_PRIVATE1 with UK_READ, UK_WRITE and UK_RW access for each iteration of the program. We will see how the child process behaves when it is given access to a shared page with each authority. We also need to ensure that shared memory is allocated as a multiple of pagesize, because that is the granularity level for protection keys.

Table 3-5 lists the sequence of operations for child and parent processes with respect to time.

Table 3-5 Execution flow

Time (sec)	Parent	Child
0	Define shared segment with size=1 page; Define access UKEY_PRIVATE1=UK_RW; Protect with pkey=UKEY_PRIVATE1 sleep(2);	Sleeping
1	Sleeping	Attach to the shared segment created by Parent; Define access UKEY_PRIVATE2=UK_READ or UK_WRITE or UK_RW; Protect with ckey=UKEY_PRIVATE2 READ1(); WRITE1("abcd"); sleep(2);
2	Sleeping	Sleeping
3	READ1(); WRITE1("ABCD"); sleep(2);	Sleeping
4	Sleeping	READ1(); WRITE1("ABCD"); sleep(2);
5	Sleeping	Sleeping
6	Detach and remove shared page; return (0);	Sleeping

Archived

Time (sec)	Parent	Child
7	Sleeping	Detach shared page; return(0)

Example 3-31 Programming example of user keys

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/syspest.h>
#include <sys/signal.h>
#include <sys/vminfo.h>
#include <sys/ukeys.h>
key_t key;
int id;
int one_page;
char* data;
int pid;
int rc;
main(int argc, char** argv)
{

ukey_t pkey=UKEY_PRIVATE1;
ukey_t ckey=UKEY_PRIVATE1;
ukeyset_t pkeyset;
ukeyset_t ckeyset;
ukeyset_t oldset;
int nkeys;
if ((nkeys=ukey_enable())==-1){
   perror("main():ukey_enable(): USERKEY not enabled");
   exit(1);
}
assert(nkeys>=2);
if (rc=ukeyset_init(&pkeyset,0)){
   perror("main():ukeyset_init(pkeyset)");
   exit(1);
}
if (rc=ukeyset_init(&ckeyset,0)){
   perror("main():ukeyset_init(ckeyset)");
   exit(1);
}
if (rc=ukeyset_add_key(&pkeyset, pkey, UK_RW)){
   perror("main():ukeyset_add_key(pkeyset, pkey,UK_RW)");
   exit(1);
}

if (!strcmp(*(argv+1),"write")){
   if (rc=ukeyset_add_key(&ckeyset, ckey, UK_WRITE)){
      perror("main():ukeyset_add_key(ckeyset, ckey,UK_WRITE)");
      exit(1);
   }
}else{
```

```
    if (!strcmp(*(argv+1),"read")){
       if (rc=ukeyset_add_key(&ckeyset, ckey, UK_READ)){
          perror("main():ukeyset_add_key(ckeyset, ckey,UK_READ)");
          exit(1);
       }
    }else{
       if (rc=ukeyset_add_key(&ckeyset, ckey, UK_RW)){
          perror("main():ukeyset_add_key(ckeyset, ckey,UK_RW)");
          exit(1);
       }
    }
 }
 key=ftok("./ukey1.c",5);
 pid=fork();
 switch(pid){
 case (0):/* CHILD */
    sleep (1);
    if ((id=shmget(key, 0, 0))==-1){
       perror("child :shmget()");
       return (1);
    }
    if ( (data=shmat(id,(void *)0, 0)) == (char*) (-1) ){
       perror("child :shmat()");
       return (1);
    }
    printf("child:data=0x%x\n",data);
    if (rc=ukey_protect(data,one_page,ckey)){
       perror("child :ukey_protect(ckey)");
       exit(1);
    }
    oldset=ukeyset_activate(ckeyset,UKA_ADD_KEYS);
    if (oldset==UKSET_INVALID){
       printf("child :ukeyset_activate() failed");
       exit (1);
    }
    printf("child :READ1 =[%s]\n",data);
    strcpy(data+strlen(data),"abcd");
    printf("child :WRITE1=[abcd]\n");
    sleep(2);
    printf("child :READ2 =[%s]\n",data);
    strcpy(data+strlen(data),"efgh");
    printf("child :WRITE2=[efgh]\n");
    sleep(2);
    if (shmdt(data)==-1){
       perror("child :shmdt()");
       return (1);
    }
    return (0);
 case (-1):/* ERROR */
    perror("fork()");
    return (1);
 default:/* PARENT */
    one_page=4096;
    printf("parent:pagesize=%d\n", one_page);
    if ((id=shmget(key, one_page, IPC_CREAT|0644))==-1){
```

```
      perror("parent:shmget()");
      return (1);
   }
   if ( (data=shmat(id,(void *)0, 0)) == (char*) (-1) ){
      perror("parent:shmat()");
      return (1);
   }
   if (rc=ukey_protect(data,one_page,pkey)){
      perror("parent:ukey_protect(pkey)");
      exit(1);
   }
   oldset=ukeyset_activate(pkeyset,UKA_ADD_KEYS);
   if (oldset==UKSET_INVALID){
      printf("parent:ukeyset_activate() failed");
      exit (1);
   }
   sleep(2);
   printf("parent:READ1 =[%s]\n",data);
   strcpy(data+strlen(data),"ABCD");
   printf("parent:WRITE1=[ABCD]\n");
   sleep(2);
   if (shmdt(data)==-1){
      perror("parent:shmdt()");
      return (1);
   }
   if (shmctl(id,IPC_RMID,0)){
      perror("parent:shmctl()");
      return (1);
   }
   return (0);
}
}
```

We compiled the program:

```
# cc -g -o ukey1 ukey1.c
```

We gave the child process write (no-read) access on the shared memory segment, and executed it as shown in Example 3-32.

Example 3-32 Executing program with write access to shared memory

```
# ./ukey1 write
parent:pagesize=4096
child:data=0x30000000
parent:READ1 =[]
parent:WRITE1=[ABCD]
# ls core
core
# dbx ukey1 core
Type 'help' for help.
[using memory image in core]
reading symbolic information ...
Segmentation fault in strlen at 0xd010da00
0xd010da00 (strlen)    89030000         lbz   r8,0x0(r3)
(dbx) where
```

```
strlen() at 0xd010da00
_doprnt(??, ??, ??) at 0xd0126350
printf(0x200003f4, 0x30000000, 0xffffffff, 0x2ff47600, 0x0, 0x0, 0x0, 0x0) at
0xd011eabc
main(argc = 2, argv = 0x2ff22554), line 126 in "ukey1.c"
(dbx)q
#
```

Notice that a segmentation fault occurred at strlen(), which was trying to read shared memory and calculate its size. Because read permission was not provided, it caused SIGSEGV.

Next, we gave the child process read (no-write) access to the shared memory segment and executed the code as shown in Example 3-33.

Example 3-33 Program with read access to shared segment

```
# ./ukey1 read
parent:pagesize=4096
child:data=0x30000000
child :READ1 =[]
parent:READ1 =[]
parent:WRITE1=[ABCD]
# ls core
core
# dbx ukey1 core
Type 'help' for help.
[using memory image in core]
reading symbolic information ...

Segmentation fault in noname.strcpy [ukey1] at 0x10001004
0x10001004 (strcpy+0x4) 99030000        stb   r8,0x0(r3)
(dbx) where
noname.strcpy() at 0x10001004
main(argc = 2, argv = 0x2ff2255c), line 127 in "ukey1.c"
(dbx)q
#
```

Notice that a segmentation fault occurred at the strcpy() function, which was trying to write to shared memory. The strlen() function did not fail this time, because we gave read access to the shared page. However, the child process does not have write access to strcpy(), which caused SIGSEGV.

We executed the program by giving read and write access, as shown in Example 3-34 on page 106.

Example 3-34 Program gets read and write access to shared segment

```
# ./ukey1 readwrite
parent:pagesize=4096
child:data=0x30000000
child :READ1 =[]
child :WRITE1=[abcd]
parent:READ1 =[abcd]
parent:WRITE1=[ABCD]
child :READ2 =[abcdABCD]
child :WRITE2=[efgh]
```


Archived

```
# ls core
core not found
```

Because the child process was able to read and write the content of shared object, there is no core file at this time.

User key API

The following data structures and APIs are declared for the user key feature. They are declared in <sys/ukeys.h>. For a more detailed explanation of the APIs refer to the IBM white paper “Storage Protection Keys on AIX Version 5.3”, which can be downloaded from:

ftp://ftp.software.ibm.com/common/ssi/rep_wh/n/PSW03013USEN/PSW03013USEN.PDF

In AIX V6.1 documentation, the white paper is available at:

http://publib.boulder.ibm.com/infocenter/pseries/v6r1/index.jsp

ukey_t
This is a basic type for a user-mode storage protection key.

ukeyset_t
This is a user key set with read/write attributes for each key in the set.

sysconf(_SC_AIX_UKEYS)
This returns the number of user keys available.

int ukey_enable()
This grants a process access to user keys memory protection facilities. It returns the number of user keys configured on the system.

int ukey_getkey(void *addr, ukey_t *ukey)
This is used to determine the user key associated with a memory address.

int ukey_protect (void *addr, size_t len, ukey_t ukey)
This modifies the memory's user key protection for a given address range. This range must include only whole pages.

int ukeyset_init (ukeyset_t *nset, unsigned int flags)
This initializes a user key set with just UKEY_PUBLIC (unless UK_INIT_ADD_PRIVATE flag is specified). The value of “flags” may be UK_READ, UK_WRITE, or UK_RW.

int ukeyset_add_key (ukeyset_t *set, ukey_t key, unsigned int flags)
This adds a user key to a user key set.

int ukeyset_remove_key (ukeyset_t *set, ukey_t key, unsigned int flags)
This removes a user key from a user key set. The value of “flags” may be UK_READ, UK_WRITE, or UK_RW.

int ukeyset_ismember (ukeyset_t set, ukey_t key, unsigned int flags)
This determines if a key is part of a keyset. The value of “flags” may be UK_READ, UK_WRITE, or UK_RW.

int ukeyset_add_set (ukeyset_t *set, ukeyset_t aset)
This adds a user key set.

int ukeyset_remove_set (ukeyset_t *set, ukeyset_t rset)
This removes a user key set.

int pthread_attr_getukeyset_np (pthread_attr_t *attr, ukeyset_t *ukeyset)
This returns a thread attributes object's user key set attribute.

int pthread_attr_setukeyset_np (pthread_attr_t *attr, ukeyset_t *ukeyset)

This sets a thread attributes object's user key set attribute.

ukeyset_t ukeyset_activate (ukeyset_t set, int command)

This activates a user keyset and returns the current user keyset. The value of "command" can be one of the following:
UKA_REPLACE_KEYS: Replace keyset with the specified keyset.
UKA_ADD_KEYS: Add the specified keyset to the current keyset.
UKA_REMOVE_KEYS: Remove the specified keyset from the active keyset.
UKA_GET_KEYS: Read the current key value without updating the current keyset. The input keyset is ignored.

3.7.8 Kernel debugger commands

The kernel debugger (KDB) has some new and changed commands to help you with storage protection keys. These commands are listed in Table 3-6.

Table 3-6 New and changed kernel debugger commands

Kernel debugger command	Explanation
`kkeymap`	Displays the available hardware keys and the kernel keys that map to each.
`kkeymap <decimal kernel key number>`	Displays the mapping of the specified kernel key to a hardware key (-1 indicates that the kernel key is not mapped).
`hkeymap <decimal hardware key number>`	Displays all the kernel keys that map to the specified hardware key.
`kkeyset <address of kkeyset_t>`	Displays the kernel key access rights represented by a kernel keyset. The operand of this command is the address of the pointer to the opaque kernel keyset, not the kernel keyset structure itself.
`hkeyset <64 bit hex value>`	Displays the hardware key accesses represented by this value if used in the AMR, and a sampling of the kernel keys involved.
`dr amr`	Displays the current AMR and the access rights it represents.
`dr sp`	Includes the AMR value.
`mr amr`	Allows modification of the AMR.
`dk 1 <eaddr>`	Displays the hardware key of the resident virtual page containing eaddr.
`mst`	Displays the AMR and context stack values. A storage key protection exception is indicated in excp_type as DSISR_SKEY.
`iplcb`	Displays the IBM processor-storage-keys property of a CPU, which indicates the number of hardware keys supported. This is in section /cpus/PowerPC of the device tree.
`vmlog`	Shows an exception value of EXCEPT_SKEY for a storage key violation.
`pft`	Displays the hardware key (labeled hkey) value for the page.
`pte`	Displays the hardware key (labeled sk) value for the page.
`scb`	Displays the hardware key default set for the segment.

Archived

Kernel debugger command	Explanation
heap	Displays the kernel and hardware keys associated with an xmalloc heap.

In the following examples we show storage key features using the kernel debugger (**kdb**).

A full mapping of kernel-to-hardware keys is shown in Example 3-35.

Example 3-35 Kernel-to-hardware key mapping

