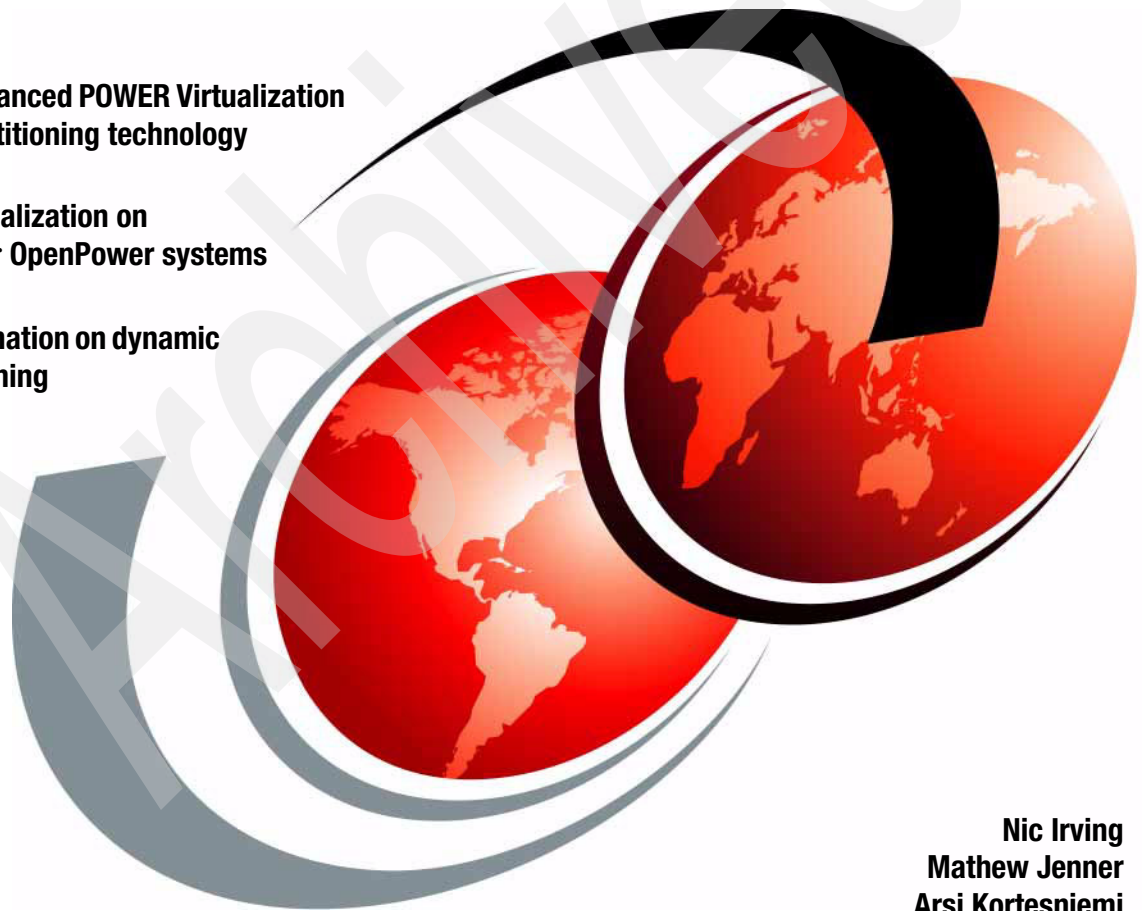


Partitioning Implementations for IBM **@**server p5 Servers

Discusses Advanced POWER Virtualization
and Micro-Partitioning technology

Describes virtualization on
IBM **@**server OpenPower systems

Includes information on dynamic
logical partitioning



Nic Irving
Mathew Jenner
Arsi Kortensniemi



International Technical Support Organization

**Partitioning Implementations for
IBM @server p5 Servers**

February 2005

Archived

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

Third Edition (February 2005)

This edition applies to IBM @server p5 servers for use with AIX 5L Version 5.3 (product number 5765-G03).

© Copyright International Business Machines Corporation 2003, 2004, 2005. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	ix
Tables	xi
Examples	xiii
Notices	xv
Trademarks	xvi
Preface	xvii
The team that wrote this redbook	xvii
Become a published author	xviii
Comments welcome	xix
Summary of changes	xxi
February 2005, Third Edition	xxi
October 2003, Second Edition	xxii
January 2003, First Edition	xxiii
Chapter 1. Logical partitioning primer	1
1.1 An introduction to partitioning	2
1.1.1 Basic types of partitioning	3
1.1.2 Partition isolation and security	3
1.2 Introduction to Micro-Partitioning and Virtualization	4
1.2.1 Micro-Partitioning	4
1.2.2 Virtual Ethernet	5
1.2.3 Virtual I/O Server	5
1.2.4 Advanced POWER Virtualization technologies	6
1.2.5 Advanced OpenPower Virtualization technologies	7
1.2.6 Obtaining the Virtual I/O Server and Partition Load Manager	8
1.3 Partitioning on eServer p5 and OpenPower servers	10
1.4 IBM Hardware Management Console	11
1.5 IBM @server Information Center	14
1.6 LPAR Validation Tool	15
1.7 Operating system support	16
1.7.1 AIX	19
1.7.2 Linux	22
Chapter 2. Partitioning implementation	25

2.1 Partitioning enablers	26
2.1.1 Hardware	26
2.1.2 Firmware	29
2.2 Partition resources	32
2.2.1 Partition and system profiles	33
2.2.2 Processors	34
2.2.3 Memory	36
2.2.4 Physical I/O slots	40
2.2.5 Virtual I/O	40
2.2.6 Minimum, desired, and maximum values	41
2.3 Resource planning using LPAR Validation Tool	42
2.3.1 System Selection dialog	43
2.3.2 Memory Specification dialog	45
2.3.3 LPAR Validation dialog	46
2.4 I/O device assignment considerations	46
2.4.1 Media devices	47
2.4.2 Boot device considerations	48
2.4.3 Network devices	52
2.4.4 Graphics console	52
2.4.5 High availability	52
2.5 LPAR limitations and considerations	53
Chapter 3. Basic partition management	57
3.1 Hardware Management Console	58
3.1.1 Managing I/O devices and slots	59
3.1.2 Managing memory	61
3.1.3 Managing processing power	62
3.1.4 Scheduling movement of resources	64
3.2 Advanced System Management Interface	64
3.2.1 Accessing the ASMI using a Web browser	65
3.2.2 Accessing the ASMI using the HMC	65
3.2.3 Network configuration	65
3.2.4 Service processor	67
3.2.5 Power/Restart control	67
3.3 Resetting a server	69
3.3.1 EEH adapters and partitioning	69
3.3.2 Restoring a server to factory settings	70
3.4 Partition Load Manager	71
3.4.1 Managing memory	76
3.4.2 Managing processors	76
3.4.3 Limitations and considerations	77
3.4.4 Installing Partition Load Manager	78
3.4.5 Querying partition status	78

3.4.6 Managing memory resource requests	79
3.4.7 Processor resources in a shared partition environment	81
Chapter 4. Virtualized resource management	83
4.1 Micro-Partitioning technology	84
4.1.1 Shared processor partitions	84
4.1.2 Processing units of capacity	86
4.1.3 Capped and uncapped mode	88
4.1.4 Virtual processors	89
4.1.5 Dedicated processors	91
4.1.6 Capped and uncapped processing units	93
4.1.7 Dynamic processor deallocation and sparing	96
4.2 Advanced Virtualization	97
4.2.1 Virtual LAN	98
4.2.2 VLAN communication by example	99
4.3 Introduction to Virtual I/O Server	104
4.3.1 Shared Ethernet Adapter	105
4.3.2 Virtual SCSI	111
4.3.3 Limitations and considerations	116
4.4 Virtual I/O Server and virtualization configuration	117
4.4.1 Using the command line interface	117
4.4.2 Managing hardware resources	120
4.4.3 Installing Virtual I/O Server	121
4.4.4 Basic configuration	125
4.4.5 Ethernet adapter sharing	126
4.4.6 Virtual SCSI disk	131
4.4.7 Defining the Virtual SCSI Server adapter on the HMC	132
4.4.8 Defining the Virtual SCSI Client adapter on the HMC	134
4.4.9 Creating the virtual target device on the Virtual I/O Server	135
4.4.10 Limitations and considerations	136
Chapter 5. Dynamic logical partitioning	139
5.1 Dynamic logical partitioning overview	140
5.1.1 Processor resources	141
5.1.2 Dynamic partitioning for Virtual Ethernet devices	146
5.1.3 Dynamic partitioning for Virtual SCSI devices	147
5.1.4 Capacity on Demand	147
5.2 The process flow of a DLPAR operation	148
5.3 Internal activity in a DLPAR event	152
5.3.1 Internal activity for processors and memory in a DLPAR event	152
5.3.2 Internal activity for I/O slots in a DLPAR event	154
5.4 DLPAR-safe and DLPAR-aware applications	155
5.4.1 DLPAR-safe	155

5.4.2 DLPAR-aware	156
5.5 Integrating a DLPAR operation into the application	157
5.5.1 Three phases in a DLPAR event.	157
5.5.2 Event phase summary	160
5.6 Script-based DLPAR event handling	160
5.6.1 Script execution environment	162
5.6.2 DLPAR script naming convention	167
5.7 DLPAR script subcommands	167
5.7.1 The scriptinfo subcommand	169
5.7.2 The register subcommand	172
5.7.3 The usage subcommand.	173
5.7.4 The checkrelease subcommand	175
5.7.5 The prerelease subcommand	176
5.7.6 The postrelease subcommand	178
5.7.7 The undoprerelease subcommand	179
5.7.8 The checkacquire subcommand	180
5.7.9 The preacquire subcommand	182
5.7.10 The postacquire subcommand	183
5.7.11 The undopreacquire subcommand	184
5.8 How to manage DLPAR scripts.	186
5.8.1 List registered DLPAR scripts	186
5.8.2 Register a DLPAR script	186
5.8.3 Uninstall a registered DLPAR script	187
5.8.4 Change the script install path	188
5.8.5 The drmgr command line options	188
5.8.6 Sample output examples from a DLPAR script.	190
5.9 API-based DLPAR event handling	194
5.9.1 The dr_reconfig system call	195
5.9.2 A sample code using the dr_reconfig system call	199
5.9.3 Sample output examples from a DLPAR-aware application	200
5.9.4 DLPAR-aware kernel extensions	205
5.10 Error handling of DLPAR operations	205
5.10.1 Possible causes of DLPAR operation failures.	205
5.10.2 Error analysis facilities	207
5.10.3 AIX error log messages when DLPAR operations fail.	212
Chapter 6. The POWER Hypervisor	217
6.1 Introduction	218
6.2 Hypervisor support	219
6.3 Hypervisor call functions	221
6.4 Micro-Partitioning technology extensions	227
6.5 Memory considerations	227
6.6 Performance considerations	229

Appendix A. Dynamic logical partitioning program templates	231
General information	232
Perl template	233
Korn shell template	245
DLPAR-aware application using a signal handler	257
How to compile and run the application	257
Appendix B. Dynamic logical partitioning output samples	273
Using the syslog facility	274
CPU addition	274
CPU removal	275
Memory addition	276
Memory removal	277
Using the AIX system trace facility	278
CPU addition trace output	278
CPU removal trace output	281
Memory addition trace output	285
Memory removal trace output	288
Using the AIX error log facility	291
Abbreviations and acronyms	295
Related publications	301
IBM Redbooks	301
IBM Redpapers	301
IBM Whitepapers	301
pSeries and eServer p5 publications	302
LPAR Validation Tool	302
Other publications	302
Online resources	303
How to get IBM Redbooks	303
Index	305