```
(0)> kk
kkey hkey name(s)
  0    0   UPUBLIC
  1    1   UPRIVATE1
 32    0   FILE_DATA

 33    2   PUBLIC

 34    3   BLOCK_DEV LVM RAMDISK FILE_SYSTEM NFS CACHEFS AUTOFS KRB5 SWAPNFS
 36    4   COMMO NETM IPSEC USB GRAPHICS
 40    5   DMA PCI VDEV TRB IOMAP
 43    5   PRIVATE*

 75    6   VMM_PMAP IOS IPC RAS LDATA_ALLOC KRLOCK XMALLOC KKEYSET J2
 81    6   FILE_METADATA
 77    7   PROC INTR VMM LFS CRED LDR KER
```

Note: In Example 3-35, the kernel key number (kkey) is that of the first key in the row. There is no guarantee that the other keys in the row are numbered sequentially.

The name for kernel key 3:

```
(0)> kk 3
 KKEY_UPRIVATE3(3) -> hkey -1
```

The hardware key corresponding to VMM_PMAP:

```
(0)> kk VMM_PMAP
KKEY_VMM_PMAP(75) -> hkey 6
```

The kernel keys corresponding to hardware key 6:

```
(0)> hk 6
 VMM_PMAP IOS IPC RAS LDATA_ALLOC KRLOCK XMALLOC KKEYSET J2 FILE_METADATA
```

The hardware keys and corresponding kernel keys in current context:

```
(0)> dr amr
amr    : F3FC000000000000
  hkey  2  RW  PUBLIC
  hkey  7  RW  PROC INTR VMM LFS CRED LDR KER
```

Example 3-36 shows the Authority Mask Register (AMR) value for the current Machine Status Table (MST).

Example 3-36 Machine Status Table (MST)

```
(0)> mst
```

```
Machine State Save Area
iar   : 000000000044F1F8  msr   : 8000000000009032  cr    : 44028028
lr    : 00000000000544D0  ctr   : 8000000000F958A0  xer   : 00000000
mq    : 00000000  asr   : 000000025F20D001  amr   : F3FC000000000000
r0  : 0000000000000001  r1  : 0FFFFFFFF3FFFDF0  r2  : 0000000002D45C30
r3  : 0000000000000000  r4  : 0000000000000000  r5  : 000000000000003F
r6  : 0000000000000000  r7  : 0000000000000001  r8  : 0000000000000000
r9  : 0000000000000000  r10 : 00000000000000FF  r11 : 00000000000000FF
r12 : 0000000000000006  r13 : F100060800820400  r14 : 00000000DEADBEEF
r15 : 0000000000000002  r16 : 00000000000A1C50  r17 : 0000000000000000
r18 : 00000000020AF366  r19 : 00000000000034C8  r20 : 0000000002A2B580
r21 : 0000000000000000  r22 : F100080710800078  r23 : F100010007FA5800
r24 : 00000000020B7EF4  r25 : 0000000000000000  r26 : 0000000000000000
r27 : 0000000002570900  r28 : 00000000025708FE  r29 : 00000000020B7E70
r30 : F10006C001356000  r31 : 00000000020B71C0

prev      0000000000000000 stackfix  0000000000000000 int_ticks 0000
cfar      0000000000003708
kjmpbuf   0000000000000000 excbranch 0000000000000000 no_pfault 00
intpri    0B               backt     00               flags     00
hw_fru_id 00000001         hw_cpu_id 00000002

(0)> hks F3FC000000000000
  hkey  2  RW  PUBLIC
  hkey  7  RW  PROC INTR VMM LFS CRED LDR KER
(0)>
```

Example 3-37 shows the hardware keys for a page frame table (PFT) entry.

Example 3-37 Hardware keys for current PFT

```
(0)> pft 1
Enter the page frame number (in hex): 1000

VMM PFT Entry For Page Frame 0000001000 of 000027FFFF

ptex = 0000000000000800 pvt = F200800040010000 pft = F200800030060000
h/w hashed sid : 0000000000000  pno : 0000000100  psize : 64K  key : 3
s/w hashed sid : 0000000000000  pno : 0000000100  psize : 64K
s/w pp/noex/hkey : 3/0/02  wimg : 2

> in use
> on scb list
> valid (h/w)
> referenced (pft/pvt/pte): 0/0/0
> modified (pft/pvt/pte): 1/0/0
base psx.................. 01 (64K)  soft psx.................. 00 ( 4K)
owning vmpool............. 00000000  owning mempool............ 00000000
owning frameset........... 00000002
source page number............ 0100
dev index in PDT.............. 0000
next page sidlist. 0000000000002C10  prev page sidlist. 0000000000001010
next page aux......... 000000000000  prev page aux......... 000000000000
waitlist.......... 0000000000000000  logage.................... 00000000
nonfifo i/o............... 00000000  next i/o fr list...... 000000000000
```

```
(0)> more (^C to quit) ?
```

3.7.9 Storage keys performance impact

Kernel keys (protection gates) have some impact on system performance because they add overhead to the functions that contain them, whether they are implicit gates used by key-unsafe extensions, or explicit gates you use to make your extension key-aware. AIX provides a mechanism to disable kernel keys for benchmarking and environments where the kernel key performance loss is unacceptable.

If you make a key-safe extension by simply adding the minimal entry and exit point protection gates, it might actually run slightly faster than it otherwise would on a keys-enabled system, because explicit gates do not use the context stack. You must trade off granularity of protection against overhead as you move into the key-protected realm, however. For example, adding protection gates within a loop for precise access control to some private object might result in unacceptable overhead. Try to avoid such situations, where possible, in the framework of your specific key-protected design.

3.8 ProbeVue

Introduced in AIX V6.1, ProbeVue is a facility that enables dynamic tracing data collection. A tracing facility is *dynamic* because it is able to gather execution data from applications without any modification of their binaries or their source codes. The term “dynamic” refers to the capability to insert trace points at run-time without the need to prepare the source code in advance. Inserting specific tracing calls and defining specific tracing events into the source code would require you to recompile the software and generate a new executable, which is referred to as a *static* tracing facility.

Currently there are no standards in the area of dynamic tracing. POSIX has defined a tracing standard for static tracing software only as described in Chapter 1 of the IBM Redbooks publication *IBM AIX Version 6.1 Differences Guide*, SG24-7559. So, no compatibility between ProbeVue and other UNIX dynamic tracing facilities can be expected until a standard is established.

Dynamic tracing benefits and restrictions

Software debugging is often considered as a dedicated task running on development systems or test systems trying to mimic real customer production systems.

However, this general statement is currently evolving due to the recent advances in hardware capabilities and software engineering, such as:

- The processing and memory capabilities of high-end servers with associated storage technologies have lead to huge systems into production.
- Dedicated solutions developed by system integrators (for example, based on ERP software) implement numerous middleware and several application layers, and have lead to huge production systems.
- Software is now mostly multithreaded and running on many processors. Thus, two runs can behave differently, depending on the order of thread execution: multithreaded applications are generally non-deterministic. Erroneous behaviors are more difficult to reproduce and debug for such software.

Archived

As a consequence, determining the root cause of a problem in today's IT infrastructure has become prohibitively expensive and a significant burden if the troubleshooting cannot be accomplished on the production system itself.

The ProbeVue dynamic tracing facility provides a way to investigate problems on production systems. ProbeVue captures execution data *without* installing dedicated instrumented versions of applications or kernels that require a service interruption for application restart or server reboot.

Additionally, ProbeVue helps you to find the root cause of errors that may occur on long-running jobs where unexpected accumulated data, queue overflows and other defects of the application or kernel are revealed only after many days or months of execution.

Because ProbeVue is able to investigate any kind of application if a Probe manager is available (see "Probe manager" on page 117), it is a privileged tracing tool capable of analyzing a complex defect as a cascading failure between multiple subsystems. With a single tracing tool, ProbeVue allows a unified instrumentation of a production system.

Note the following ProbeVue considerations:

- Tracing an executable without modifying it requires you to encapsulate the binary code with a control execution layer. This control layer will interrupt the mainline code and start the instrumentation code at the trace point, to allow context tracing. The instrumented code is executed through an interpreter to prevent errors in this code from affecting the application or the kernel. Interpreter languages are known to be slower than compiled languages; the dynamic interpreted tracing points are potentially slower than the static compiled ones.
- System administrators and system integrators need to have deep application architecture knowledge, and not just knowledge about the tracing tool. If tracing points and actions are not set properly, then investigation of the software executed may be ineffective (in addition to making the application potentially slower). Also, you must keep in mind that application debugging in earlier development stages is always more effective, because software developers have a much better inside view of application architecture.

For these reasons, ProbeVue is a complementary tracing tool to the static tracing methods, adding new and innovative tracing capabilities to running production systems.

ProbeVue dynamic tracing benefits

As a dynamic tracing facility, ProbeVue has the following major benefits:

- Trace hooks do not have to be pre-compiled. ProbeVue works on unmodified kernel and user applications.
- The trace points or probes have no effect (do not exist) until they are dynamically enabled.
- Actions (specified by the instrumentation code) to be executed at a probe point or the probe actions are provided dynamically at the time the probe is enabled.
- Trace data captured as part of the probe actions are available for viewing immediately and can be displayed as terminal output or saved to a file for later viewing.

ProbeVue can be used for performance analysis as well as for debugging problems. It is designed to be safe to run on production systems and provides protection against errors in the instrumentation code.

The section defines some of the terminology used. The subsequent sections introduce Vue, the programming language used by ProbeVue and the **probevue** command that is used to start a tracing session.

Archived

3.8.1 ProbeVue terminology

ProbeVue introduces a terminology relative to the concepts used in dynamic tracing. The following list describes the terms used with ProbeVue.

Probe A probe is a software mechanism that interrupts normal system action to investigate and obtain information about current context and system state. This is also commonly referred to as *tracing*.

Tracing actions or **probe actions**
These terms refer to the actions performed by the probe. Typically, they include the capturing of information by writing the current values of global and context-specific information to a trace buffer. The obtained information, thus captured in the trace buffer, is called *trace data*. The system usually provides facilities to consume the trace; that is, to read the data out of the trace buffer and make it available to users.

Probe point A probe point identifies the points during normal system activity that are capable of being probed. With dynamic tracing, probe points do not have any probes installed in them unless they are being probed.
- *Enabling a probe* is the operation of attaching a probe to a probe point.
- *Disabling a probe* is the operation of removing a probe from a probe point.
- *Triggering* or *firing a probe* refers to the condition where a probe is entered and the tracing actions are performed.

ProbeVue supports two kinds of probe points.

Probe location This is a location in user or kernel code where some tracing action (for example, capture of trace data) is to be performed. Enabled probes at a probe location fire when any thread executing code reaches that location.

Probe event This is an abstract event at whose occurrence some tracing action is to be performed. Probe events do not easily map to a specific code location. Enabled probes that indicate a probe event fire when the abstract event occurs.

ProbeVue also distinguishes probe points by their type.

Probe type This identifies a set of probe points that share some common characteristics; for instance, probes that when enabled fire at the entry and exit of system calls, or probes that when enabled fire when system statistics are updated.

Distinguishing probes by probe types induces a structure to the wide variety of probe points. So, ProbeVue requires a probe manager to be associated with each probe type.

Probe manager This is the software code that defines and provides a set of probe points of the same probe type (for example, the *system calls* probe manager).

3.8.2 Vue programming language

The Vue programming language is used to provide your tracing specifications to ProbeVue. The Vue programming language is often abbreviated to the *Vue language* or just to *Vue*.

A Vue script or Vue program is a program written in Vue. You can use a Vue script to:

- Identify the probe points where a probe is to be dynamically enabled.
- Identify the conditions, if any, which must be satisfied for the actions to be executed when a probe fires.
- Identify the actions to be executed, including what trace data to capture.
- Associate the same set of actions for multiple probe points.

In short, a Vue script tells ProbeVue where to trace, when to trace, and what to trace.

It is recommended that Vue scripts have a file suffix of *.e* to distinguish them from other file types, although this is not a requirement.

3.8.3 The probevue command

The **probevue** command is used to start a dynamic tracing session or a ProbeVue session. The **probevue** command takes a Vue script as input, reading from a file or from the command line, and activates a ProbeVue session. Any trace data that is captured by the ProbeVue session can be printed to the terminal or saved to a user-specified file as per options passed in the command line.

The ProbeVue session stays active until a <Ctrl-C> is typed on the terminal or an exit action is executed from within the Vue script.

Each invocation of the **probevue** command activates a separate dynamic tracing session. Multiple tracing sessions may be active at one time, but each session presents only the trace data that is captured in that session.

Running the **probevue** command is considered a privileged operation, and privileges are required for non-root users who wish to initiate a dynamic tracing session. For a detailed description of the **probevue** command, refer to *AIX Version 6.1 Commands Reference, Volume 4*, SC23-5246.

3.8.4 The probevctrl command

The **probevctrl** command changes and displays the ProbeVue dynamic tracing parameters, the per-processor trace buffer size, the consumed pinned memory, the user owning the session, the identifier of the process that started the session, and the information on whether the session has kernel probes for the ProbeVue sessions.

For a detailed description of the **probevctrl** command, refer to *AIX Version 6.1 Commands Reference, Volume 4*, SC23-5246.

3.8.5 Vue overview

Vue is both a programming and a script language. It is not an extension of C language, nor a simple mix of C and **awk**. It has been specifically designed as a dedicated dynamic tracing language. Vue supports a subset of C and scripting syntax that is most beneficial for dynamic tracing purposes.

This section describes the structure of a Vue script.

Structure of a Vue script

A Vue script consists of one or more clauses. The clauses in a Vue script can be specified in any order. Figure 3-3 is a typical layout of a Vue script.

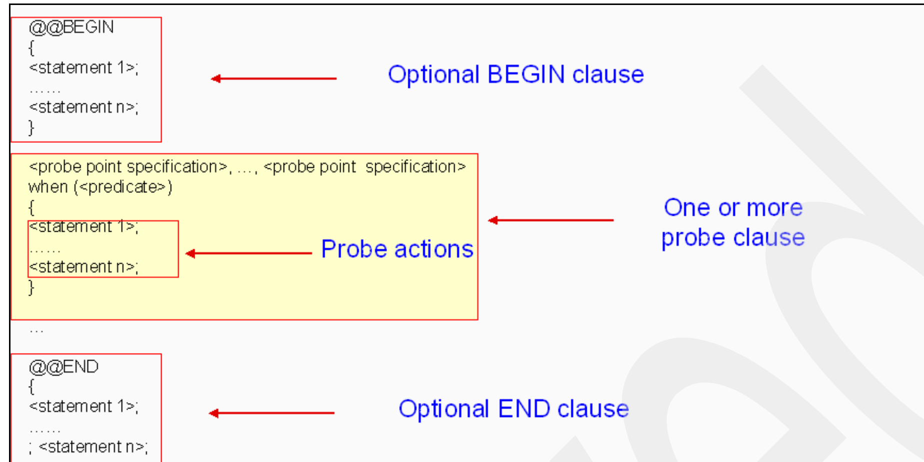


Figure 3-3 Structure of a Vue script

Following are two Vue script examples.

- This canonical “Hello World” program prints "Hello World" into the trace buffer and exits.

```
#!/usr/bin/probevue

   /* Hello World in probevue */
   /* Program name: hello.e */

   @@BEGIN
   {
      printf("Hello World\n");
      exit();
   }
```

- This “Hello World” program prints "Hello World" when <Ctrl-C> is typed on the keyboard.

```
#!/usr/bin/probevue

   /* Hello World 2 in probevue */
   /* Program name: hello2.e */

   @@END
   {
      printf("Hello World\n");
   }
```

Each clause of a Vue script consists of the following three elements:

- Probe point specification

  The probe point specification identifies the probe points to be dynamically enabled.

- Action block

  The action block is used to identify the set of probe actions to be performed when the probe fires.

Archived

- An optional predicate

  The predicate, if present, identifies a condition that is to be checked at the time the probe is triggered. The predicate must evaluate to TRUE for the probe actions of the clause to be executed.

These elements are described in more detail in the following sections.

Probe point specification

A probe point specification identifies the code location whose execution, or the event whose occurrence, should trigger the probe actions. Multiple probe points can be associated with the same set of probe actions and the predicate, if any, by providing a comma-separated list of probe specifications at the top of the Vue clause.

The format for a probe specification is probe-type specific. The probe specification is a tuple of ordered list of fields separated by colons. It has the following general format:

```
@@<probetype>:<probetype field1>:...:<probetype fieldn>:<location>
```

AIX V6.1 supports the following probe types:

1. User Function Entry probes (or uft probes)

   For example, a uft probe at entry into any function called foo() (in the main executable or any of the loaded modules including libraries) in process with ID = 34568:

   ```
   @@uft:34568:*:foo:entry
   ```

2. System Call Entry/Exit probes (or syscall probes)

   For example, a syscall probe at the exit of a read system call:

   ```
   @@syscall:*:read:exit
   ```

3. Probes that fire at specific time intervals (or interval probes)

   For example, an interval probe to fire every 500 milliseconds (wall clock time):

   ```
   @@interval:*:clock:500
   ```

Action block

The action block identifies the set of actions to be performed when a thread hits the probe point. Supported actions are not restricted to the capturing and formatting of trace data; the full power of Vue can be employed.

An action block in Vue is similar to a procedure in procedural languages. It consists of a sequence of statements that are executed in order. The flow of execution is essentially sequential. The only exceptions are that conditional execution is possible using the if-else statement, and control may be returned from within the action block using the return statement.

Unlike procedures in procedural languages, an action block in Vue does *not* have an output or a return value. And it does not have inherent support for a set of input parameters. On the other hand, the context data at the point where a probe is entered can be accessed within the action block to regulate the actions to be performed.

Predicate

Predicates should be used when execution of clauses at probe points must be performed conditionally.

Archived

The predicate section is identified by the presence of the **when** keyword immediately after the probe specification section. The predicate itself consists of regular C-style conditional expressions with the enclosing parentheses.

A predicate has the following format:

```
when ( <condition> )
```

For example, this is a predicate indicating that probe points should be executed for process ID = 1678:

```
when ( __pid == 1678 )
```

Probe manager

The probe manager is an essential component of dynamic tracing. Probe managers are the providers of the probe points that can be instrumented by ProbeVue.

Probe managers generally support a set of probe points that belong to some common domain and share some common feature or attribute that distinguishes them from other probe points. Probe points are useful at points where control flow changes significantly, at points of state change or other similar points that of significant interest. Probe managers are careful to select probe points only in locations that are safe to instrument.

ProbeVue currently supports the following three probe managers:

1. System call probe manager

   The system call (syscall) probe manager supports probes at the entry and exit of well-defined and documented base AIX system calls. The syscall probe manager accepts a four tuple probe specification in one of the following formats where the `<system_call_name>` field is to be substituted by the actual system call name.:

   ```
   * syscall:*:<system_call_name>:entry
   * syscall:*:<system_call_name>:exit
   ```

   These indicate that a probe is to be placed at the entry and exit of system calls. Assigning the asterisk (*) to the second field indicates that the probe will be fired for all processes. In addition, a process ID can be specified as the second field of the probe specification to support probing of specific processes.

   ```
   * syscall:<process_ID>:<system_call_name>:entry
   * syscall:<process_ID>:<system_call_name>:entry
   ```

2. User function probe manager

   The user function tracing (uft) probe manager supports probing user space functions that are visible in the XCOFF symbol table of a process. These entry points, usable as probe points, are currently are restricted to those written in C language text file. The uft probe manager currently accepts a five tuple probe specification only in the following format:

   ```
   uft:<processID>:*:<function_name>:entry
   ```

   Note: The uft probe manager requires the process ID for the process to be traced and the complete function name of the function at whose entry point the probe is to be placed. Further, the uft probe manager currently requires that the third field be set to an asterisk (*) to indicate that the function name is to be searched in any of the modules loaded into the process address space, including the main executable and shared modules.

3. Interval probe manager

   The interval probe manager supports probe points that fire at a user-defined time-interval. The probe points are not located in kernel or application code, but instead are based on wall clock time interval-based probe events. The interval probe manager accepts a four tuple probe specification in the following format:

   ```
   @@interval:*:clock:<# milliseconds>
   ```

   The second field is an asterisk (*), indicating that the probe can be fired in any process. Currently, the interval probe manager does not filter probe events by process IDs. For the third field, the only value supported is currently the clock keyword that identifies the probe specification as being for a wall clock probe.

   The fourth or last field, that is, the <# milliseconds> field, identifies the number of milliseconds between firings of the probe. Currently, the interval probe manager requires that the value for this field be exactly divisible by 100 and consist only of digits 0 - 9. Thus, probe events that are apart by 100ms, 200ms, 300ms, and so on, are allowed.

Vue functions

Unlike programs written in C or in FORTRAN programming languages or in a native language, scripts written in Vue do not have access to the routines provided by the AIX system libraries or any user libraries. However, Vue supports its own special library of functions useful for dynamic tracing programs. Functions include:

Tracing-specific functions

get_function — Returns the name of the function that encloses the current probe
timestamp — Returns the current time stamp
diff_time — Finds the difference between two time stamps

Trace capture functions

printf — Formats and prints values of variables and expressions
trace — Prints data without formatting
stktrace — Formats and prints the stack trace

List functions

list — Instantiate a list variable
append — Append a new item to list
sum, max, min, avg, count — Aggregation functions that can be applied on a list variable

C-library functions

atoi, strstr — Standard string functions

Functions to support tentative tracing

start_tentative, end_tentative — Indicators for start and end of tentative tracing
commit_tentative, discard_tentative — Commit or discard data in tentative buffer

Miscellaneous functions

exit — Terminates the tracing program
get_userstring — Read string from user memory

Archived

The Vue string functions can be applied only on variables of string type and not on a pointer variable. Standard string functions like strcpy(), strcat(), and so on, are not necessary in Vue, because they are supported through the language syntax itself.

For additional information, see the article “ProbeVue: Extended Users Guide Specification” at:

http://www.ibm.com/developerworks/aix/library/au-probevue/

3.8.6 ProbeVue dynamic tracing example

This is a basic ProbeVue example to show how ProbeVue works and how to use ProbeVue on a running executable without restarting or recompiling it.

The following steps must be performed:

1. The C program shown in Example 3-38, named `pvue`, is going to be traced dynamically.

Example 3-38 Basic C program to be dynamically traced: pvue.c