Figures

1-1	Advanced POWER Virtualization feature.	7
1-2	HMC connection	13
1-3	InfoCenter - Planning for AIX logical partitions	16
1-4	Mixed operating systems	17
2-1	HMC panel to enable the Virtualization Engine Technologies	30
2-2	POWER hypervisor	32
2-3	Partitions, partition profiles, and system profiles	34
2-4	Allow and disallow shared processor partition utilization authority	36
2-5	LPAR Validation Tool, creating a new partition	43
2-6	LPAR Validation Tool, System Selection dialog	44
2-7	LPAR Validation Tool, System Selection processor feature selection.	44
2-8	LPAR Validation Tool, Memory Specifications dialog	45
2-9	LPAR Validation Tool, slot assignments	46
3-1	Dual HMC configuration.	59
3-2	DLPAR Virtual Adapter menu	60
3-3	Virtual Adapter capabilities	61
3-4	Move memory resources - step 1	61
3-5	Move memory resources - step 2	62
3-6	Move processor resources.	63
3-7	Move processing units	63
3-8	ASMI network configuration, powered on	66
3-9	ASMI network configuration	67
3-10	ASMI, Power On/Off System	69
3-11	Partition Load Manager functionality	72
3-12	Comparison of features of PLM and hypervisor.	73
3-13	PLM overview	74
3-14	PLM resource distribution for partitions	75
3-15	PLM, Show LPAR Statistics.	78
4-1	Create Logical Partition Profile - shared processors	85
4-2	Choose desired, minimum, and maximum processing units	87
4-3	Processing units of capacity	88
4-4	Specify processing sharing mode and weight	89
4-5	Specify number of virtual processors.	90
4-6	Create Logical Partition Profile, dedicated processors	91
4-7	Allow shared processor pool statistics access to a partition	92
4-8	Allow idle processors to be shared	93
4-9	Distribution of capacity entitlement on virtual processors	94
4-10	Shared capped processor partition utilization	95

4-11	Shared uncapped processor partition utilization	96
4-12	Example of a VLAN	98
4-13	VLAN configuration	100
4-14	logical view of an inter-partition VLAN	102
4-15	Connection to external network using AIX routing	105
4-16	SEA configuration	106
4-17	Multiple SEA configuration	108
4-18	Link Aggregation (EtherChannel) pseudo device	110
4-19	Virtual SCSI architecture overview	113
4-20	Logical Remote Direct Memory Access	114
4-21	Virtual SCSI device relationship on Virtual I/O Server	115
4-22	Virtual SCSI device relationship on AIX client partition	116
4-23	Hardware Management Console - create Virtual I/O server	122
4-24	Activate I/O_Server_1 partition	122
4-25	Selecting the profile	123
4-26	Choosing SMS boot mode	123
4-27	SMS menu	124
4-28	Finished Virtual I/O Server installation	125
4-29	Creating the trunk Virtual I/O Server	127
4-30	Virtual Ethernet Adapter Properties panel	128
4-31	Dynamically adding or removing virtual adapters to a partition	129
4-32	Example of an I/O server partition bridge	130
4-33	Virtual SCSI Adapter Properties panel on the IO Server	132
4-34	Virtual SCSI Adapter Properties panel on the client partition	134
5-1	Dynamic Logical Partitioning Processor Resources	141
5-2	Add Processor Resources	142
5-3	Advanced Processor Settings - Uncapped Mode	143
5-4	Remove Processing Units	145
5-5	Move Processing Units	146
5-6	Process flow of a DLPAR operation	149
5-7	Three DLPAR phases of a DLPAR event	158
5-8	A DLPAR script invoked by the drmgr command	162
5-9	DLPAR operation failed message	206
5-10	DLPAR operation failure detailed information	207
6-1	POWER Hypervisor on AIX 5L and Linux	219
6-2	lparstat -H command output	226
6-3	Logical Partition Profile Properties - current memory settings	228
6-4	lparstat -h 1 16 command output	229
6-5	lparstat -i command output	230

Tables

1-1	eServer p5 and OpenPower servers	10
1-2	Maximum number of processors, memory size, and partitions	10
1-3	Required Hardware Management Console	12
1-4	Operating systems supported functions.	18
1-5	AIX - supported features	19
2-1	Default memory block sizes	39
2-2	Reasonable settings for shared processor partitions.	54
4-1	Micro-Partitioning technology overview on @server p5 systems	86
4-2	Interpartition of VLAN communication	101
4-3	VLAN communication to external network	101
4-4	Main differences between EC and LA aggregation	109
4-5	Limitations for logical storage management	137
5-1	Applications that should be DLPAR-aware	156
5-2	Considerations during each event phase.	160
5-3	General DLPAR environment variables	164
5-4	Processor-specific DLPAR environment variables	165
5-5	Memory-specific DLPAR environment variables	166
5-6	General DLPAR output variables	166
5-7	DLPAR script subcommands	168
5-8	Required output name-value pairs for the scriptinfo subcommand	170
5-9	Optional output name-value pairs for the scriptinfo subcommand.	171
5-10	Required output name-value pair for the register subcommand	173
5-11	Required output name-value pair for the usage subcommand	174
5-12	The drmgr command line options	189
5-13	The dr_reconfig flag parameters	196
5-14	AIX error logs generated by DLPAR operations	211
5-15	General AIX error messages	212
5-16	drmgr-specific AIX error messages	212
5-17	DLPAR operation-specific AIX error messages	213
5-18	DLPAR resource-specific AIX error messages	214
6-1	Hypervisor calls	223

Examples

4-1	\$help command for an overview	118
4-2	\$help command for a specific command	119
5-1	Registered sample DLPAR scripts	171
5-2	Register a DLPAR script	186
5-3	Sample output: 2 GB memory addition	190
5-4	Sample output: 1 GB memory removal	191
5-5	Sample output: 1 CPU addition	192
5-6	Sample output: 2 CPU removal	193
5-7	The dr_reconfig system call usage	195
5-8	The dr_reconfig info parameter	196
5-9	Sample output: 1 GB memory addition	201
5-10	Sample output: 1 GB memory removal	202
5-11	Sample output: 2 CPU addition	203
5-12	Sample output: 1 CPU removal	204
5-13	START Trace panel	209
5-14	Generate a Trace Report panel	210
A-1	DR_UNSAFE_PROCESS	232
A-2	DLPAR script template: Perl	233
A-3	DLPAR script template: Korn shell	245
A-4	C language application with a signal handler	257
B-1	Sample syslog output for a CPU addition request	274
B-2	Sample syslog output for a CPU removal request	275
B-3	Sample syslog output for a memory addition request	276
B-4	Sample syslog output for a memory removal request	277
B-5	Sample system trace output for a CPU addition request	278
B-6	Sample system trace output for a CPU removal request	281
B-7	Sample system trace output for a memory addition request	285
B-8	Sample system trace output for a memory removal request	288
B-9	Sample AIX error log entry: DR_MEM_UNSAFE_USE	291
B-10	Sample AIX error log entry: DR_DMA_MIGRATE_FAIL	292
B-11	Sample AIX error log entry: DR_DMA_MAPPAER_FAIL	293

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

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


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in

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

Trademarks

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

@server®
@server®
Redbooks (logo) ™
ibm.com®
iSeries™
i5/OS™
pSeries®
zSeries®
AIX 5L™
AIX/L®

AIX®
AS/400®
Chipkill™
Electronic Service Agent™
Hypervisor™
HACMP™
IBM®
Micro-Partitioning™
OpenPower™
PowerPC Architecture™

PowerPC®
POWER™
POWER4™
POWER5™
Redbooks™
RS/6000®
S/370™
Versatile Storage Server™
Virtualization Engine™

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

This redbook provides a broad understanding of logical partitioning on the IBM @server p5 servers, focusing particularly on the increased function available when these servers are combined with AIX® 5L™ Version 5.3 and Advanced POWER™ Virtualization features. It also provides a discussion of available Linux® support and IBM @server OpenPower™ systems. This redbook covers the following subject areas:

- ▶ Advanced POWER Virtualization
- ▶ Micro-Partitioning™ technology
- ▶ Virtual I/O
 - Virtual SCSI
 - Virtual Ethernet
- ▶ Dynamic logical partitioning implementation

The audience for this redbook are technical support specialists, customers, and business partners.

The team that wrote this redbook

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

Nic Irving is a System Administrator at Computer Sciences Corporation in Australia. He has nine years of experience in the AIX field. His areas of expertise include SP, pSeries®, SAN, and clustering.

Matthew Jenner is a Technical Solutions Architect in Australia. He has 14 years of experience in the Midrange UNIX® and mainframe fields. He has worked at IBM® for eight years. His areas of expertise include the various software and hardware components of the pSeries and RS/6000® product range. He has also written about IBM @server pSeries server consolidation.

Arsi Kortensniemi is an advisory IT specialist, and he works at IBM Systems and Technology Group as Technical Sales Support in Finland. He has 12 years of experience in the IT field and has worked at IBM for four years. His areas of expertise include pSeries hardware, AIX, storage, and hardware architecture planning. He has also practical experience in implementing pSeries dynamic partitioning into various software and database environments.

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

ITSO, Austin Center

Scott Vetter

IBM Austin

Nia Kelley, Luke Browning, Andy McLaughlin, Duke Paulsen, Carolyn Scherrer, Jeffrey George, Bill Dempwolf

IBM Poughkeepsie

Christopher V Derobertis

IBM La Gaude, France

Bruno Blanchard

IBM Hannover, Germany

Peter Domberg

IBM Seoul, Korea

TaiJung Kim

IBM Beijing, China

Wei Ding

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:


redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 905
11501 Burnet Road
Austin, Texas 78758-3493

Summary of changes

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

Summary of Changes
for SG24-7039-02
for Partitioning Implementations for IBM server p5 Servers
as created or updated on October 13, 2005.

February 2005, Third Edition

This revision reflects the addition, deletion, or modification of new and changed information described below. With the popularity of LPAR, and the platforms now managed by the HMC, this publication is now more focused on the p5 aspects of LPAR.

New information

The following chapters are new:

- ▶ Chapter 3, “Basic partition management” on page 57
- ▶ Chapter 4, “Virtualized resource management” on page 83
- ▶ Chapter 6, “The POWER Hypervisor” on page 217

Changed information

The following chapters were rewritten to cover new features:

- ▶ Chapter 1, “Logical partitioning primer” on page 1
- ▶ Chapter 2, “Partitioning implementation” on page 25
- ▶ Previously Chapter 3, Chapter 5, “Dynamic logical partitioning” on page 139.
- ▶ Appendix A, “Dynamic logical partitioning program templates” on page 231
- ▶ Appendix B, “Dynamic logical partitioning output samples” on page 273.

October 2003, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described in the sections that follow.

New information

The following chapters are new:

- ▶ Chapter 10, “Dynamic reconfiguration using DLPAR scripts”
- ▶ Chapter 11, “Resource sets”
- ▶ Chapter 12, “Autonomic applications”
- ▶ Appendix D, “Using the Job Scheduling Console”
- ▶ Appendix E, “Advanced DLPAR script examples”
- ▶ Appendix F, “Autonomic application example”

Changed information

The following chapters were rewritten in order to cover new features, enhancements, and usage examples provided by the latest products:

- ▶ Chapter 4, “HMC graphical user interface”
- ▶ Chapter 9, “DLPAR operation using a command line interface”

Unchanged information

The following chapters and appendixes are unchanged but were reviewed:

- ▶ Chapter 1, “Logical partitioning overview”
- ▶ Chapter 2, “Partitioning implementation on pSeries servers”
- ▶ Chapter 3, “Dynamic logical partitioning”
- ▶ Chapter 5, “Basic managed system operation tasks”
- ▶ Chapter 6, “Creating and managing partitions”
- ▶ Chapter 7, “Installing and migrating AIX in a partitioned environment”
- ▶ Chapter 8, “DLPAR operation using graphical user interface”
- ▶ Appendix A, “Test environment”
- ▶ Appendix B, “Dynamic logical partitioning program templates”
- ▶ Appendix C, “Dynamic logical partitioning output samples”

January 2003, First Edition

The first version of this book was written by the following authors:

Keigo Matsubara, Akichika Ozeki, Erlander Lo, Deniz S. Erguvan, Jennifer Davis, Theeraphong Thitayanun, Viraf Patel

The following list shows contributors for the first version of this book:

IBM Austin

Andy McLaughlin, Ann Wigginton, Bob Foster, Bob Minns, Carolyn Scherrer, Christopher Chan, David Sheffield, Dave Willoughby, Duke Paulsen, Edward Shvartsman, Jane Chilton, John O'Quin, John Purcell, Julie Craft, Kanisha Patel, Larry Amy, Luke Browning, Mark Rogers, Michael Mall, Michael S. Williams, Minh Nguyen, Paul B. Finley, Randy Swanberg, Richard Cutler, Steven Molis, Susan Caunt, Trish Pierce, Truc Nguyen, and Walter Lipp

IBM Philadelphia

Rob Jackard

IBM Poughkeepsie

Michael Schmidt and Ron Goering

Logical partitioning primer

This chapter introduces the concepts and terminology that are necessary to understand the logical partitioning implementation on the IBM eServer p5 and OpenPower servers. It discusses the following topics:

- ▶ 1.1, “An introduction to partitioning” on page 2
- ▶ 1.2, “Introduction to Micro-Partitioning and Virtualization” on page 4
- ▶ 1.3, “Partitioning on eServer p5 and OpenPower servers” on page 10
- ▶ 1.4, “IBM Hardware Management Console” on page 11
- ▶ 1.5, “IBM ^ Information Center” on page 14
- ▶ 1.6, “LPAR Validation Tool” on page 15
- ▶ 1.7, “Operating system support” on page 16

If you are a system administrator who has responsibility for managing partitioning-capable pSeries servers, it is imperative that you become familiar with the aspects described in this chapter before you run the system in a logical partitioned environment.