```
#include <fcntl.h>
main()
{
int x, rc;
int buff[100];

for (x=0; x<5; x++){
        sleep(3);
        printf("x=%d\n",x);
}
sleep (3);
fd=open("./pvue.c",O_RDWR,0);
x =read(fd,buff,100);
printf("[%s]\n",buff);
}
```

2. We compile and execute the program in background:

   ```
   # cc -q64 -o pvue pvue.c
   # ./pvue &
   [1]     262272
   ```

   The command returns the process ID (212272), which will be used as a parameter for the Vue script.

3. In order to trace dynamically the number of calls executed by the pvue process to the subroutines printf(), sleep(), entry of read(), and exit of read(), we have developed a ProbeVue script, named pvue.e, shown in Example 3-39 on page 119.

   The script uses the process ID as an entry parameter ($1). Note also the first line of the Vue script, specifying that the **probevue** program is used as an interpreter for this script.

Example 3-39 Sample Vue script, named pvue.e

```
#!/usr/bin/probevue
@@BEGIN
{
        printf("Tracing starts now\n");
}
@@uft:$1:*:printf:entry
```

```
{
        int count;
        count = count +1;
        printf("printf called %d times\n",count);
}
@@uft:$1:*:sleep:entry
{
        int count1;
        count1 = count1 +1;
        printf("sleep called %d times\n",count1);
}
@@syscall:*:read:exit
        when (__pid == $1)
{
        printf("read entered\n");
}
@@syscall:*:read:entry
        when (__pid == $1)
{
        printf("read exited\n");
}
@@END
{
        printf("Tracing ends now\n");
}
```

4. We execute the pvue.e script with the **probevue** command passing the process ID to be traced as parameter:

   ```
   # probevue ./pvue.e  262272
   ```

We obtain the tracing output shown in Example 3-40.

Example 3-40 Start Vue script providing pid

```
# ./pvue.e  262272
Tracing starts now
printf called 1 times
sleep called 1 times
printf called 2 times
/*
* Although the sleep precedes the printf in the loop, the reason we got the printf
in the trace was because the program was in the middle of the first sleep when we
started tracing it, and so the first thing we traced was the subsequent printf.
*/
sleep called 2 times
printf called 3 times
sleep called 3 times
printf called 4 times
sleep called 4 times
printf called 5 times
sleep called 5 times
read exited
read entered
printf called 6 times
^CTracing ends now
#
```

Vue built-in variables

Vue defines the following set of general purpose built-in variables:

Variable	Description
__tid	Thread Id of target thread
__pid	Process Id of target thread
__ppid	Parent process Id of target thread
__pgid	Process group Id of target thread
__pname	Process name of target thread
__uid, __euid	Real and effective user Id of target thread
__trcid	Process Id of tracing thread
__errno	Current errno value for the target thread
__kernelmode	Kernel mode (value = 1) or User mode (value = 0)
__execname	Current executable name
__r3,..,__r10	GP Register values for function parameters or return values

Built-in variables may be used in the predicate section of a Vue clause. To see __pid, refer to Example 3-39 on page 119.

String

The string data type is a representation of string literals. The user specifies the length of the string type. Here is an example of a string declaration:

```
“String s[25]”
```

The following operators are supported for the string data types:

```
"+", "=", "==", "!=", ">", ">=", "<" and "<=".
```

Vue supports several functions that return a string data type. It automatically converts between a string data type and C-style character data types (char * or char[]) as needed.

List

List is used to collect a set of integral type values. It is an abstract data type and cannot be used directly with the standard unary or binary operators. Instead, Vue supports following operations for the list type:

- A constructor function, list() to create a list variable.
- A concatenation function, append() to add an item to the list.
- The "=" operator that allows a list to be assigned to another.
- A set of aggregation functions that operate on a list variable and return a scalar (integer) value like sum(), avg(), min(), max(), and so on.

Example 3-41 on page 123 uses the list variable `lst`. Prototypes and a detailed explanation of List data types can be found in “Chapter 4. Dynamic Tracing” in *AIX Version 6.1 General Programming Concepts: Writing and Debugging Programs*, SC23-5259.

Symbolic constants

Vue has predefined symbolic constants which can be used:-

- NULL

Archived

- Errno names
- Signal names
- FUNCTION_ENTRY: Identifies function entry point. Used with get_location_point()
- FUNCTION_EXIT: Identifies function exit point. Used with get_location_point()

Supported keywords/data-types of C in Vue

Table 3-7 lists the C keywords used in the Vue language.

Table 3-7 C keywords in Vue

Supported	Allowed in header files only	Unsupported
char	auto	break
double	const	case
else	extern	continue
enum	register	default
float	static	do
if	typedef	for
int	volatile	goto
long		switch
return		while
short		
signed		
sizeof		
struct		
union		
unsigned		
void		

Elements of shell

Vue translates exported shell variables (specified by the $ prefix) and positional parameters into their real values during the initial phase of compilation. So, $1, $2, and so on, will be replaced with corresponding value by ProbeVue.

When assigning general environment variables to a string, you need to make sure it starts and ends with a backward (\) slashmark. For instance, the environment variable "VAR=abcdef" will result in an error when assigned to a string. Defining it as "VAR=\"abcdef\"" is the proper way of using it in a Vue script.

Restriction: The special shell parameters like $$, $@, and so on are not supported in Vue. However, they can be obtained by other predefined built-in variables.

Example 3-41 illustrates the use of various variable types, and also uses kernel variables. The comments provided in the Vue script explain the scope of various variables.

Archived

Example 3-41   Sample Vue script pvue2.e

```
#!/usr/bin/probevue

/*
 * Strings are by default Global variables
 */
String global_var[4096];

/*
 * Global variables are accessible throughout the scope of Vue file in any clause
 */
__global global_var1;
__kernel long lbolt;

/*
 * Thread variables are like globals but instantiated per traced thread first
 * time it executes an action block that uses the variable
 */
__thread int thread_var;

/*
 * Built-in variables  are not supposed to be defined. They are by default
 * available to the clauses where is makes any sense. They are : __rv, __arg1,
 * __pid etc. __rv (return value for system calls) makes sense only when system
 * call is returning and hence available only for syscall()->exit() action block.
 * While __arg1, __arg2 and arg3 are accessible at syscall()->entry(). As system
 * call read() accespts only 3 arguments so only __arg1, __arg2 and __arg2 are
 * valid for read()->entry() action block.
 * __pid can be accessed in any clause as current process id is valid everywhere
 */
int read(int fd, void *buf, int n);

@@BEGIN
{
   global_var1=0;
   lst=list();
   printf("lbolt=%lld\n",lbolt);
}

@@uft:$1:*:printf:entry
{
   global_var1=global_var1+1;
   append(lst,1);
}

/*
 * Valid built-in variables : __pid, __arg1, __arg2, __arg3
 * __rv not valid here, its valid only inside read()->exit()
 */
@@syscall:*:read:entry
   when ( __pid == $1 )
{

/*
 * Automatic variable is not accessible outside their own clause. So auto_var
```

```
 * will result in error out of read()->entry()
*/
   __auto int auto_var;

   thread_var=__arg1;
   global_var=get_userstring(__arg2,15);
   global_var1=__arg3;

   printf("At read()->entry():\n");
   printf("\tfile descriptor        ====>%d\n", thread_var);
   printf("\tfile context (15 bytes)====>%s\n", global_var);
   printf("\tMAX buffer size        ====>%d\n", global_var1);
}

/* Valid built-in variables : __pid, __rv
 * __arg1, __arg2 and __arg2 are not valid here, they are valid only in
 * read()->entry()
 */
@@syscall:*:read:exit
   when ( __pid == $1 )
{
   printf("At read()->exit(): bytes read=%d by read(%d, %s, %d)\n",__rv,
           thread_var, global_var, global_var1);
}
@@END
{
   /*
    * auto_var not accessible here  as its an Automatic variable in clause
"read:entry"
    */
   printf("\nthread_var=%d global_var=%s global_var1=%d\n", thread_var,
           global_var, global_var1);
}
```

Example 3-42 displays the C file that will be traced using the Vue script.

Example 3-42 Sample C program to be traced

```
#include <fcntl.h>
main()
{
   int x,rc,fd;
   int buff[4096];
   for (x=0; x<5; x++){
      sleep(3);
      printf("x=%d\n",x);
      fd=open("./pvue.c",O_RDWR,0);
      rc =read(fd,buff,4096);
      close(fd);
   }
}
```

Example 3-43 displays the output of the program traced.

Archived

Example 3-43 Output of pvue2.e script