1.1 An introduction to partitioning

There is a strong demand for high-end systems that can provide greater flexibility. In particular, system administrators want the ability to subdivide high-end systems into smaller partitions that are capable of running a version of an operating system or a specific set of application workloads.

IBM initially started work on partitioning in S/370™ mainframe systems in the 1970s. Since then, logical partitioning (LPAR) on IBM mainframes (now called IBM @server zSeries®) has evolved from a predominantly physical partitioning scheme, based on hardware boundaries, to one that allows for virtual and shared resources with dynamic load balancing. In 1999, IBM implemented LPAR support on the AS/400® platform (now called iSeries™). In 2000, IBM announced the ability to run the Linux operating system in an LPAR on a zSeries server, followed by the pSeries and iSeries platforms.

Continuing the evolution of partitioning technology on pSeries servers, the IBM @server p5 and OpenPower extends its capabilities by further improving flexibility in partition usage. There are now two types of partitions in the IBM @server p5 and OpenPower. Partitions can have dedicated processors, or they can have virtualized processors from a single pool of shared physical processors. Sharing a pool of virtualized processors is known as *Micro-Partitioning technology*. Both types of partitions can coexist at the same time in the same system.

Ethernet and SCSI I/O devices also have been virtualized enabling these resources to be shared by multiple partitions. The advantages of this technology include allocations of smaller resource units, more partitions, and higher, more efficient resource utilization.

IBM @server p5 and OpenPower systems have the ability to use Virtual Ethernet and Virtual SCSI, although they can use the physical resources, if desired, or a mix of the two.

For more information about virtualization of resources, refer to Chapter 4, “Virtualized resource management” on page 83.

1.1.1 Basic types of partitioning

This publication refers to various partitioning mechanisms. The following are some terms and definitions that this book uses:

► **Building block**

A collection of system resources, such as CPUs, memory, and I/O connections. These may be physically packaged as a self-contained symmetric multiprocessing (SMP) system (rack-mounted or desk-side) or as boards within a larger multiprocessor system. There is no requirement for the CPUs, memory, and I/O slots to occupy the same physical board within the system, although they often do.

► **Physical partition**

One or more building blocks linked together by a high-speed interconnect. Generally, the interconnect is used to form a single, coherent memory address space. In a system that is only capable of physical partitioning, a partition is a group of one or more building blocks that is configured to support an operating system image. Other vendors may refer to physical partitions as *domains* or *nPartitions*. The maximum number of physical processors in a POWER5™ system at the time of the writing of this book is 64.

► **Logical partition**

A subset of logical resources that are capable of supporting an operating system. A logical partition consists of CPUs, memory, and I/O slots that are a subset of the pool of available resources within a system.

► **Dedicated processor partition**

A logical partition whose CPU resources are dedicated to the LPAR along with the memory and I/O slots. CPU idle time cannot be used by other LPARs.

► **Shared processor partition**

Using Micro-Partitioning technology, physical processors are divided into virtual processors that are shared in a pool between one or more LPARs. The LPARs using the shared pool can be a mix of operating systems.

Note: The major difference between partitioning types is the granularity and flexibility that is available for allocating resources to an operating system image. Logical partitions have finer granularities than physical partitions, while Micro-Partitioning technology allows for even smaller allocations of resources.

1.1.2 Partition isolation and security

From a functional point of view, applications run inside partitions in the same way that they run on a stand-alone @server p5 and OpenPower server. There are no

issues when moving an application from a stand-alone server to a partition. The design of a partitioning-capable server is such that one partition is isolated from software that is running in the other partitions, including protection against natural software defects and even deliberate software attempts to break the partition barriers. It has the following security features:

- ▶ **Protection against inter-partition data access**

The design of partitioning-capable @server p5 and OpenPower servers prevents any data access between partitions, other than using shared networks. This design isolates the partitions against unauthorized access across partition boundaries.

- ▶ **Unexpected partition crash**

A software failure within a partition should not cause any disruption to the other partitions. Neither an application failure nor an operating system failure inside a partition interferes with the operation of other partitions.

- ▶ **Denial of service across shared resources**

The design of partitioning-capable @server p5 and OpenPower servers prevents partitions from making extensive use of a shared resource so that other partitions using that resource become starved. This design means that partitions sharing the same peripheral component interconnect (PCI) bridge chips, for example, cannot occupy the bus indefinitely.

With partition isolation and security, you can consolidate applications safely within partitions in partitioning-capable @server p5 and OpenPower servers without compromising overall system security.

1.2 Introduction to Micro-Partitioning and Virtualization

The following sections introduce the basic concepts of Micro-Partitioning as well as the Virtualization capabilities of the @server p5 and OpenPower servers. You can find a more detailed discussion about using @server p5 servers and AIX 5L Version 5.3 in Chapter 4, “Virtualized resource management” on page 83.

1.2.1 Micro-Partitioning

The Micro-Partitioning model offers a virtualization of system resources. In POWER5 processor-based systems, physical resources are abstracted into virtual resources that are available to partitions. This sharing method is the primary feature of this new partitioning concept, and it happens transparently.

POWER5 Micro-Partitioning technology specifies processor capacity in processing units. One processing unit represents 1% of one physical processor.

1.0 represents the power of one processor. A partition defined with 220 processing units is equivalent to the power of 2.2 physical processors. Creating a partition using Micro-Partitioning technology, the minimum capacity is 10 processing units, or 1/10 of a physical processor. A maximum of 10 partitions for each physical processor may be defined, but on a loaded system the practical limit is less. The practical limit to the number of partitions is based on available hardware and performance objectives.

Micro-Partitions can also be defined with capped and uncapped attributes. A capped Micro-Partition is not allowed to exceed the defined capacity. A configuration flag inside the Hardware Management Console (HMC) menus determines whether the capacity is capped. An uncapped partition, however, is allowed to consume additional capacity with fewer restrictions. Uncapped partitions can be configured to the total idle capacity of the server or to a percentage of the total idle capacity.

1.2.2 Virtual Ethernet

Virtual Ethernet enables inter-partition communications without a dedicated physical network adapter. With this feature, you can define in-memory point-to-point connection between partitions. These connections have the same characteristics as a high-bandwidth Ethernet network and support multiple networking protocols, such as IPv4, IPv6, and ICMP.

1.2.3 Virtual I/O Server

The Virtual I/O Server is a special-purpose partition that provides virtual I/O resources to client partitions. The Virtual I/O Server owns the real resources that are shared with other clients. With Virtual I/O technology, you can assign a physical adapter to a partition to be shared by one or more partitions, enabling clients to minimize their number of physical adapters. You can use the Virtual I/O Server to reduce costs by eliminating the requirement that each partition has a dedicated network adapter, disk adapter, and disk drive.

To ensure stable performance, it is preferable to use the Virtual I/O server in a partition with dedicated resources.

The following sections discuss the two major functions that the Virtual I/O Server provides.

Shared Ethernet Adapter

A Shared Ethernet Adapter (SEA) is a new service that acts as a Layer 2 network switch to route network traffic from a Virtual Ethernet to a real network adapter. The SEA must run in a Virtual I/O Server partition.

The advantage of the SEA is that partitions can communicate outside the system without having a physical network adapter attached to the partition. Up to 18 VLANs can be shared on a single network interface. The amount of network traffic will limit the number of client partitions that are served through a single network interface.

Virtual SCSI

Access to real storage devices is implemented through the Virtual SCSI services, a part of the Virtual I/O Server partition. Logical volumes that are created and exported on the Virtual I/O Server partition are shown at the virtual storage client partition as a SCSI disk. All current storage device types such as SAN, SCSI, and RAID are supported.

The Virtual I/O server supports logical mirroring and RAID configurations. Logical volumes created on RAID or JBOD configurations are bootable, and the number of logical volumes is limited to the amount of storage available and architectural limits of the Logical Volume Manager (LVM).

Resources removed from a partition are marked as free (free resources) and are owned by the global firmware of system. You can consider these resources as kept in the free resource pool. You can add free resources to any partition in a system as long as the system has enough free resources.

Note: The Shared Ethernet Adapter and Virtual SCSI server functions are provided in the Virtual I/O Server that is included in the Advanced POWER Virtualization feature, an additional feature of @server p5 systems.

1.2.4 Advanced POWER Virtualization technologies

This section provides information about the packaging information for the Advanced POWER Virtualization feature available on @server p5 systems. A more detailed description, including configuration, is provided in Chapter 4, “Virtualized resource management” on page 83.

This feature is a combination of the hardware capability inherent in POWER5 servers and the following components, available together as a single-priced feature:

- ▶ Firmware enablement for Micro-Partitioning technology
- ▶ Installation image for the Virtual I/O Server software which supports:
 - SEA
 - Virtual SCSI
- ▶ Partition Load Manager

Note: Virtual Ethernet and partitioning are available without this feature when the server is attached to an HMC.

Simultaneous Multi Threading (SMT) is available on the base hardware and requires no additional features. See 2.1.1, “Hardware” on page 26 for a discussion about SMT.

Figure 1-1 shows a detailed overview of the different parts of the hardware order that enable firmware and that include the software orders.

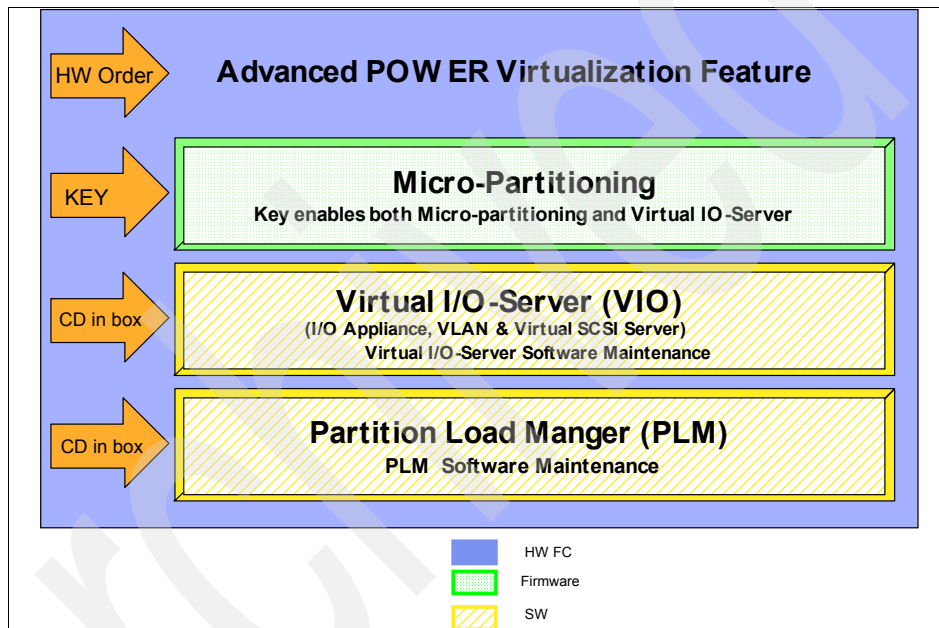


Figure 1-1 Advanced POWER Virtualization feature

1.2.5 Advanced OpenPower Virtualization technologies

The Advanced OpenPower Virtualization technologies are available on IBM @server OpenPower systems. The Advanced OpenPower Virtualization technologies are a combination of hardware and software features that include the following components and are available as separately priced options:

- ▶ Firmware enablement for the POWER Hypervisor™, which supports:
 - LPAR
 - Dynamic logical partitioning
 - Micro-Partitioning

- ▶ Installation image for the Virtual I/O Server software, which supports:
 - SEA
 - Virtual SCSI server

When you order the hardware feature directly from an IBM marketing representative and specify with the initial system order, the shipped firmware is activated to support Micro-Partitioning and the Virtual I/O Server. For upgrades, or any orders placed through IBM Business Partners, customers and IBM Business Partners must follow the instructions included in the option kit to receive an activation code that will enable the firmware.

These two technology offerings are ordered separately. While the Virtual I/O Server is not required for the POWER Hypervisor to operate, the POWER Hypervisor must be installed for Virtual I/O Server to function. While not required, we highly recommend the Virtual I/O Server for use with the POWER Hypervisor technology. It facilitates the sharing of physical I/O resources between logical partitions and can significantly increase system use and capability.

You can find additional information about Advanced OpenPower Virtualization technologies at:

<http://www.ibm.com/servers/eserver/linux/power/features/virtualization.html>

1.2.6 Obtaining the Virtual I/O Server and Partition Load Manager

Virtual I/O Server and Partition Load Manager (PLM) are licensed software components of the Advanced POWER Virtualization feature. They contain one charge unit per installed processor, including software maintenance. The initial software license charge for Virtual I/O Server and PLM is included in the price of the Advanced POWER Virtualization feature.

The related hardware features that include Virtual I/O Server and PLM are:

- ▶ 9110-510 (FC 7432)
- ▶ 9111-520 (FC 7940)
- ▶ 9113-550 (FC 7941)
- ▶ 9117-570 (FC 7942)
- ▶ 9119-590 (FC 7992)
- ▶ 9119-595 (FC 7992)
- ▶ 9123-710 (FC 1965)
- ▶ 9124-720 (FC 1965)

For each Virtual I/O Server license ordered, an order for one of the following software maintenance agreement (SWMA) is also required:

- ▶ One-year (5771-VIO)
- ▶ Three-year (5773-VIO)

The processor-based license enables you to install multiple Virtual I/O Server partitions on a single server to provide redundancy and to spread the Virtual I/O Server workload across multiple partitions.

Virtual I/O Server

The Virtual I/O Server resides in a POWER5 partition as a single-function appliance, which is created using the HMC. The Virtual I/O Server installation media ships with a POWER5 system configured with the Advanced POWER Virtualization feature. It supports network install (NIMOL from HMC) or CD installation, AIX 5L Version 5.3, SUSE LINUX Enterprise Server 9 for POWER, and Red Hat Enterprise Linux AS for POWER Version 3 as Virtual I/O clients.

The Virtual I/O Server provides the Virtual SCSI server and Shared Ethernet Adapter virtual I/O function to client Linux or AIX partitions. This @server p5 partition is not intended to run applications or for general user login.

Partition Load Manager

With PLM for AIX 5L, you can maximize the use of processor and memory resources on POWER5 servers that support dynamic logical partitioning. Within the constraints of a user-defined policy, resources are automatically moved to partitions with a high demand from partitions with a lower demand. Thus, you can fully use resources that would otherwise go unused.

PLM supports management of any dynamic LPAR that is running the following:

- ▶ AIX 5L Version 5.2 ML5200-04
- ▶ AIX 5L Version 5.3 or later

Features available in PLM are:

- ▶ Automated processor and memory reconfiguration
- ▶ Real-time partition configuration and load statistics
- ▶ Support for dedicated and shared processor partition groups
- ▶ Support for manual provisioning of resources
- ▶ Graphical user interface (Web-based System Manager)

For each Partition Load Manager V1.1 (5765-G31) license that you order, you must also order one of the following SWMAs:

- ▶ One-year (5771-PLM)
- ▶ Three-year (5773-PLM)

Software maintenance for PLM is priced on a per-processor basis, by processor group. Refer to 3.4, “Partition Load Manager” on page 71 for more detailed information about PLM.

1.3 Partitioning on eServer p5 and OpenPower servers

Table 1-1 provides the @server p5 and OpenPower server product range available at the time of the writing of this book.

Table 1-1 eServer p5 and OpenPower servers

Official product model name	Short product name	M/T-MDL
IBM @server p5 Model 510	p5-510	9110-510
IBM @server p5 Model 520	p5-520	9111-520
IBM @server p5 Model 550	p5-550	9113-550
IBM @server p5 Model 570	p5-570	9117-570
IBM @server p5 Model 590	p5-590	9119-590
IBM @server p5 Model 595	p5-595	9119-595
IBM @server OpenPower 710	OpenPower 710	9123-710
IBM @server OpenPower 720	OpenPower 720	9124-720

Note: Hereafter, this book uses the short product names.

Micro-Partitioning is supported across the entire POWER5 product line, from the entry to the high-end systems. Table 1-2 shows the maximum number of logical partitions and shared processor partitions that the different models support (provided that enough boot devices are available).

Table 1-2 Maximum number of processors, memory size, and partitions

Short product name	Max number of processors	Max Memory size	Max number of I/O drawers	Dedicated processor partitions	Shared processor partitions
p5-510	2	32 GB	0	2	20
p5-520	2	32 GB	4	2	20
p5-550	4	64 GB	8	4	40
p5-570	16	512 GB	20	16	160
p5-590	32	1 TB	7	32	254
p5-595	64	2 TB	11	64	254

Short product name	Max number of processors	Max Memory size	Max number of I/O drawers	Dedicated processor partitions	Shared processor partitions
OpenPower 710	2	32 GB	0	12	20
OpenPower 720	4	64 GB	2	4	40

The maximums stated in Table 1-2 are supported by the hardware. However, the practical limits based on production workloads can be significantly lower.

The logical partitioning concept and required tasks are basically similar on these partitioning-capable @server p5 and OpenPower server models. However, assigning I/O resources to partitions depends on the models, and there are significant differences. For the hardware model-specific information about the I/O resource assignments, see 2.4, “I/O device assignment considerations” on page 46.

1.4 IBM Hardware Management Console

To configure and manage logical partitions on a @server p5 and OpenPower server, you must have at least one Hardware Management Console (HMC). For more information about the HMC, refer to 3.1, “Hardware Management Console” on page 58.

Depending on the partitioning-capable server models, you can order the HMC as a feature code or a separate orderable product, as shown in Table 1-3 on page 12. The 7310 is for POWER5-based systems, and the 7315 is for POWER4™-based systems.

Table 1-3 Required Hardware Management Console

Short Product Name	HMC	Note
p5-595	7310-C04 or 7310-CR3	1
p5-590	7310-C04 or 7310-CR3	1
p5-570	7310-C04 or 7310-CR3	2
p5-550	7310-C04 or 7310-CR3	2
p5-520	7310-C04 or 7310-CR3	2
p5-510	7310-C04 or 7310-CR3	2
OpenPower 720	7310-C04 or 7310-CR3	2
OpenPower 710	7310-C04 or 7310-CR3	2
1. An HMC is required. 2. An HMC is required if the system is partitioned. The HMC is not required if the system is running as a full system partition.		

The HMC provides a set of functions that is necessary to manage the systems. The functions are LPAR, dynamic LPAR (DLPAR), Capacity on Demand without reboot, inventory and microcode management, and remote power control functions.

The HMC is a dedicated computer that provides a graphical user interface for configuring and operating servers that are functioning either in non-partitioned or in a full system partition. It is configured with a set of hardware management applications for configuring and partitioning the server. One HMC is capable of controlling multiple servers. At the time of the writing of this book, a maximum of 32 non-clustered servers and a maximum of 254 partitions are supported by one HMC. You can also add a second HMC for redundancy (see Figure 1-2 on page 13).

Note: It is not possible to connect POWER4 and POWER5 processor-based systems to the same HMC simultaneously.

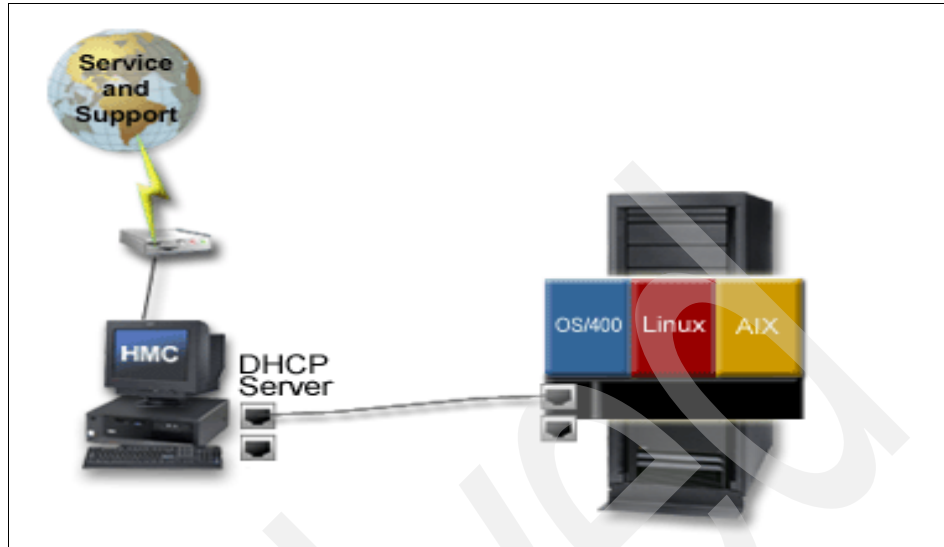


Figure 1-2 HMC connection

The HMC provides a set of functions that are necessary to manage partition configurations by communicating with the service processor, as follows:

- ▶ LPR control
- ▶ Capacity on Demand resource control
- ▶ Creation of partition and system profiles
- ▶ Boot, start, and stop actions for the system or individual partitions
- ▶ Displaying system and partition status

In a non-partitionable system, the operator panel displays the LED codes. In a partitioned system, the operator panel shows the word LPAR instead of any partition LED codes. Therefore, all LED codes for system partitions are displayed over the HMC.

- ▶ An imbedded DVD-RAM for creating and storing configuration backup information
- ▶ Cluster support when combined with IBM Cluster Systems Management V1.4 or later
- ▶ Using a virtual console for each partition or controlled system

With this feature, you can access every partition over the trusted network HMC connection to the server. This is a convenient feature when the partition is not reachable across the public network.

- ▶ The HMC provides a Service Focal Point for the systems it controls. It is connected to the service processor of the system using network connection and must be connected to each partition using an Ethernet LAN for Service Focal Point and to coordinate dynamic logical partitioning operations.
- ▶ The HMC provides tools for problem determination and service support, such as call-home and error log notification through an analog phone line or Ethernet.

Note: A partitioning-capable pSeries server that is managed by HMC is also referred to as a managed system.

1.5 IBM @server Information Center

The IBM @server Information Center provides a source of technical information about IBM @server p5 hardware. It offers technical documentation about how to configure and optimize @server p5 and OpenPower servers.

With AIX 5L Version 5.3, the role of the IBM @server and AIX Information Center has been expanded to provide a standardized and central repository for all relevant AIX and pSeries (prior to @server p5) manuals and documentation. The two components which make up the AIX InfoCenter structure are the existing InfoCenter Web portal and the AIX documentation CD. Both of these sources contain the following:

- ▶ A message database that shows the meaning of error messages and, in many cases, how to recover from the error. This database also provides information for LED codes and error identifiers.
- ▶ How-to tips with step-by-step instructions for completing system administrator and user tasks.
- ▶ FAQs for quick answers to common questions.
- ▶ The entire AIX software documentation library for Version 5.1, Version 5.2, and Version 5.3. Each publication is available in PDF format, and abstracts are provided for books for Version 5.2 and Version 5.3.
- ▶ Links to related documentation from IBM, including white papers, IBM Redbooks, and technical reports on topics such as RS/6000, SP, and HACMP™ for AIX. Release Notes and readme files are also available.

You can install the AIX Information Center application from the AIX Documentation CD. You can install and use it on a local system, or you can install it on a documentation server for intranet use.

The IBM @server hardware Information Center adds several new videos for customer-installable features and customer-replaceable parts.

For the latest AIX and pSeries (other than @server p5 hardware) information, refer to the AIX and pSeries Information Center at:

http://publib.boulder.ibm.com/pseries/en_US/infocenter/base

For the latest IBM @server p5 hardware information, see:

<http://publib.boulder.ibm.com/eserver/>

1.6 LPAR Validation Tool

Another new functional role which has been incorporated into the Information Center application is the ability to download new and useful tools. One example is the LPAR Validation Tool (LVT) on the InfoCenter Web site:

<http://www.ibm.com/servers/eserver/series/lpar/systemdesign.htm>

The LVT assists you in the design of an LPAR system and provides an LPAR validation report that reflects your system requirements while not exceeding LPAR recommendations.