```
# ./pvue2.e 356786
lbolt=13454557
At read()->entry():
        file descriptor        ====>3
        file context (15 bytes)====>
        MAX buffer size        ====>4096
At read()->exit(): bytes read=190 by read(3, , 4096)
At read()->entry():
        file descriptor        ====>3
        file context (15 bytes)====>#include <fcntl
        MAX buffer size        ====>4096
At read()->exit(): bytes read=190 by read(3, #include <fcntl, 4096)
At read()->entry():
        file descriptor        ====>3
        file context (15 bytes)====>#include <fcntl
        MAX buffer size        ====>4096
At read()->exit(): bytes read=190 by read(3, #include <fcntl, 4096)
At read()->entry():
        file descriptor        ====>3
        file context (15 bytes)====>#include <fcntl
        MAX buffer size        ====>4096
At read()->exit(): bytes read=190 by read(3, #include <fcntl, 4096)
At read()->entry():
        file descriptor        ====>3
        file context (15 bytes)====>#include <fcntl
        MAX buffer size        ====>4096
At read()->exit(): bytes read=190 by read(3, #include <fcntl, 4096)
^C
```

3.8.7 Other considerations for ProbeVue

ProbeVue is supported on workload partitions (WPARs). However, WPARs cannot be migrated if any ProbeVue sessions are active on it. ProbeVue privileges must be granted on WPARs.

3.9 Xmalloc debug enhancements in AIX V6.1

Xmalloc debug (XMDBG) is a service provided as part of the xmalloc allocator. When enabled in the kernel, XMDBG provides significant error checking for the xmalloc() and xmfree() kernel services.

XMDBG will catch errors that might otherwise result in system outages, such as traps, Data Storage Interrupts (DSIs), and hangs. Typical errors include freeing memory that is not allocated, allocating memory without freeing it (memory leak), using memory before initializing it, and writing to freed storage.

In previous AIX versions, XMDBG features required a system reboot to be enabled. Additionally, because this was essential for catching certain types of memory issues, it was common to request that a customer enable XMDBG and then recreate the problem.

Archived

This places a large part of the burden of data collection on the customer, and limited first-failure data capture ability (FFDC). With AIX V6.1 (and also AIX V5.3), XMDBG will be enabled by default with multiple debugging levels (unlike in AIX 5.3, which only allows enable/disable), and switching between these levels does not require system reboot.

> **Note:** This section is meant only for support personnel or for system administrators under the supervision of support personnel. End customer use without supervision is not recommended.

3.9.1 New features in xmalloc debug

The following new features have been added to the xmalloc debug enhancement for Run time error checking (RTEC):

- Four debugging levels for xmalloc debug (XMDBG) are provided:
  - disabled (debug level -1)
  - minimal (debug level 1): ERRCHECK_MINIMAL
  - normal (debug level 3): ERRCHECK_NORMAL
  - detail (debug level 7): ERRCHECK_DETAIL
  - maximal (debug level 9): ERRCHECK_MAXIMAL

  Disabled and detail modes are the same as off and on in AIX V5.3. Minimal and normal modes will add randomized information gathering, with all the First Failure Data Capture capability of detail XMDBG, but with a lessened probability of occurring (which reduces the performance impact, but also lessens the likelihood of catching a potential problem).

  The default checking level on AIX V6.1 is ERRCHECK_NORMAL. In AIX V5.3, it used to be ERRCHECK_MINIMAL. Maximal error checking is performed at ERRCHECK_MAXIMAL.
- The xmalloc run time error checking (RTEC) function is integrated into the RAS component hierarchy and appears as the alloc component. The alloc component has several subcomponents. XMDBG capabilities are controlled under the alloc.xmdbg subcomponent. This will enable system administrators to alter many tunable aspects of XMDBG using the **errctrl** command.
- Debug features are enabled across any heap created via the heap_create() interface, including private heaps.

3.9.2 Enabling/disabling xmalloc RTEC and displaying current value

The xmalloc RTEC can be controlled in different ways. At one level, xmalloc RTEC can be disabled (or re-enabled) along with all other AIX run-time error checking, or it can be controlled individually.

Example 3-1 on page 57 shows how to use the smit menu command **smit ffdc** to enable RTEC. Alternatively, you can use the following commands:

- Turn Off Error Checking for xmalloc

  ```
  errctrl -c alloc.xmdbg errcheckoff
  ```
- Turn On Error Checking for xmalloc for the previously set checking level or default level if no previous check level exists.

  ```
  errctrl -c alloc.xmdbg errcheckon
  ```

Note: Executing **errctrl** with the **–P** flag will apply the changes for future reboots as well. There was no support for the **-P** flag in prior AIX versions.

To display the current RTEC level for xmalloc and its subcomponents, execute the following command:

```
errctrl -c alloc -q -r
```

Example 3-44 RTEC level for alloc and its subcomponent

```
# errctrl -c alloc -q -r
---------------------------------------------+-------+-------+-------+--------
                                             | Have  |ErrChk |LowSev |MedSev
    Component name                           | alias | /level| Disp  | Disp
---------------------------------------------+-------+-------+-------+--------
alloc
     .heap0                                  |  NO   | ON /0 |  48   |  64
     .xmdbg                                  |  NO   | ON /9 |  64   |  80
```

Example 3-44 shows the RTEC level for alloc and its subcomponents. Note that alloc.xmdbg is set to errorcheckmaximum (which is explained in more detail in 3.9.3, "Run-time error checking (RTEC) levels for XMDBG (alloc.xmdbg)" on page 127). In this example, alloc.heap0 has no error checking enabled.

To display the current RTEC level for any subcomponent of xmalloc, execute:

```
errctrl -c alloc.<subcomponent> -q -r
```

For example, the command **errctrl -c alloc.xmdbg -q -r** will show the RTEC level for alloc.xmdbg.

3.9.3 Run-time error checking (RTEC) levels for XMDBG (alloc.xmdbg)

The probability of taking each option will be determined by the debug level set for the alloc.xmalloc component. However, there is also a way to tune individual probabilities to desired levels. For further information about forcing individual tunables, refer to 3.9.4, "XMDBG tunables affected by error check level" on page 131.

Error checking characteristics can be changed for xmalloc subsystem with component-specific tunables. All options can be configured at run time by using the **errctrl** command.

Minimal Error Checking Level

When the error-checking level is set to minimal (level 1), the checks and techniques used by xmalloc are applied at fairly low frequencies. It can be set to minimal by executing:

```
errctrl -c alloc.xmdbg errcheckminimal
```

The frequency of various xmalloc debug tunables can be viewed by using the **kdb** subcommand **xm –Q**. Example 3-45 on page 128 shows the frequencies for various tunables.

Minimal checking is the default checking level on version 5. The frequency that appears next to each tunable is proportional to the frequency base (1024). From the example, you can see that the `Ruin All Data` technique will be applied 5 times out of every 1024 (0x400) calls to xmalloc() (about 0.5% of every 1024 xmalloc() calls). Also, 16 byte allocations will be promoted about 10 times out of every 1024 calls to xmalloc(), which is about 1% of the time.

Note: The frequency of checks made for all checking levels is subject to change.

Example 3-45 Frequencies for xmalloc debug tunable at minimal level

```
(0)> xm -Q
XMDBG data structure @ 00000000025426F0
Debug State:   Enabled
Frequency Base:    00000400
Tunable                     Frequency
Allocation Record           00000033
Ruin All Data               00000005
Trailer non-fragments       00000005
Trailer in fragments        00000005
Redzone Page                00000005
VMM Check                   0000000A
Deferred Free Settings
  Fragments                 00000005
  Non-fragments             00000005
  Promotions                00000066

Page Promotion
       Frag size         Frequency
        [00010]          0000000A
        [00020]          0000000A
        [00040]          0000000A
        [00080]          0000000A
        [00100]          0000000A
        [00200]          0000000A
        [00400]          0000000A
        [00800]          0000000A
        [01000]          0000000A
        [02000]          0000000A
        [04000]          0000000A
        [08000]          0000000A

Ratio of memory to declare a memory leak: 0x400(1024)/0x400(1024)
Outstanding memory allocations to declare a memory leak: -1
Deferred page reclamation count (-1 == when necessary): 16384
Minimum allocation size to force a record for: 1048576
```

Normal error checking level

When the error checking level is set to normal (level 3), the checks and techniques are applied at higher frequencies than what minimal checking provides. Normal error checking is the default level setting in AIX V6.1. It can be set by executing the following command:

```
errctrl -c alloc.xmdbg errchecknormal
```

In Example 3-46 on page 128, frequencies for xmalloc tunables at normal level are shown. A "trailer" will be added to a fragment about 51 (0x33) times out of every 1024 times a fragment is allocated (about 5%). The deferred free technique will be applied to page promotions about 153 (0x99) times out of every 1024 (0x400) times a fragment is promoted, which is about 15% of the time.

Example 3-46 Frequencies for xmalloc tunables at normal level

```
(0)> xm -Q
XMDBG data structure @ 00000000025426F0
Debug State:   Enabled
```

```
Frequency Base:    00000400
Tunable                     Frequency
Allocation Record           00000099
Ruin All Data               00000033
Trailer non-fragments       0000000A
Trailer in fragments        00000033
Redzone Page                0000000A
VMM Check                   0000000A
Deferred Free Settings
  Fragments                 0000000A
  Non-fragments             0000000A
  Promotions                00000099

Page Promotion
       Frag size         Frequency
        [00010]          0000000D
        [00020]          0000000D
        [00040]          0000000D
        [00080]          0000000D
        [00100]          0000000D
        [00200]          0000000D
        [00400]          0000000D
        [00800]          0000000D
        [01000]          0000000D
        [02000]          0000000D
        [04000]          0000000D
        [08000]          0000000D

Ratio of memory to declare a memory leak: 0x400(1024)/0x400(1024)
Outstanding memory allocations to declare a memory leak: -1
Deferred page reclamation count (-1 == when necessary): 16384
Minimum allocation size to force a record for: 1048576
```

Detail error checking level

Detail error checking level corresponds to level 7. The checks and techniques are applied at fairly high frequencies. This gives a high checking level with a goal of affecting the overall system performance as little as possible. It can be set by executing:

```
errctrl -c alloc.xmdbg errcheckdetail
```

Example 3-47 shows frequencies for various tunables of alloc.xmdbg at level errcheckdetail. For instance, Allocation Records are kept on every call to xmalloc() (0x400 out of 0x400 calls). 0x80 byte Fragments are promoted 0x200 out of every 0x400 times the 0x80 byte fragment is allocated (50%).

Example 3-47 Frequencies for xmalloc at detail level

```
(0)> xm -Q
XMDBG data structure @ 00000000025426F0
Debug State:   Enabled
Frequency Base:    00000400
Tunable                     Frequency
Allocation Record           00000400
Ruin All Data               00000200
Trailer non-fragments       00000066
Trailer in fragments        00000200
Redzone Page                00000266
VMM Check                   00000266
Deferred Free Settings
  Fragments                 00000066
```

```
  Non-fragments             00000066
  Promotions                00000200

Page Promotion
       Frag size         Frequency
        [00010]          00000200
        [00020]          00000200
        [00040]          00000200
        [00080]          00000200
        [00100]          00000200
        [00200]          00000200
        [00400]          00000200
        [00800]          00000200
        [01000]          00000200
        [02000]          00000200
        [04000]          00000200
        [08000]          00000200

Ratio of memory to declare a memory leak: 0x400(1024)/0x400(1024)
Outstanding memory allocations to declare a memory leak: -1
Deferred page reclamation count (-1 == when necessary): 16384
Minimum allocation size to force a record for: 1048576
```

Maximal error checking level

At maximal error checking level, all tunables are set at maximum levels (level 9). Performance is greatly affected at this checking level. All the frequencies should match the frequency base, meaning all checks are always done. It is set by executing:

```
errctrl -c alloc.xmdbg errchecklevel=9
```

Example 3-48 shows frequencies for various tunables of alloc.xmdbg at the highest RTEC level.

Example 3-48 Frequencies for xmalloc at maximal level

```
(0)> xm -Q
XMDBG data structure @ 00000000025426F0
Debug State:   Enabled
Frequency Base:    00000400
Tunable                     Frequency
Allocation Record           00000400
Ruin All Data               00000400
Trailer non-fragments       00000400
Trailer in fragments        00000400
Redzone Page                00000400
VMM Check                   00000400
Deferred Free Settings
  Fragments                 00000400
  Non-fragments             00000400
  Promotions                00000400

Page Promotion
       Frag size         Frequency
        [00010]          00000400
        [00020]          00000400
        [00040]          00000400
        [00080]          00000400
        [00100]          00000400
        [00200]          00000400
        [00400]          00000400
```

```
        [00800]          00000400
        [01000]          00000400
        [02000]          00000400
        [04000]          00000400
        [08000]          00000400

Ratio of memory to declare a memory leak: 0x400(1024)/0x400(1024)
Outstanding memory allocations to declare a memory leak: -1
Deferred page reclamation count (-1 == when necessary): 16384
Minimum allocation size to force a record for: 1048576
```

Note: The **bosdebug -M** command sets all the frequencies for alloc.xmdbg at maximal level except for "promotion settings" which are all set to zero (0). A reboot is required in order for **bosdebug** to take effect.

3.9.4 XMDBG tunables affected by error check level

As previously mentioned, xmalloc RTEC features are activated for a given allocation based on probabilities. The **errctrl** command that controls the tunables takes the probability of application (frequency) as an argument.

In AIX V6.1, the user can set the probability of a check being performed by specifying the frequency of a tunable as a number between 0 and 1024. This is the number of times out of the base frequency (1024) that the technique is to be applied by xmalloc. For example, to request 50%, the user specifies a frequency of 512.

Frequencies can be input as decimal or hex numbers, so 50% can be specified as 0x200. As a convenient alternative, the frequency can be expressed as a percentage. To do this, the user specifies a number between 0 and 100 followed by the percent (%) sign. The following sections detail the RTEC tunables for the xmalloc.xmdbg component.

Keep an allocation record

This option sets the frequency of keeping a record for an allocation. Records are also kept if any other debug technique is applied, so the percentage of allocations with a record may be considerably larger than this number would otherwise indicate.

The allocation record contains a three-level stack trace-back of the xmalloc() and xmfree() callers, as well as other debug information about the allocated memory. The presence of a record is a minimum requirement for xmalloc run time error checking.