Note: The LVT is not a marketing configurator, nor is it intended to replace one.

Figure 1-3 on page 16 shows one way to locate the LVT from inside the InfoCenter. You can find a detailed discussion about the LVT in 2.3, “Resource planning using LPAR Validation Tool” on page 42.

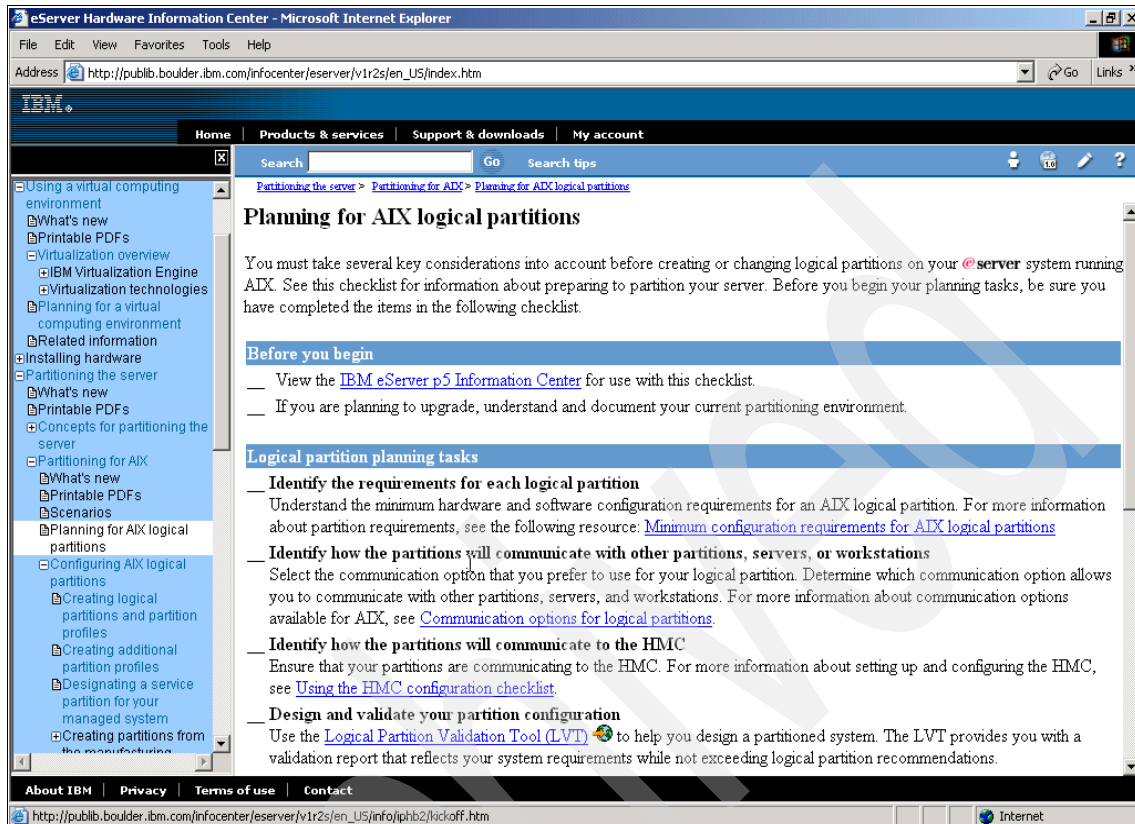


Figure 1-3 InfoCenter - Planning for AIX logical partitions

1.7 Operating system support

With the introduction of the POWER5 processor and the POWER hypervisor, the ability of the supported operating systems to exploit the advancements in hardware and firmware is critical in leveraging all of the benefits of the @server p5 and OpenPower server. This section describes what new and existing features are supported on each of the available operating systems.

AIX and Linux are supported operating systems that you can install on selected @server p5 models. A version of Linux is available for OpenPower systems. These operating systems operate as independent logical servers. However, partitions share some system attributes, such as the system serial number, system model, and processor feature code. All other system attributes can vary among partitions.

A mixed environment between AIX 5L Version 5.2 and Version 5.3 and Linux partitions on @server p5 servers is supported. Figure 1-4 shows a sample configuration with mixed operating systems including different AIX versions. The first five partitions use dedicated processors. The AIX 5L Version 5.2 partition is not able to join the virtual I/O paths, but it provides all the known LPAR and DLPAR features. It has to be configured with dedicated I/O adapters. The AIX 5L Version 5.3 partitions using shared processors likewise can use dedicated storage and dedicated networking.

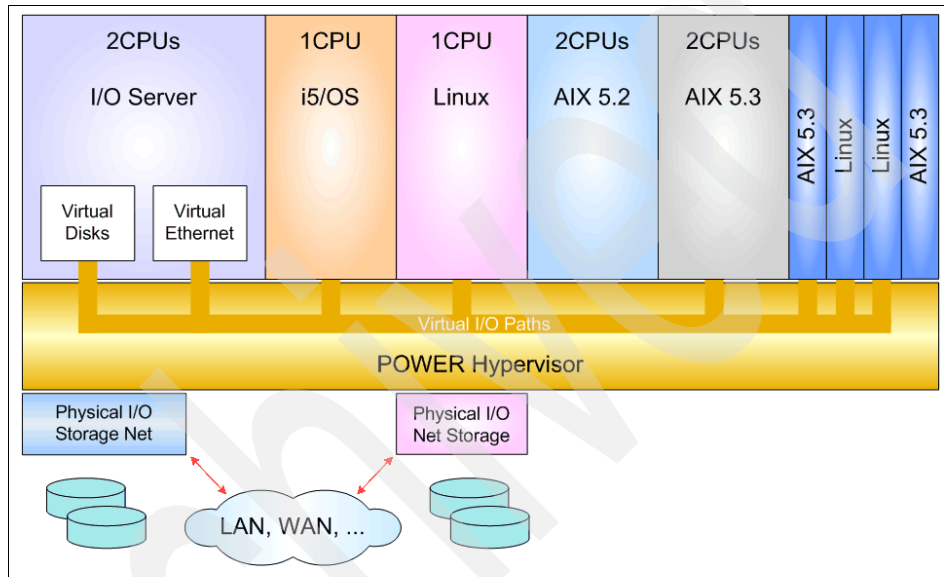


Figure 1-4 Mixed operating systems

Note: Partitions using virtualized shared resources cannot perform DLPAR operations with partitions that are assigned dedicated hardware resources.

Note: The Linux and i5/OS™ partitions are able to participate in selected virtual I/O operations on the @server p5 servers that are mixed operating environments.

Table 1-4 illustrates the available operating system supported functions on @server p5 servers for partitions in a mixed operating system environment.

Table 1-4 Operating systems supported functions.

Function	AIX 5.2	AIX 5.3	Linux SLES 9	Linux RHEL3 U3
Max 254 partitions	N	Y	Y	Y
Micro-Partitioning	N	Y	Y	Y
Capped and uncapped partitions	N	Y	Y	Y
Capacity on Demand				
-processors	Y	Y	Y	Static
-memory	Y	Y	Static	Static
DLPAR				
-processors	Y	Y	Y	N
-memory	Y	Y	N	N
-I/O	Y	Y	Y	N
POWER5 support				
-base	Y	Y	Y	Y
-SMT	N	Y	Y	Y
Virtual SCSI server	N	Y	N	N
Virtual SCSI client	N	Y	Y	Y
Virtual LAN	N	Y	Y	Y
EEH Recovery	Y	Y	N	N
Large page support	Y	Y	Y	N
Concurrent diagnostics	Y	Y	N	N
PCI Hot Plug	Y	Y	Y	N
I/O drawer/tower concurrent add/remove	N	Y	Y	Y
Memory resilience	N	Y	N	N
Machine check handling	Y	Y	Y	Y

1.7.1 AIX

To coincide with the POWER5 processor and new @server p5 product range, IBM released the AIX 5L Version 5.3 operating system to leverage the new server's 64-bit system and software architecture. This release supports new IBM @server p5 hardware systems, advanced POWER virtualization and symmetric multi-threaded POWER5 processors for improved system performance and utilization. AIX 5L Version 5.3 offers scalability for up to 64-way systems.

In addition to the introduction of the Advanced POWER Virtualization features, AIX 5L Version 5.3 also includes new system management tools, security enhancements, support for NFS V4, POSIX Real-time, and the consolidation of both the online and CD-ROM based AIX documentation into the InfoCenter application.

Table 1-5 provides an overview of which advance POWER virtualization features are available on both @server p5 supported AIX operating system platforms.

Table 1-5 AIX - supported features

Feature	AIX 5L v5.2 ML4	AIX 5L v5.3
POWER4 support	Yes	Yes
POWER5 support	Yes	Yes
Dynamic LPAR -CPU	Yes	Yes
Dynamic LPAR - Memory	Yes	Yes
Dynamic LPAR -I/O	Yes	Yes
Micro-Partitioning technology	No	Yes
Virtual Ethernet	No	Yes
Virtual SCSI Client	No	Yes
>140 Partitions	No	Yes

Note: Advanced POWER Virtualization is a separate additional feature that you need to order with a @server p5 server.

AIX 5L Version 5.2

In addition to all the enhancements and components provided in the former releases, AIX 5L Version 5.2 provides many enhancements. This section introduces briefly some of the enhancements that this book covers.

Note: AIX 5L Version 5.2 ML4 or later is required when using @server p5 servers.

Note: The 32-bit AIX kernel supports up to 96 GB of physical memory size and 32-way processors, regardless of whether it is in a partition or it is running as a full system partition.

Fast reboot in a partitioned environment

Rebooting an operating system instance in a partition is much faster than a full system reboot of a comparable conventional @server p5 system because less hardware initialization is required.

Partition reboots are merely a re-establishment of the pSeries Open Firmware operating system boot loader environment and, by nature, are very quick. A Full System Partition reboot repeats all the hardware initialization phases of the processors, caches, and memory. These phases are done by the service processor, and the I/O drawers and I/O adapters are done by the system firmware. When configuring all system resources in a single partition, hypervisor remains resident in memory. This configuration enables the extremely rapid re-establishment of the boot environment but requires the reservation of the first physical memory block by the hypervisor.

Dynamic logical partitioning

Starting with AIX 5L Version 5.2, AIX supports DLPAR, which allows you to add and remove resources dynamically without requiring a partition reboot. Both 32- and 64-bit kernels running in a partition support the DLPAR function. DLPAR provides the following features:

- ▶ Dynamic allocation and de-allocation of processors, memory, and PCI I/O adapters.
- ▶ Capacity on Demand (CoD).

For a detailed explanation of DLPAR, see 5.1, “Dynamic logical partitioning overview” on page 140.

Although POWER5 processor based pSeries servers support AIX 5L Version 5.2, it is not possible to run an AIX 5L Version 5.2 partition with Micro-Partitioning, Virtual SCSI, Virtual Ethernet, or SEAs.

AIX 5L Version 5.3

This section introduces the AIX 5L Version 5.3 enhancements that this book discusses.

Micro-Partitioning

The system administrator defines the number of virtual processors that a partition can use as well as the actual physical processor capacity. The system administrator can define a partition with a processor capacity as small as 10 processor units. These 10 units represent 1/10 of a physical processor, and each processor can be shared by up to 10 shared processor partitions.

Shared processor partitions need dedicated memory. However, the partition I/O requirements can be supported through Virtual Ethernet and Virtual SCSI. Using all virtualization features, 254 shared processor partitions are supported.

Virtual Ethernet

The development of Virtual Ethernet enables inter-partition communications without the need of a dedicated physical network adapter. The system administrator can define in-memory point-to-point connections between partitions. These connections have the same characteristics as a high-bandwidth Ethernet network and support multiple networking protocols, such as IPv4, IPv6, and ICMP.

Note: The maximum number of virtualized Ethernet adapters supported on a @server p5 AIX 5L Version 5.3 logical partition is 256.

Shared Ethernet Adapter

The Shared Ethernet Adapter (SEA) is a bridge from a physical Ethernet adapter to one or more Virtual Ethernet adapters. The adapter provides a secure route for partition network traffic to an external Ethernet network. With @server p5 servers, you must order this service with the Virtual I/O Server partition feature.

Virtual SCSI

The Virtual SCSI feature was developed to maximize the available hardware resources of the @server p5 servers. Because some @server p5 servers can support up to 254 partitions within the one server, the demand placed on the PCI slots and storage devices, either internal or attached, has increased significantly.

To meet these needs, the Virtual SCSI is designed to provide logical and physical volumes. From the Virtual I/O Server, these logical volumes appear to be SCSI disks on the client partition, giving the system administrator maximum flexibility in configuring partitions. At the time of the writing of this book, Virtual SCSI

supports Fibre Channel, parallel SCSI, and SCSI RAID devices using SCSI protocol.

Virtual SCSI includes two additional services which also run on the Virtual I/O Server: Reliable Command / Response Transport and Logical Remote DMA to service I/O requests for an I/O client. With these services, the I/O client uses the services of its own SCSI adapter.

Note: Because there are numerous limitations and considerations when implementing the new features, these features are covered in more detail in Chapter 4, “Virtualized resource management” on page 83.

1.7.2 Linux

With the release of the POWER5 processor based pSeries servers, the support for logical partitions running the Linux operating system has continued with the inclusion of the latest @server p5 and OpenPower server and virtualization features. Both the SUSE Linux Enterprise Server 9 (SLES9) and Red Hat Enterprise Linux 3 (RHEL3) distributions have been designed to take advantage of the new POWER5 processor and virtualization features, such as Micro-Partitioning, SMT, Virtual LAN, and Virtual SCSI client. The OpenPower product line supports only Linux as an operating system.

Support for various virtualization features is dependent on the Linux Distribution and Kernel version. At the time of the writing of this book, features such as Micro-Partitioning, Simultaneous Multi-threading (available on SUSE Linux Enterprise Server 9 only), Virtual Ethernet, and Virtual SCSI client are supported.

Also, some of the other @server p5 and OpenPower server features that are supported by SLES9 and RHEL3 are first failure data capture, double data rate IBM Chipkill™ memory, error checking and correcting memory, Dynamic Processor De-allocation, and hot-plug PCI slots, fans, and power.

For the latest updates to the SUSE or Red Hat Linux offering for POWER5 processor-based IBM @servers, refer to:

<http://www.redhat.com/software>

<http://www.suse.com>

<http://www.ibm.com/servers/eserver/linux/power/index.html>

<http://www.ibm.com/linux/whitepapers/>

Note: One function not supported by Linux on @server p5 and OpenPower servers is Dynamic Memory allocation or de-allocation.

Another key pSeries server component that continues to mature on the Linux operating systems is the increasing adoption of proven autonomic computing technologies from IBM.

A full list of reliability, availability, and scalability features that are supported by SLES9 and RHEL3 is available at:

http://www.ibm.com/servers/eserver/linux/power/whitepapers/linux_overview.pdf

Not all devices and features supported by the AIX operating system are supported in logical partitions that are running the Linux operating system. You can find information about external devices and features supported on @server p5 server products at:

<http://www.ibm.com/servers/eserver/pseries/hardware/factsfeatures.html>

<http://www-1.ibm.com/servers/eserver/pseries/linux>

http://www.ibm.com/servers/eserver/pseries/linux/whitepapers/linux_pseries.html

Partitioning implementation

This chapter explains the partitioning implementation on @server p5 and OpenPower servers and includes the following sections:

- ▶ 2.1, “Partitioning enablers” on page 26
- ▶ 2.2, “Partition resources” on page 32
- ▶ 2.3, “Resource planning using LPAR Validation Tool” on page 42
- ▶ 2.4, “I/O device assignment considerations” on page 46
- ▶ 2.5, “LPAR limitations and considerations” on page 53

2.1 Partitioning enablers

With the introduction of the @server p5 and OpenPower servers, the flexibility and adaptability of the partitioned environment has increased significantly. Each logical partition can now support both dedicated and virtualized server resources, while maintaining the strict isolation of the assigned resources from other partitions in the server. To achieve this flexibility, the hardware and firmware system components work together to support the available partitioning implementations on @server p5 and OpenPower servers.

Note: In the remainder of this book, when we refer to available partitioning implementations, we include both dedicated and virtualized partitions in the server environment.

2.1.1 Hardware

This section discusses the hardware components that support the available partitioning environments on partitioning-capable pSeries servers.

POWER5 processor

As the next generation of dual-core 64-bit processors, the POWER5 continues to incorporate IBM proven fabrication technologies such as silicon-on insulator and copper interconnection. The newer design also includes increased processor performance and an enhanced computing approach through improved granularity by way of the Advanced POWER Virtualization technology features. In addition to these characteristics, POWER5 provides the following functions to support all of the available partitioned environments:

► **POWER Hypervisor**

The POWER5 processor supports a special form of instructions. These instructions are exclusively used by new controlling firmware called the POWER Hypervisor. The hypervisor performs the initialization and configuration of the processors on partition activation. It provides privileged and protected access to assigned partition hardware resources and enables the use of Advanced POWER Virtualization features. The hypervisor receives and responds to these requests using specialized hypervisor calls. The hypervisor functions are discussed in “Firmware” on page 29.

Note: This book uses the term hypervisor to refer to the POWER Hypervisor.

► **Simultaneous multi-threading (SMT)**

SMT allows for better utilization and greater performance of a POWER5 processor by executing two separate threads simultaneously on an SMT-enabled processor. These threads are switched if one of the threads experiences a stalled or long latency event, such as memory miss or disk data fetch. The SMT feature works with both dedicated and shared processor resources and provides an estimated performance increase of 30%.

► **Dynamic Power Management**

To address one of the most important issues facing modern Complementary Metal Oxide Semiconductor chip designs, the POWER5 processor has newly embedded logic which reduces the amount of power that the processor consumes. The techniques developed to achieve this reduction in power usage were reducing switch power during idle clock cycles and a low-power mode of operation, which causes both processor threads to be set to the lowest possible priority.

Note: SMT is active on the POWER5 processor when combined with AIX 5L Version 5.3 or SUSE Linux.

Capacity on Demand

To enable customers to meet temporary, permanent, or unexpected computing workloads, certain @server p5 servers provide a number of enhanced Capacity on Demand methods, namely for processors. These include the following:

► **Capacity Upgrade on Demand**

With this feature, additional processors and memory are shipped with the server and can be later activated in minimum increments of one.

► **On/Off Capacity on Demand**

If ordered, this feature allows you to temporarily activate a minimum of one processor to meet peak system loads. With this feature, you must report your on and off activity to IBM at least once a month.

► **Reserve Capacity on Demand**

This feature further enhances the micro-partition flexibility of the @server p5 servers, by allowing you to order a number of inactive processors, which are placed in the shared processor pool as reserved processors. When the assigned or available processor resources that are used in uncapped partitions reaches 100%, additional processing days (normally a 24 hours period) are subtracted from a pre-paid number of processor days.

► Trial Capacity on Demand

This option provides you with a one-time, no-cost processor activation for a maximum period of 30 consecutive days. It is available as a complementary service when access to Capacity on Demand resources is required immediately.

There are basic rules which apply for each specific @server p5 server that supports Capacity on Demand. For more information, refer to the technical overview documentation for the individual server as shown in “Related publications” on page 301.

Note: At the time of the writing of this book, only p5-550 and p5-570 systems with a 2-way 1.65 GHz POWER5 and above processor card support the Capacity Upgrade on Demand options. All p5-590 and p5-595 also support these features.

Interrupt controller

The interrupt controller that manages the peripheral interrupt requests to the processors works in a fashion similar to other pSeries SMP servers. In a partitioned environment, the interrupt controller supports multiple global interrupt queues, which can be individually programmed to send external interrupts only to the set of processors that are allocated to a specific partition. Therefore, the processors in a partition can only receive interrupt requests from devices inside their partition.

PCI host bridges

The PCI host bridges control the PCI slots in the I/O drawers, as in conventional @server p5 servers. The PCI host bridges use translation control entry (TCE) tables for the I/O address to memory address translation in order to perform direct memory access (DMA) transfers between memory and PCI adapters. The TCE tables are allocated in the physical memory.

In a partitioned environment, the hypervisor controls the DMA addressing to the partition memory for all I/O devices in all partitions. The hypervisor uses central TCE tables for all I/O devices, which are located outside of the memory of the partitions. The hypervisor can manage as many TCE tables as it needs. For example, each PCI host bridge could have its own TCE table. The number of TCEs needed, and thus the number of TCE tables, is a function of the number of PCI host bridges and slots. The address mapping is protected on a per-adaptor basis. The PCI host bridges that are used in @server p5 and OpenPower servers support the control of the PCI adapter DMA by the hypervisor.

The key point is that a logical partition is only given a window of TCEs per I/O slot that are necessary to establish DMA mappings for the device in that slot. The hypervisor controls TCE accesses by the operating system to ensure that they are to a window owned by the partition.

Error handling

The basis of the Enhanced Error Handling function is to isolate and limit the impact of hardware errors or failures to a single partition. To further enhance the Reliability, Availability, and Serviceability capabilities, the POWER5 processor has improved the following features:

- ▶ Most firmware updates enable the system to remain operational.
- ▶ Error checking and correcting has been extended to inter-chip connections for the Fabric and Processor bus.
- ▶ Partial L2 cache de-allocation is possible.
- ▶ The number of L3 cache line deletes improved from two to ten for better self-healing capability.

Service processor

All the @server p5 and OpenPower server models have an enhanced service processor compared to existing pSeries models. The two major enhancements of the @server p5 and OpenPower SP are:

- ▶ The Hardware Management Console (HMC) communicates with the physical server using an Ethernet network
- ▶ The power control of the attached I/O subsystems occurs from the HMC using system power control network connection(s).

2.1.2 Firmware

Support of partitioning on @server p5 and OpenPower servers requires the new firmware-based hypervisor, partition Open Firmware, and Run-Time Abstraction Service. See Figure 2-2 on page 32.

POWER Hypervisor

The hypervisor is a component of the global firmware. It owns the partitioning model and the resources that are required to support this model. The hypervisor enables the use of Virtual I/O Server and other Advanced POWER5 Virtualization features.

When the Advanced POWER Virtualization hardware feature is specified with the initial system order, the firmware is shipped pre-activated to support Micro-Partitioning technology and the Virtual I/O Server. For upgrade orders, a

key is shipped to enable the firmware, similar to the Capacity Upgrade on Demand key.

Figure 2-1 shows the HMC panel where the Virtualization Engine™ Technologies are enabled.

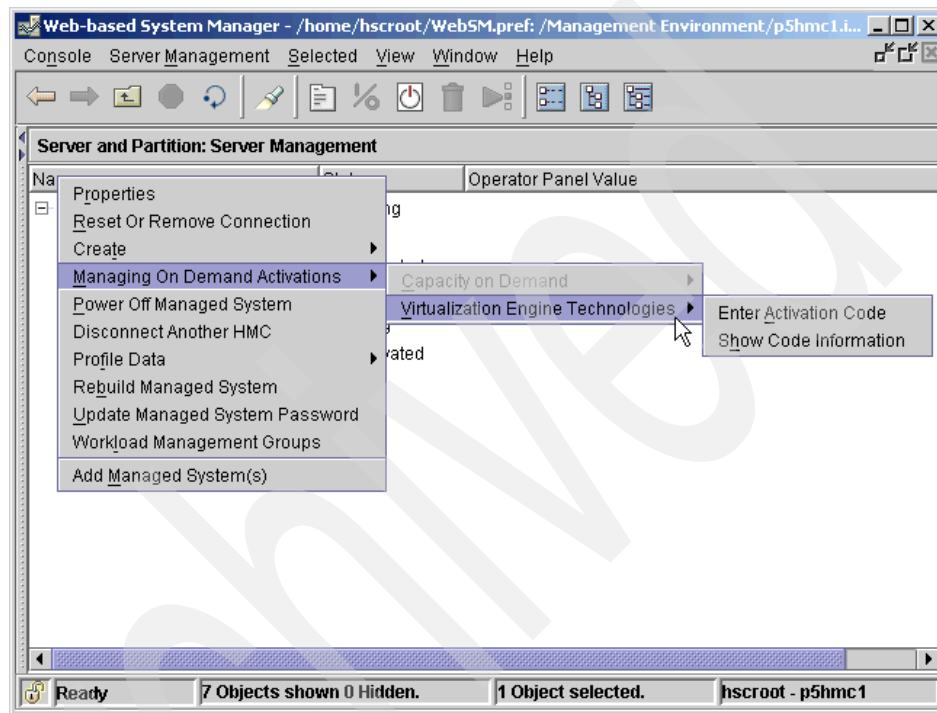


Figure 2-1 HMC panel to enable the Virtualization Engine Technologies

In addition to the Page Frame Table and Translation Control Entry, the hypervisor also handles the following system service calls:

- ▶ Virtual memory management
- ▶ Debugger support
- ▶ Virtual terminal support
- ▶ Processor register hypervisor resource access
- ▶ Dump support
- ▶ Debugger support
- ▶ Memory migration support
- ▶ Performance monitor support
- ▶ Virtualization I/O interrupts
- ▶ Micro-Partitioning scheduling
- ▶ Dynamic LPAR operations

The hypervisor is a callable, active, interrupt-driven service in @server p5 and OpenPower systems. This differs from POWER4-based systems where the hypervisor was a passive callable library.