```
errctrl -c alloc.xmdbg alloc_record=<frequency>
```

Ruin storage

This option sets the frequency at which xmalloc() will return storage that is filled with a "ruin" pattern. This helps catch errors with un-initialized storage, because a caller with bugs is more likely to crash when using the ruined storage. Note that xmalloc() does not perform any explicit checks when this technique is employed. The ruined data will contain 0x66 in every allocated byte on allocation, and 0x77 in every previously allocated byte after being freed.

```
errctrl -c alloc.xmdbg ruin_all=<frequency>
```

Check for overwrites in small allocations

This is one of the three options that affect the frequency of trailers. There are two options that deal with trailers, and a third option deals with compatibility with previous versions of AIX. This option is specific for allocations that are less than half a page. A trailer is written

immediately after the returned storage. Trailers can consume up to 128 bytes of storage. When storage is freed, xmfree() will ensure consistency in the trailer bytes and log an error.

```
errctrl -c alloc.xmdbg small_trailer=<frequency>
```

Note: The tunable small_trailer did not exist on 5.3, because all trailers were controlled with the single tunable known as alloc_trailer.

The error disposition can be made more severe by changing the disposition of medium severity errors as follows:

```
errctrl -c alloc.xmdbg medsevdisposition=sysdump
```

Be aware, however, that overwrites to the trailers and other medium severity errors will cause a system crash if the severity disposition is changed to be more severe.

Check for overwrites in large allocations

This option sets the frequency of trailers that are added to allocations that require at least a full page. This technique catches the same type of errors as a redzone, but a redzone always starts at the next page boundary, and a trailer follows immediately after the bytes that are beyond the requested size.

Trailers are checked at fragment free time for consistency. The error disposition can be affected for these checks just as it is for the small_trailer option. Trailers and redzones can be used together to ensure that overruns are detected. Trailers are not used if the requested size is exactly a multiple of the page size. Overwrites can still be detected by using the redzone option.

```
errctrl -c alloc.xmdbg large_trailer=<frequency>
```

Check for overwrites in all allocations

This option is provided just for compatibility with AIX 5.3. It sets the frequency that xmalloc() will add a trailer to all allocations. To accomplish this, it overwrites the settings of both the small_trailer and large_trailer options.

```
errctrl -c alloc.xmdbg alloc_trailer=<frequency>
```

Promote fragment allocations to whole pages

When an allocation that is less than half a 4 K page is promoted, the returned pointer is as close to the end of the page as possible while satisfying alignment restrictions and an extra “redzone” page is constructed after the allocated region. No other fragments are allocated from this page.

This provides isolation for the returned memory and catches users that overrun buffers. When used in conjunction with the df_promote option, this also helps catch references to freed memory. This option uses substantially more memory than other options.

Sizes that are greater than 2 K are still promoted in the sense that an extra redzone page is constructed for them.

Note: The page size of the heap passed to xmalloc() makes no difference. If the heap normally contains 64 K pages (kernel_heap or pinned_heap on a machine that supports a 64 K kernel heap page size), then the returned memory of a promoted allocation will still be backed by 4 K pages.

These promoted allocations come from a region that has a 4 K page size, to avoid using an entire 64 K page as a redzone.

The following option sets the frequency for which allocations are promoted. Supported sizes are all powers of two: 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, and 32768.

```
errctrl -c alloc.xmdbg promote=<size>,<frequency>
```

Note: In AIX V5.3, this feature did not provide a redzone page, and always caused the freeing of fragment to be deferred. To provide a redzone page, 5.3 used:

```
errctrl -c alloc.xmdbg doublepage_promote=<size>,<frequency>
```

In AIX V6.1, this option is provided but the function is identical to the promote option.

Also, in AIX V5.3, the doublepage_promote option always caused the freeing of fragment to be deferred.

Change the promotion settings of all sizes at once

This option duplicates the function of the promote option, but does not take size as an argument. It applies the input frequency to all the promotion sizes with a single command.

```
ecctrl –c alloc.xmdbg promote_all=<frequency>
```

Note: The command **`bosdebug -s <promotion_frequency>`** is used to set promotion setting for subsequent reboot.

Defer the freeing of pages/promoted allocations

The deferred free technique means that when a memory object is freed, xmalloc() will take measures to ensure that the object is not reallocated immediately. This technique helps catch references to memory that has been freed.

```
errctrl -c alloc.xmdbg df_promote=<frequency>
```

This option affects the freeing of promoted fragments. It sets the frequency with which the freeing of promoted fragment is deferred. Page promotion (that is, the promote option) and df_promote are designed to be used together.

Defer the freeing of pages/small allocations

This option sets the frequency at which non-promoted fragments will be deferred. A memory page that xmalloc manages contains multiple fragments of the same size or is part of a range of pages.

Be aware that there is a difference between the option def_free_frag and the option df_promote. The options are similar, but with def_free_frag, the freeing of every fragment on a page will be deferred together. This implies the number of pages used by these two techniques is substantially different:

- The df_promote option constructs *one* fragment per page (with an additional redzone page).
- The def_free_frag option constructs *multiple* fragments per page (with no redzone).

```
errctrl -c alloc.xmdbg def_free_frag=<frequency>
```

Note: The options def_free_frag, promote_all, and df_promote do not exist in AIX 5.3.

Defer the freeing of pages/large allocations

This option helps catch references to memory that has been freed. It sets the frequency at which xmalloc defers the freeing of larger allocations. Larger allocations are at least one entire 4 K page in size.

Archived

This option should be used with care because it can be expensive from a performance standpoint. When large ranges are freed and deferred, all the pages in the range are disclaimed. Presuming there is no error, all the memory will be faulted and zero filled the next time it is referenced. “Read” references to freed memory are medium severity errors, while “write” references always cause a system crash. If the disposition of medium severity errors is set to cause a system crash, the system will crash on a “read” reference.

```
errctrl -c alloc.xmdbg deferred_free=<frequency>
```

Redzones for large allocations

This option sets the frequency of redzone page construction. This option is specific for allocations of a page or more. With default error disposition in effect any “read” references to redzone pages will cause an error log event, and “write” references will cause a system crash.

```
errctrl -c alloc.xmdbg redzone=<frequency>
```

VMM page state checks

This option sets the frequency at which xmfree() will check page protection settings, storage key bits and pin counts for memory being freed back to a heap. Not all errors in this area are fatal. For instance, a page that has higher than expected pin count at free time will waste pinned storage, but there are usually no fatal consequences of that.

When a page is returned that has a lower than expected pin count, has the wrong page protection settings, or has the wrong hardware storage key associated with it, the system will crash.

```
errctrl -c alloc.xmdbg vmmcheck=<frequency>
```

3.9.5 XMDBG tunables not affected by error check level

Memory leak percentage

This tunable sets the percentage of heap memory that can be consumed before an error is logged. This is specific to the heaps controlled by the component. Heaps that are controlled by other components are not affected.

For example, alloc.heap0 is a separate component that controls the heap used by the loader, and it uses a different percentage than the kernel_heap, which is controlled by alloc.xmdbg. Component level heaps created by heap_create() can be registered separately, and can be given different percentages. Refer to 3.9.7, “Heap registration for individual debug control” on page 137 for information about the individual heap registration mechanism.

```
errctrl -c alloc.xmdbg memleak_pct=<percentage>
```

> **Tip:** This tunable requires the user to make a judgment about how much storage should be consumed before a leak should be suspected. Users who do not have that information should not use the command. The default percentage is 100% (1024/1024).

Memory leak errors are classified as LOW_SEVERITY errors and the default disposition is to ignore them. The error disposition for low severity errors can be modified to log an error or to cause a system crash. This tunable can be seen in **`KDB`** by using the **`xm -Q`** command. The field Ratio of memory to declare a memory leak shows the memory leak percentage.

Memory leak count

This tunable sets an outstanding allocation limit for all the fragment sizes. This is intended as an aid in catching memory leaks that are very slow-growing. If the total number of outstanding allocations of any fragment size grows beyond this limit, an error is logged. For example, an

Archived

error occurs if the limit is set to 20,000, and 20,001 allocations are outstanding for any of the fragment sizes. This error is classified as a LOW_SEVERITY error, and the default disposition for the error is to ignore it. The default value of this setting is -1 meaning no check is made. This limit must be set to a positive value (=< 1024) by the operator to cause the check to be made.

```
errctrl -c alloc.xmdbg memleak_count=<num>
```

This tunable can be seen in **KDB** by using the **xm -Q** command. The field Outstanding memory allocations to declare a memory leak shows the memory leak count.

Large allocation record keeping

This tunable sets the size of an allocation that we will always record. Very large allocations are frequently never freed, so this setting allows the operator to record all outstanding allocations that are greater than or equal to “minsize” bytes. The default value of this tunable is 0x1000000 bytes.

```
errctrl -c alloc.xmdbg minsize=<num>
```

This tunable can be seen in **KDB** by using the **xm –Q** command. The field Minimum allocation size to force a record for shows the large allocation record keeping.

Reset error log handling

This tunable avoids having the error log become flooded. Each subcomponent of the alloc component will only record up to 200 errors in the error log before reaching a threshold. If the 200 log limit is reached and the count is not reset, error logging by the component will not resume until after a partition reboot.

```
errctrl -c alloc.xmdbg reset_errlog_count
```

Deferral count

This tunable is the total number of pages that are deferred before xmalloc() recycles deferred storage back to a heap. Deferring the freeing of storage for a very long time can result in fragmented heaps that result in allocation failures for large requests. Xmalloc supports setting this option to -1 which causes xmalloc() to defer reallocation as long as possible. This means the heap is exhausted before memory is recycled. On AIX V6.1, the default value is 0x4000 deferrals.