The hypervisor resides outside of the partition system memory in the first physical memory block at physical address zero. This first physical memory block is not usable by any of the partition operating systems in a partitioned environment.

Note: For information about the hypervisor, see Chapter 6, “The POWER Hypervisor” on page 217.

Open Firmware

A @server p5 and OpenPower server has one instance of Open Firmware both when in the partitioned environment and when running as a full system partition. Open Firmware has access to all devices and data in the system. Open Firmware is started when the system goes through a power-on reset. Open Firmware, which runs in addition to the hypervisor in a partitioned environment, runs in two modes: global and partition. Each mode of Open Firmware shares the same firmware binary that is stored in the flash memory.

In a partitioned environment, Open Firmware runs on top of the global Open Firmware instance. The partition Open Firmware is started when a partition is activated. Each partition has its own instance of Open Firmware and has access to all the devices assigned to that partition. However, each instance of Open Firmware has no access to devices outside of the partition in which it runs. Partition firmware resides within the partition memory and is replaced when AIX takes control. Partition firmware is needed only for the time that is necessary to load AIX into the partition system memory. The global firmware resides with the hypervisor firmware in the first 256 MB of the physical memory.

The global Open Firmware environment includes the partition manager component. That component is an application in the global Open Firmware that establishes partitions and their corresponding resources (such as CPU, memory, and I/O slots), which are defined in partition profiles. The partition manager manages the operational partitioning transactions. It responds to commands from the service processor external command interface that originate in the application that is running on the HMC.

To confirm the current firmware level, you can use the `lscfg` command as follows:

```
lscfg -vp | grep -p 'Platform Firmware:'  
Platform Firmware:  
ROM Level.(alterable).....RH020930  
Version.....RS6K  
System Info Specific.(YL)...U1.18-P1-H2/Y1  
Physical Location: U1.18-P1-H2/Y1
```

This example shows firmware level RH020930.

Note: The firmware level shown in this example might be a different level from that shown on your system.

Figure 2-2 shows the POWER hypervisor environment using Open Firmware.

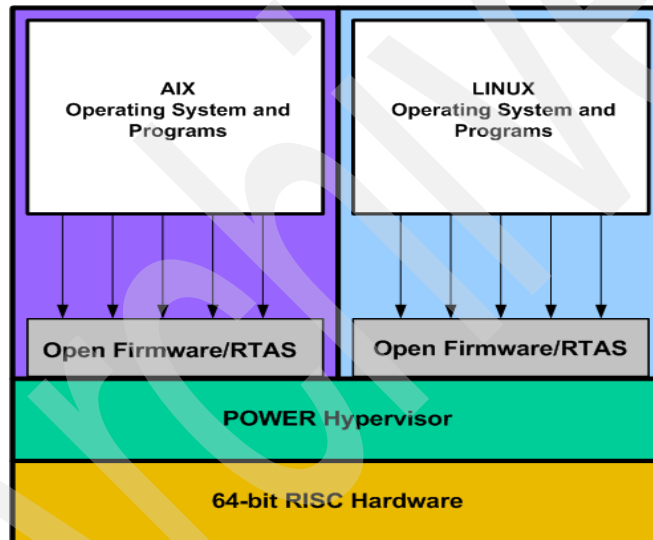


Figure 2-2 POWER hypervisor

2.2 Partition resources

Logical partitioning allows you to assign dedicated processors or, when you use the Micro-Partitioning feature of @server p5 and OpenPower systems, to assign processing units to partitions. You can define a partition with a processor capacity as small as 0.10 processing units, which represents 10 percent of a

physical processor. You can also assign physical memory and physical I/O devices or virtual I/O-devices (SCSI or Ethernet) to partitions.

The following sections give an overview of resource assignments.

2.2.1 Partition and system profiles

The information about resources that are assigned to a partition is stored in a partition *profile*. Each partition can have multiple partition profiles. By switching from one partition profile to another, you can change how resources are assigned. For example, you can assign relatively small resources to small online transactions on weekdays and assign large resources to high-volume batch transactions on weekends.

To change partition profiles, you must shut down the operating system instance that is running in the partition and stop (deactivate) the partition. You can also define a system profile (for administrative purposes) as an optional task. By using a system profile, you can turn on multiple partitions in a specific order in one operation.

There are two types of profiles: partition and system.

- ▶ **Partition profile**

A partition profile stores the information about the assigned resources for a specific partition, such as processor, memory, physical I/O devices, and virtual I/O devices (Ethernet, serial, and SCSI). Each partition must have a unique name and at least one partition profile. A partition can have several partition profiles, but it reads only one partition profile when it is started (activated). You select a partition profile when you activate the partition. Otherwise, the default partition profile is used. You can designate any partition profile as the default partition profile. If there is only one partition profile for a partition, it is always the default.

- ▶ **System profile**

A system profile provides a collection of partition profiles that should be started at the same time. The partition profiles are activated in the order of the list that is defined in the system profile.

Both types of profiles are stored in the non-volatile random access memory (NVRAM) of the server. Although you can create many partition profiles and system partition profiles, the actual number possible depends on your profile configuration, because both types of profiles share the same memory area in the NVRAM.

Figure 2-3 on page 34 summarizes the relationship among partitions, partition profiles, and system profiles. In this figure, partition A has three partition profiles,

B has one, and C has two. The default partition profile for each partition is represented with a check mark. The system profile X is associated with partition profiles A1, B1, and C2, and the system profile Y is associated with partition profiles A1 and C1. Keep in mind the following points:

- ▶ You do not have to associate all the partition profiles with system profiles. In this example, the partition profiles A2 and A3 are not associated with any system profile.
- ▶ It is possible to associate a partition profile to multiple system profiles. In this example, the partition profile A1 is associated with system profile X and Y.

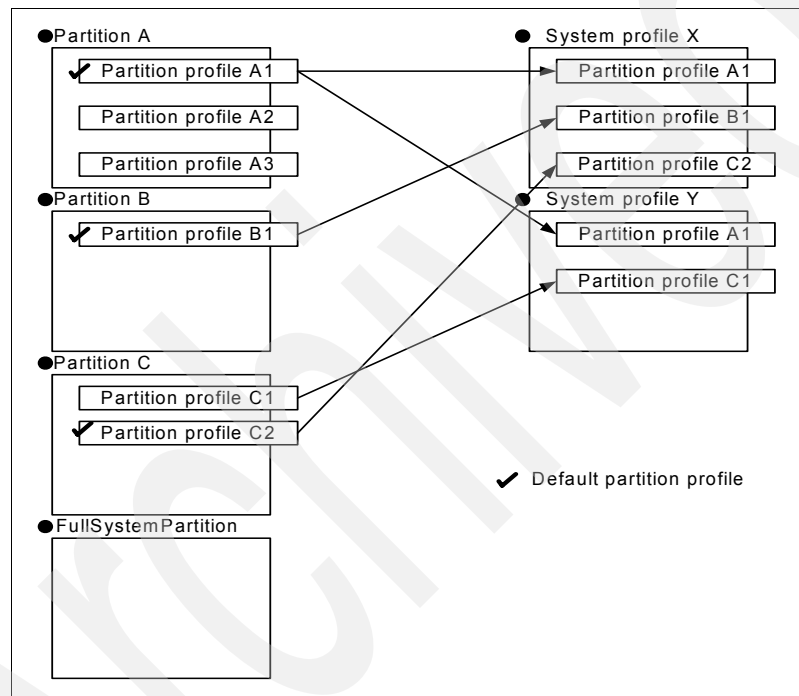


Figure 2-3 Partitions, partition profiles, and system profiles

To create partition profiles and system profiles, use the IBM Hardware Management Console for @server p5 and OpenPower systems.

2.2.2 Processors

Continuing the evolution of the partitioning technology on pSeries servers, the POWER5 processor improves its flexibility in using partitions. There are two types of partitions in @server p5 and OpenPower servers. Partitions can have processors dedicated to them, or they can have their processors virtualized from

a pool of shared physical processors. This is known as Micro-Partitioning technology. With this technology, both types of partitions can coexist in the same system at the same time.

The concept of Micro-Partitioning technology allows the resource definition of a partition to allocate fractions of processors to the partition. Physical processors have been virtualized to enable these resources to be shared by multiple partitions. There are several advantages associated with this technology, including higher resource utilization and more partitions that can exist concurrently. Micro-Partitioning technology is implemented on *PowerPC* p5 servers with AIX 5L Version 5.3 or Linux operating system environments.

A dedicated processor partition, such as the partitions that are used on POWER4 processor based servers, have an entire processor that is assigned to a partition. These processors are owned by the partition where they are running and are not shared with other partitions. Also, the amount of processing capacity on the partition is limited by the total processing capacity of the number of processors configured in that partition, and it cannot go over this capacity (unless you add or move more processors from another partition to the partition that is using a dynamic LPAR operation).

By default, a powered-off logical partition using dedicated processors will have its processors available to the shared processing pool. When the processors are in the shared processing pool, an uncapped partition that needs more processing power can use the idle processing resources. However, when you turn on the dedicated partition while the uncapped partition is using the processors, the activated partition regains all of its processing resources. If you want to prevent dedicated processors from being used in the shared processing pool, you can disable this function using the logical partition profile properties panels on the Hardware Management Console (see Figure 2-4 on page 36).

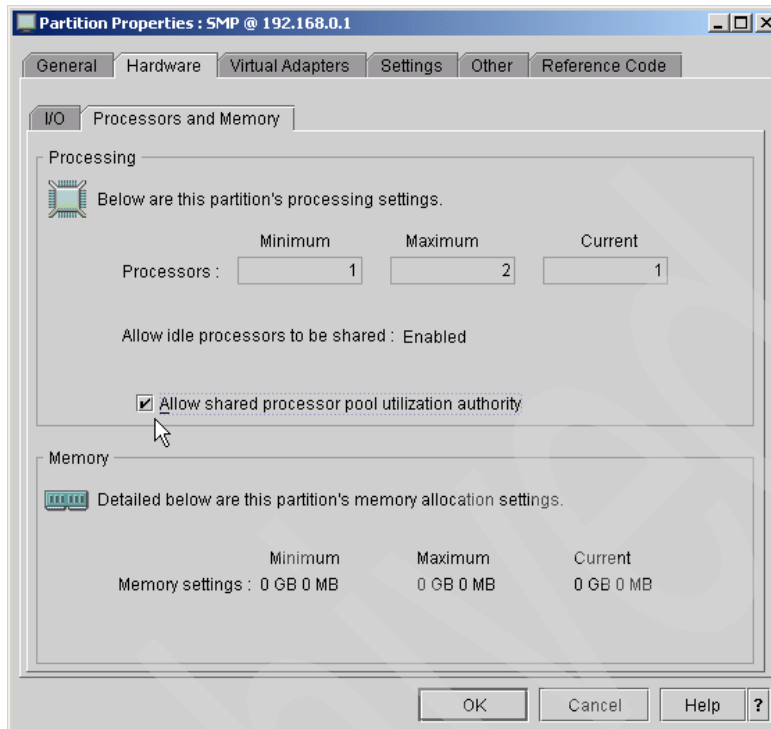


Figure 2-4 Allow and disallow shared processor partition utilization authority

Micro-Partitioning technology differs from dedicated processor partitions in that physical processors are abstracted into *virtual processors* which are then assigned to partitions. These virtual processors have capacities ranging from 10 percent of a physical processor up to the entire processor. Therefore, a system can have multiple partitions that share the same physical processor and that divide the processing capacity among themselves.