```
errctrl –c alloc.xmdbg deferred_count=<num>
```

This tunable can be seen in **KDB** using the **xm –Q** command. The field Deferred page reclamation count shows the deferral count.

> **Note:** The alloc.xmdbg and alloc.heap0 components and their potential child components support a variety of tunables that can be changed as a group. Use the **errctrl** command using the **errcheckminimal**, **errchecknormal**, **errcheckdetail**, and **errchecklevel** subcommands.
>
> Changes to the alloc.xmdbg component apply to the kernel_heap, pinned_heap, and all heaps created by kernel subsystems via the heap_create() subroutine. alloc.xmdbg is the default xmalloc-related component, and other components are mentioned only for completeness.
>
> The alloc.heap0 component applies to the loader-specific heap that appears in the kernel segment.

3.9.6 KDB commands for XMDBG

- **xm -Q** will display the value of current system level tunables.
- **xm -H @<heap_name> -Q** will display the value of tunables for the heap <heap_name>.
- **xm -L 3 -u** will sort all the allocation records in the system by size, and finds records that have matching stack trace-backs. The command supports three levels of stack trace-back, and the -L option allows up to three levels to be specified. It displays a summary of each matching record by size.

Example 3-49 shows the total usage information about the kernel heap, and then information that includes three levels of stack trace. Only partial output is shown.

Example 3-49 Detailed kernel heap usage information

```
(0)> xm -L 3 -u
    Storage area................... F100060010000000..F100060800000000
    ............(34091302912 bytes, 520192 pages)
    Primary heap allocated size.... 351797248 (5368 Kbytes)
    Alternate heap allocated size.. 56557568 (863 Kbytes)
Max_req_size....... 1000000000000000 Min_req_size....... 0100000000000000
Size                Count Allocated from
---------------------------------------------------------------------------
Max_req_size....... 0100000000000000 Min_req_size....... 0000000010000000
Size                Count Allocated from
---------------------------------------------------------------------------
Max_req_size....... 0000000010000000 Min_req_size....... 0000000001000000
Size                Count Allocated from
---------------------------------------------------------------------------
000000000C000000        1           4664B8 .nlcInit+0000D4
                                    62AF30 .lfsinit+00018C
                                    78B6E4 .main+000120
Max_req_size....... 0000000001000000 Min_req_size....... 0000000000100000
Size                Count Allocated from
---------------------------------------------------------------------------
00000000108DD000       32           658B38 .geninit+000174
                                    45C9C0 .icache_init+00009C
                                    45D254 .inoinit+000770
0000000000900000        3           5906B4 .allocbufs+0000B0
                                    5920CC .vm_mountx+0000C8
                                    592BE4 .vm_mount+000060
0000000000800000        1           786334 .devsw_init+000050
                                    78B6E4 .main+000120
                                    34629C .start1+0000B8
            :
            :
```

Notice the following in the text:

- The text shows three levels of stack trace for kernel heap.

  There are 32 records where inoint() called icache_init(), which called geninit(), which in turn called xmalloc(). Total memory usage for these 32 calls is 0x108DD000 bytes.
- Similarly, there are 3 records where start1() called main(), which called devsc_init(), which in turn called xmalloc(). Total memory usage by all these 3 records is 0x800000 bytes.

The command **xm -H @<heap_name> -u** will show the total memory usage of the heap <heap_name>.

Archived

3.9.7 Heap registration for individual debug control

Kernel programmers can register the heap they created using heap_create() under the alloc.xmdbg component by using the following command:

```
errctrl -c alloc.xmdbg skhp_reg=<addr>
```

Where `<addr>` is the address of heap and needs to be found from **KDB**. In Example 3-49 on page 136, `my_heap` is an individual heap and is valid for registration under alloc.xmdbg. To find its address in kernel space, execute **dd my_heap** in **KDB**.

After heap is registered under alloc.xmdbg, it should be visible under its component hierarchy. Execute following command to see it:

```
errctrl -q -c alloc.xmdbg -r
```

The new child component represents the heap "my_heap" and can be controlled individually by using pass-through commands. For further information about this topic, refer to 3.9.4, "XMDBG tunables affected by error check level" on page 131, and to3.9.5, "XMDBG tunables not affected by error check level" on page 134.

Note that persistence (the **–P** flag of the **errctrl** command) is not supported with this mechanism because this new subcomponent will not exist at the next system boot.

Archived

Archived

A

AIX features availability

Table A-1 lists AIX RAS tools availability.

Table A-1 AIX RAS tools availability

Name	AIX version	Hardware required
Dynamic Logical Partitioning (DLPAR) (AIX Related to hardware)	5.1	POWER4,5
CPU Guard	5.2	POWER4
CPU Sparing	5.2	POWER6
Predictive CPU Deallocation and Dynamic Processor Deallocation	5.3	+2 CPU
Processor recovery and alternate processor	5.3	POWER6
Excessive Interrupt Disablement Detection	5.3 ML3	
Memory Page Deallocation	5.3 ML3	POWER5 and later
System Resource Controller (SRC)	All versions	
PCI hot plug management	5.1	
Reliable Scalable Cluster Technology (RSCT) Subsystem	All versions	
Synchronous Unrecoverable Errors (SUE)	5.2	POWER4 and later
Automated system hang recovery	5.2	
Run-Time Error Checking (RTEC)	5.3 TL3	
Kernel Stack Overflow Detection	5.3 TL5	
Kernel No-Execute Protection	5.3 TL5	
Extended Error Handling (EEH)	5.3 TL5	
Page space verification	6.1	
User Storage Keys	5.3 TL5	POWER6
Kernel Storage Keys	6.1	POWER6

© Copyright IBM Corp. 2008. All rights reserved.

Name	AIX version	Hardware required
Advanced First Failure Data Capture Features (FFDC)	5.3 TL3	
Traditional System Dump	All versions	
Firmware-Assisted Dump	6.1	The P6, 4 GB memory, selected disk
Live Dump and Component Dump	6.1	
dumpctrl command	6.1	
Parallel dump	5.3 TL5	
Minidump	5.3 ML3	
Trace (System trace)	All versions	
POSIX trace	6.1	
iptrace	5.2	
Component Trace facility (CT)	5.3 TL5	
Lightweight Memory Trace (LMT)	5.3 TL3	
ProbeVue	6.1	
Error Logging	All versions	
Concurrent update	6.1	
Core file control	All versions (1)	
Recovery Framework	6.1	
Virtual IP address support (VIPA)	5.1	
Multipath IP routing	5.1	
Dead Gateway Detection	5.1	
EtherChannel	5.1	
Hot-swap disk	5.1	
System backup (MKSYSB) and restore	All versions	
Alternate disk installation	All versions	
Network Install Manager (provisioning tool)	All versions	
LVM RAID options	All versions	
LVM Hot Spare Disk	All versions	
LVM Advanced RAID Support via dynamic volume expansion	5.1	
Enhanced Journaled File System	5.2	
GLVM	5.3 ML3	
MPIO	5.2	
Dynamic tracking of FC devices	5.2	

Archived

Name	AIX version	Hardware required
Electronic Service Agent	5.2 ML1	
Resource Monitoring and Control (RMC) Subsystem	5.1	
Topas	5.2	
The **raso** command	5.3 ML3	
Dynamic Kernel Tuning	5.3	
Partition mobility	5.3	POWER6
Live application mobility	5.3	

(1) lscore and chcore commands were added in 5.3

Archived

Archived

Abbreviations and acronyms

ACL	Access Control List
AIX	Advanced Interactive eXecutive
AMR	Authority Mask Register
API	Application Programming Interface
ARP	Address Resolution Protocol
ATS	Advanced Technical Support
BOS	Base Operating System
CPU	Central Processing Unit
CSM	Cluster Systems Management
CT	Component Trace
DLPAR	Dynamic LPAR
DMA	Direct Memory Access
DR	Disaster Recovery
FFDC	First Failure Data Capture
FRU	Field Replaceable Unit
GLVM	Geographical Logical Volume Manager
GPFS™	General Parallel File System™
HACMP	High Availability Cluster Multi-Processing
HACMP/XD	HACMP eXtended Distance
HMC	hardware Management Console
IBM	International Business Machines Corporation
IP	Internet Protocol
IPC	Inter-Process Communication
ITSO	International Technical Support Organization
JFS	Journaled File System
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LFS	Logical File System
LMT	Lightweight Memory Trace
LPAR	Logical Partition
LUN	Logical Unit Number
LVM	Logical Volume Manager
MAC	Media Access Control
MMU	Memory Management Unit
MPIO	Multi-Path Input/Output
NFS	Network File System
NIM	Network Installation Manager
ODM	Object Data Manager
PCI	Peripheral Component Interconnect
PCM	Path Control Module
PFT	Page Frame Table
PURR	Processor Utilization Resource Register
RAS	Reliability, Availability, Serviceability
RM	Resource Manager
RMC	Resource Monitoring and Control
RSCT	Reliable Scalable Clustering Technology
RTEC	Real Time Error Checking
SAN	Storage Area Network
SCSI	Small Computer System Interface
SDD	Subsystem Device Driver (Storage)
SEA	Shared Ethernet Adapter
SFDC	Second Failure Data Capture
SMIT	System Management Interface Tool
SMP	Symmetric Multi-Processing
SPOF	Single Point of Failure
SRC	System Resource Controller
SUE	System Unrecoverable Error
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TL	Technology Level
TLB	Transaction Look-aside Buffer
VGDA	Volume Group Descriptor Area
VIOS	Virtual I/O Server
VIPA	Virtual IP Address
VLAN	Virtual LAN
VMM	Virtual Memory Manager
VMX	Vector Multimedia eXtension
VPD	Vital Product Data
WLM	Workload Manager
WPAR	Workload Partition

Archived

© Copyright IBM Corp. 2008. All rights reserved.

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks publications

For information about ordering these publications, see "How to get IBM Redbooks publications" on page 146. Note that some of the documents referenced here may be available in softcopy only.

- *AIX 5L Differences Guide Version 5.2 Edition,* SG24-5765
- *IBM eServer Certification Study Guide - AIX 5L Performance and System Tuning*, SG24-6184
- *AIX Version 6.1 Differences Guide*, SC27-7559
- *Implementing High Availability Cluster Multi-Processing (HACMP) Cookbook*, SG24-6769
- *IBM System p5 Approaches to 24x7 Availability Including AIX 5L*, SG24-7196
- *NIM from A to Z in AIX 5L*, SG24-7296
- *AIX V6 Advanced Security Features Introduction and Configuration*, SG24-7430
- *Introduction to Workload Partition Management in IBM AIX Version 6*, SG24-7431
- *IBM System p Live Partition Mobility,* SG24-7460
- *AIX 5L Differences Guide Version 5.3 Edition,* SG24-7463
- *Advanced POWER Virtualization on IBM System p5: Introduction and Configuration*, SG24-7940
- *IBM eServer p5 590 and 595 System Handbook*, SG24-9119

Other publications

These publications are also relevant as further information sources:

- *AIX 5L Version 5.3 Commands Reference, Volume 1, a-c,* SC23-4888
- *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts,* SC23-4900
- *AIX 5L System Management Guide: Operating System and Devices*, SC23-5204
- *AIX Version 6.1 Commands Reference, Volume 4*, SC23-5246
- *AIX Version 6.1 General Programming Concepts: Writing and Debugging Programs*, SC23-5259
- *Reliable Scalable Cluster Technology: Administration Guide*, SA22-7889

Archived

© Copyright IBM Corp. 2008. All rights reserved.

Online resources

These Web sites are also relevant as further information sources:

- POWER6 Availability white paper

  `http://www-05.ibm.com/cz/power6/files/zpravy/WhitePaper_POWER6_availabilty.PDF`

- Dynamic CPU deallocation article in IBM Systems Journal

  `http://www.research.ibm.com/journal/sj/421/jann.html`

- AIX V5.3 documentation page

  `http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp`

- IBM Systems and Technology Group Technotes (FAQs): Minidump

  `http://igets3.fishkill.ibm.com/DCF/isg/isgintra.nsf/all/T1000676?OpenDocument&Highlight=0,minidump`

- AIX Security

  `http://www.redbooks.ibm.com/redbooks/pdfs/sg247430.pdf`

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy books, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Numerics

64-bit kernel 58–59, 63, 83, 101
64K page 132

A

action block 123
Active dead gateway detection 38
AIX 5.3 126, 132–133
AIX capability 56
AIX mobility features 52
AIX Reliability, Availability, and Serviceability (RAS) components hierarchy 56
AIX service 57
AIX system 57–58, 60–61
  dump 60
  outage 8
AIX V5.3
  ML3 61
  TL3 8
  TL5 8, 56
AIX V6.1 9, 55
  default checking level 126
  release 83
AIX Version 139
  5.3 107
  5.3 ML3 8
  V5.3 ML3 56
AIX6.1 XMDBG 126
alloc.xmdb g 126–128
  alloc_record 131
  alloc_trailer 132
  component 135, 137
  def_free_frag 133
  deferred_count 135
  deferred_free 134
  doublepage_promote 133
  errcheckdetail 129
  errchecklevel 130
  errcheckon 126
  large_trailer 132
  medsevdisposition 132
  memleak_count 135
  memleak_pct 134
  minsize 135
  promote_all 133
  redzone 134
  reset_errlog_count 135
  RTEC level 127
  ruin_all 131
  skhp_reg 137
  small_trailer 132
  vmmcheck 134
Alternate disk installation 40–41
alternate processor recovery 13
AMR value 99
API 98
Authority Mask Register (AMR) 21
autonomic computing 11
aware component 64

B

base kernel 71–72

C

Capacity on Demand 14
cluster security services 16
commands
  /usr/sbin/bosboot 19
  alog 31
  cfgmgr 31
  chcore 35
  chdev 12
  chps 21
  chvg 43
  ctctrl 30
  curt 28
  dmpuncompress 27
  dumpctrl 24, 26
  errctrl 13
  errdemon 31
  errlast 31
  errlogger 31
  errpt 31
  filemon 49
  fileplace 49
  ioo 49–50
  iostat 49
  ipreport 28
  iptrace 28
  livedumpstart 26
  lparstat 49
  lsconf 49
  lscore 35
  lvmstat 49
  mdmprpt 62
  mkps 21
  multibos 41
  netstat 49
  nfso 50
  no 49–50
  perfpmr 49
  pprof 28
  probevue 30
  raso 50–51
  sar 49
  schedo 50
  snap 24, 61
  splat 28

© Copyright IBM Corp. 2008. All rights reserved.

sysdumpdev 27
tprof 28
trace 28
trcctl 28
trcevgrp 28
trcrpt 28
tunchange 49
tuncheck 49
tundefault 49
tunrestore 49
tunsave 49
vmo 50
vmstat 49
component dump 25, 64, 68, 140
Component Trace (CT) 23, 29, 56, 140
compressed dump file 27
Concurrent update 55, 71, 140
command 72
package file 73
terminology 71
Continuous availability 1, 4–5, 8
core file 35
core file control 35
CPU deallocation 13
CPU Guard 12
CPU_ALLOC_ABORTED entry 59

D

data storage interrupt 21
DB2 core 100
Dead gateway detection 36–37
default amount 57–58
device driver 29
DGD 38
Disaster recovery (DR) 2–3
DLPAR 12
dynamic kernel tuning 50
dynamic processor deallocation 13
Dynamic tracking of fibre channel devices 47

E

Efix Attribute 74, 76
Efix Description 75
efix file 74–75
Total number 75
EFIX Id 73
efix installation 74–75
EFIX label 73–74, 76
EFIX package 73–75
Efix State 75
EFIXES Menu 76
Electronic Service Agent 48
Enhanced Journaled File System (JFS2) 46
entry field 57, 68, 73, 76
select values 73, 77
ERRCHECK_MAXIMAL 126
ERRCHECK_MAXIMAL Disabled 58
error detection 8
error logging 30
EtherChannel 36, 38–39
Ethernet link aggregation 17
Extended Error Handling (EEH) 20, 139

F

FFDC 7–8, 29
FFDC capability 7
Field Replaceable Unit (FRU) 7
firmware-assisted system 24
First Failure Data Capture (FFDC) 7, 12, 21, 23, 56–57, 126
format minidumps 62

G

gated 38
Geographic Logical Volume Manger (GLVM) 44
GLVM 44

H

hardware key
mapping 83
hardware keysets 98–99
heap registration 134
heap size 26
hot-swap disks 40

I

IBM System p
p5-595 server 7
inetd.conf 33
int argc 93–94
iptrace 29

J

Journaled File System (JFS) 46

K

kdb context 71
kernel component 29
kernel extension 29, 64, 71–72
Exporting file 93
kernel key 79
kernel keys 22
kernel no-execute protection 20
kernel recovery framework 19
kk VMM_PMAP 109

L

LDAP 51
Lightweight Memory Trace (LMT) 8, 23, 30, 55, 57–58, 140
List EFIXES 72
Live application mobility 52
live dump 24–25, 64–65, 68–69
Live partition mobility 52
LMT buffer 59

LMT file 60
Logical Volume Manager (LVM) 44
ls core 105–106
LUN subcomponent 71
LVM 17
LVM mirroring 17
LVM quorum 44

M

Machine Status Table (MST) 63, 109
memory page deallocation 14
Memory Persistent Deallocation 14
minidump 27, 55, 61–62
minidump entry 62
MINIDUMP_LOG 62
MPIO 17, 46
multibos utility 41
Multipath I/O (MPIO) 46
Multipath IP routing 36–37

N

Network Installation Manager (NIM) 41–42
NIM server 41
NVRAM 27

O

operating system 7, 9
  reboot 102
operating system error 8

P

parallel dump 27
partition reboot 135
Passive dead gateway detection 38
path control module (PCM) 46
pax file 61
PCI hot plug management 15
performance monitoring 48
PFT entry 110
POSIX trace 29, 55, 140
POWER5 system 59
probe actions 113
probe event 113
probe location 113
Probe manager 113
probe point 113
probe point specification 115–116
probe type 113
probevctrl command 114
ProbeVue 30, 55
  action block 115–116
  dynamic tracing benefits and restrictions 111
  interval probe manager 118
  introduction 111
  predicate 116
  probe actions 113
  probe event 113
  probe location 113
  Probe manager 117
  probe manager 113
  probe point 113
  probe point specification 115–116
  probe type 113
  probevctrl command 114
  probevue command 114
  ProbeVue example 119
  ProbeVue Terminology 113
  system call probe manager 117
  user function probe manager 117
  Vue functions 118
  Vue overview 114
  Vue program 114
  Vue Programming Language 113
  Vue script 114–115
probevue command 114
processor utilization register (PURR) 28
protection gate
  corresponding kernel keys 109

R

RAID 17, 43
rare event 57
read/write access 98
read/write attribute 107
Reboot Processing 75
Redbooks Web site 146
  Contact us xii
Redzone 128–129
Reliability, Availability, and Serviceability (RAS) 22, 51, 56, 58, 60
Reliable Scalable Cluster Technology (RSCT) 15, 139
remaining CPU 62
resource manager 16
Resource Monitoring and Control (RMC) 15, 141
RMC 16
routed 38
RSCT 16
RTEC level 127, 130
  alloc.xmdbg 130
Run time error checking
  xmalloc debug enhancement 126
Run-Time Error Checking (RTEC) 8, 20, 23, 56, 126, 139

S

SC_AIX_UKEYS 107
SCSI disc 65
Second Failure Data Capture (SFDC) 58, 102
SIGSEGV 106
single points of failure (SPOFs) 3
smitty menu 72
SMP 12
Special Uncorrectable Error 18
SRC 14
stack overflow 20
Storage key 55, 79–80
storage key 19, 21
subcomponent 64–65

subsequent reboot 71, 133
SUE 18
Symbolic constant 121
syslogd 32
System backup (mksysb) 40
system call 91, 96
  return value 123
system crash 23, 61
system dump 23
System p
  hardware 8
  hardware aspect 8
  server 6
System p hardware 8
system reboot 102, 126
System Resource Controller (SRC) 14, 29, 139
system trace 27

T

target thread 121
  Current errno value 121
  effective user Id 121
tmp/ibmsupt directory 61
topas 49
Tracing Facility
  dynamic tracing benefits and restrictions 111
  interval probe manager 118
  predicate 116
  probe action block 115–116
  probe actions 113
  probe event 113
  probe location 113
  Probe manager 113, 117
  probe point 113
  probe point specification 115–116
  probe type 113
  probevctrl command 114
  ProbeVue 111
  probevue command 114
  ProbeVue example 119
  ProbeVue Terminology 113
  system call probe manager 117
  user function probe manager 117
  Vue functions 118
  Vue Overview 114
  Vue program 114
  Vue Programming Language 113
  Vue script 114–115
trcrpt command 57, 60–61
  M parameter 60

U

UK_RW access 102
uncompressed header 62
user key 100
user keys 22

V

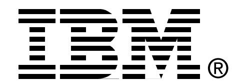
VGDA 44
VGSA 44
VIOS 17
VIPA 37
Virtual I/O Server 17
Virtual IP address 37
Virtual IP address support 36
Virtual SCSI 65, 71
Volume Group Descriptor Area 44
Volume Group Status Area 44
Vue Overview 114
Vue program 114
Vue Programming Language 113
Vue script 114–115, 122

X

xmalloc allocator 125
xmalloc debug
  tunable 128

Archived

Archived



IBM AIX Continuous Availability Features

Learn about AIX V6.1 and POWER6 advanced availability features

View sample programs that exploit storage protection keys

Harden your AIX system

This IBM Redpaper describes the continuous availability features of AIX Version 6, Release 1. It also addresses and defines the terms Reliability, Availability, and Serviceability (RAS) as used in an IT infrastructure. It touches on the global availability picture for an IT environment in order to better clarify and explain how AIX can improve that availability. The paper is intended for AIX specialists, whether customers, business partners, or IBM personnel, who are responsible for server availability.

A key goal of AIX development is to improve overall system serviceability by developing problem determination tools and techniques that have minimal impact on a live system; this document explains the new debugging tools and techniques, as well as the kernel facilities that work in conjunction with new hardware, that can help you provide continuous availability for your AIX systems.

The paper provides a broad description of the advanced continuous availability tools and features on AIX that help to capture software problems at the moment they appear, with no need to recreate the failure. In addition to software problems, the AIX kernel works closely with advanced hardware features to identify and isolate failing hardware and replace hardware components dynamically without bringing down the system.

The tools discussed include Dynamic Trace, Lightweight Memory Trace, Component Trace, Live dump and Component dump, Storage protection keys (kernel and user), Live Kernel update, and xmalloc debug.

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

REDP-4367-00