2.2.3 Memory

When discussing memory, it is important to highlight that the new @server p5 and OpenPower servers and their associated virtualization features have adopted an even more dynamic memory allocation policy than the previous partition capable pSeries servers. Also, despite its increased flexibility, the underlying fundamentals and mechanisms within a virtual or dedicated logical environment has remained relatively static.

Because the word memory is overused in various contexts, we have provided definitions of the following four terms regarding memory: virtual, physical, real, and logical memory.

The term *virtual memory* is used in many operating system environments to express the function that enables the operating system to act as though it were equipped with a larger memory size than it physically has.

Because each process should be isolated from the other processes, each has its own virtual memory address range, called the process address space. Each process address space is classified into several memory regions named segments. Each segment is again divided into small size memory regions, named pages.

Because not all of the virtual memory can sit in the *physical memory* in the system, only some portions of virtual memory are mapped to physical memory. The rest of the virtual memory is divided by page size. Each page can be mapped to a disk block in paging spaces or can reside in a block of files in the file systems. This address translation is managed by the virtual memory manager (VMM) of the operating system using hardware components, such as the hardware page frame table and translation look-aside buffer.

The term *real memory* is often used to represent the physical memory, especially when discussing the VMM functionality in the kernel. The modifier *real* comes from the real addressing mode defined in some processor architectures (including PowerPC®), where address translation is turned off. In a non-partitioned environment, because there is a one-to-one relationship between the real and physical memory, the difference between these two terms can be ignored in most cases.

The physical address space must encompass all addressable hardware components, such as memory cards, I/O ports, bus memory, and so on. Depending on the hardware implementation and restrictions, address ranges might need to be dispersed throughout the physical address space, which could result in a discontinuous physical memory address space. For example, if a PCI adapter device requires DMA, the device's DMA address is mapped on the specific physical memory address range by a PCI host bridge. Most VMMs of modern operating systems are designed to handle non-contiguous physical memory addresses. However, operating systems require a certain amount of contiguous physical memory that can be addressed as translate-off, typically for bootstrapping, in a non-partitioned environment.

In a partitioned environment, real and physical memories must be distinguished. The physical memory address, which previously meant the *real memory* address, is no longer used in that way because there is an extra level of addressing in a partitioned environment.

To support any operating system, including AIX and Linux, which requires real mode code execution and the ability to present a real address space starting at zero to each partition in the system, the *logical memory* concept is adopted. Logical memory is an abstract representation that provides a contiguous memory address to a partition. Multiple non-contiguous physical memory blocks are mapped to provide a contiguous logical memory address space. The logical address space provides the isolation and security of the partition operating system from direct access to physical memory, allowing the hypervisor to police valid logical address ranges assigned to the partition. The contiguous nature of the logical address space is used more for simplifying the hypervisor's per-partition policing than it is used because it is an operating system requirement. The operating system's VMM handles the logical memory as though it were physical memory in a non-partitioned environment.

Important: For @server p5 and OpenPower systems, the size of the minimum logical memory block has been reduced from 256 MB to 16 MB to facilitate smaller partitions.

In a partitioned environment, some of the physical memory areas are reserved by several system functions to enable partitioning in the partitioning-capable pSeries server. You can assign unused physical memory to a partition. You do not have to specify the precise address of the assigned physical memory in the partition profile, because the system selects the resources automatically.

From the hardware perspective the minimum amount of physical memory for each partition is 128 MB, but in most cases AIX needs 256 MB of memory. After that, you can assign further physical memory to partitions in increments of 16 MB.

The AIX VMM manages the logical memory within a partition as it does the real memory in a stand-alone pSeries server. The hypervisor and the POWER5 processor manage access to the physical memory.

Memory requirements for partitions depend on partition configuration, I/O resources assigned, and applications used. Memory can be assigned in increments of 16 MB, 32 MB, 64 MB, 128 MB, and 256 MB. The default memory block size varies according to the amount of configurable memory in the system, as shown in Table 2-1 on page 39.

Table 2-1 Default memory block sizes

Amount of configurable memory	Default memory block size
Less than 4 GB	16 MB
Greater than 4 GB, up to 8 GB	32 MB
Greater than 8 GB, up to 16 GB	64 MB
Greater than 16 GB, up to 32 GB	128 MB
Greater than 32 GB	256 MB

The default memory block size can be changed by using the Logical Memory Block Size option in the Advanced System Management Interface. To change the default memory block size, you must be a user with administrator authority, and you must shut down and restart the managed system for the change to take effect. If the minimum memory amount in any partition profile on the managed system is less than the new default memory block size, you must also change the minimum memory amount in the partition profile.

Depending on the overall memory in your system and the maximum memory values that you choose for each partition, the server firmware must have enough memory to perform logical partition tasks. Each partition has a Hardware Page Table (HPT). The size of the HPT is based on an HPT ratio of 1/64 and is determined by the maximum memory values that you establish for each partition.

Server firmware requires memory to support the logical partitions on the server. The amount of memory that is required by the server firmware varies according to several factors. Factors influencing server firmware memory requirements include:

- ▶ Number of logical partitions
- ▶ Partition environments of the logical partitions
- ▶ Number of physical and virtual I/O devices used by the logical partitions
- ▶ Maximum memory values given to the logical partitions

Generally, you can estimate the amount of memory that is required by server firmware to be approximately eight percent of the system installed memory. The actual amount that is required will generally be less than eight percent. However, there are some server models that require an absolute minimum amount of memory for server firmware, regardless of the previously mentioned considerations.

When selecting the maximum memory values for each partition, consider the following:

- ▶ That maximum values affect the HPT size for each partition
- ▶ The logical memory map size for each partition
- ▶ Using the LPAR Validation Tool to provide the actual memory that is used by firmware

Note: The Hardware Page Table is created based on the maximum values that are defined on partition profile.

2.2.4 Physical I/O slots

Physical I/O devices are assignable to partitions on a PCI slot (physical PCI connector) basis. It is not the PCI adapters in the PCI slots that are assigned as partition resources, but the PCI slots into which the PCI adapters are plugged.

When using physical I/O devices to install an operating system, you have to assign at least one, typically an SCSI adapter that is able to boot the operating system, and an adapter to access the install media. Instead of physical I/O devices, you can assign a virtual I/O device that behaves like a physical I/O device.

Once installed, you need at least one physical device adapter that is connected to the boot disk or disks. For application use and system management purposes, you also have to assign at least one physical network adapter. You can allocate physical slots in any I/O drawer on the system.

If you must add physical PCI slots into a running partition, you have two possibilities:

1. You can run an DLPAR operation from HMC to add or to move an empty PCI slot to the partition. After successful addition, you will use AIX PCI Hot Plug Manager to add a PCI Hot Plug Adapter.
2. You can assign more PCI slots than required for the number of adapters in the partition, even if these PCI slots are not populated with PCI adapters. This provides you with the flexibility to add PCI adapters into the empty slots of an active partition, using the PCI Hot Plug insertion and removal capability.

2.2.5 Virtual I/O

Virtual I/O allows the @server p5 and OpenPower servers to support more partitions than it has slots for I/O devices by enabling the sharing of I/O adapters between partitions.

Virtual Ethernet enables a partition to communicate with other partitions without the need for an Ethernet adapter. A shared Ethernet adapter, supported by the Virtual I/O Server, allows a shared path to an external network.

Virtual SCSI enables a partition to access block-level storage that is not a physical resource of that partition. With the Virtual SCSI design, the virtual storage is backed by a logical volume on a portion of a disk or an entire physical disk. These logical volumes appear to be the SCSI disks on the client partition, which gives the system administrator maximum flexibility in configuring partitions.

Virtual SCSI support is provided by a service running in an I/O server that uses two primitive functions:

- ▶ Reliable Command / Response Transport
- ▶ Logical Remote DMA to service I/O requests for an I/O client, such that, the I/O client appears to enjoy the services of its own SCSI adapter

The term *I/O server* refers to platform partitions that are servers of requests, and *I/O client* refers to platform partitions that are clients of requests, usually I/O operations, that use the I/O server's I/O adapters. This allows a platform to have more I/O clients than it may have I/O adapters, because the I/O clients share I/O adapters using the I/O server.

2.2.6 Minimum, desired, and maximum values

In a partition profile, you need to specify three kinds of values for each resource.

For memory, you must specify *minimum*, *desired*, and *maximum* values.

For processor, you define whether you use dedicated or shared processors. If you chose to use dedicated processor, you can specify *minimum*, *desired*, and *maximum* values. For shared processors, you need to specify *minimum*, *desired*, and *maximum* values for both processing units and virtual processors.

Note: If you define the maximum value of virtual processors as one, you cannot obtain processing units over 1.00 to that partition without changing the maximum value of virtual processors in that partition's profile on HMC.

For physical and virtual I/O slots, you must specify the *required* and *desired* values.

Note: You cannot move or remove an I/O slot if it is defined as required on the partition profile. First, you must change its state from required to desired. After that, you must run the AIX `rmdev` command to remove the PCI slot from the running operating system.

If any of the three types of resources cannot satisfy the specified minimum and required values, the activation of a partition fails. If the available resources satisfy all the minimum and required values but do not satisfy the desired values, the activated partition will get as many of the resources that are available.

The maximum value is used to limit the maximum processor and memory resources when dynamic logical partitioning operations are performed on the partition.

Note: The maximum memory value also affects the size of the partition page table.

2.3 Resource planning using LPAR Validation Tool

The LPAR Validation Tool (LVT) is a tool that helps you to validate the resources that are assigned to LPARs. The LVT was designed specifically for the latest @server p5 servers. As partitioned environments grow increasingly more complex, the LVT tool should be your first resource to determine how a system can be effectively partitioned.

Note: At the time of the writing of this book, the LVT did not include the OpenPower models.

LVT is a Java™-based tool that is loaded on a Microsoft® Windows® 95 or above workstation with at least 128 MB of free memory. Its footprint on disk, at the time of the writing of this book, is about 47 MB. It includes an IBM Java Runtime Environment 1.4. The installation adds an icon to the desktop.

For information, including user's guide and download information, see:

<http://www.ibm.com/servers/eserver/iseriess/lpar/systemdesign.htm>

During the development of this book, the LVT was receiving regular updates. Installation only required a couple of minutes, with the most time devoted to downloading the code. An update, available as a separate file, brings the base code to the latest level and is significantly smaller in size.

2.3.1 System Selection dialog

After the tools are launched, you can create a new configuration or load an existing one, as shown in Figure 2-5.

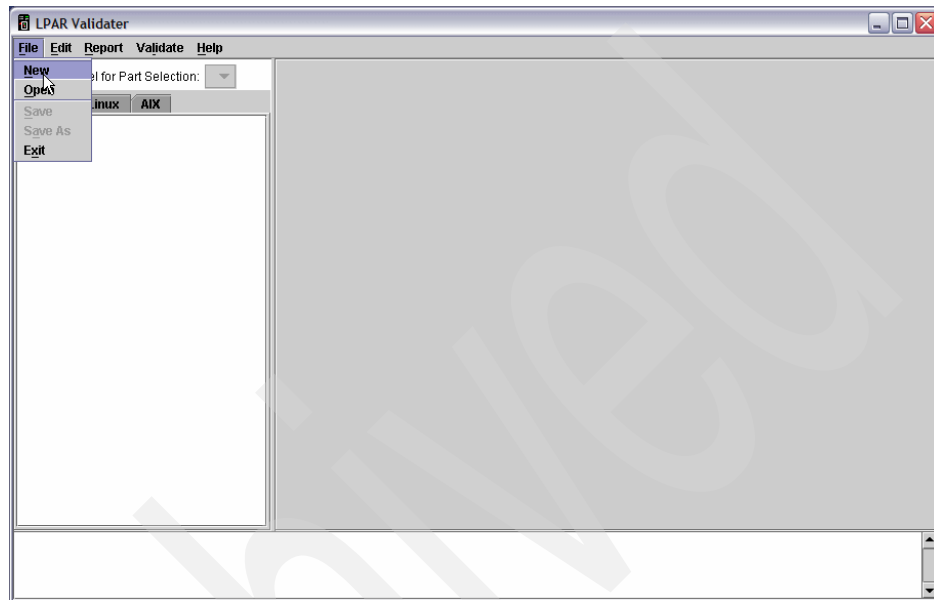


Figure 2-5 LPAR Validation Tool, creating a new partition

When you open a new configuration, you select the basic attributes of the machine that you plan to validate, as shown in Figure 2-6 on page 44.

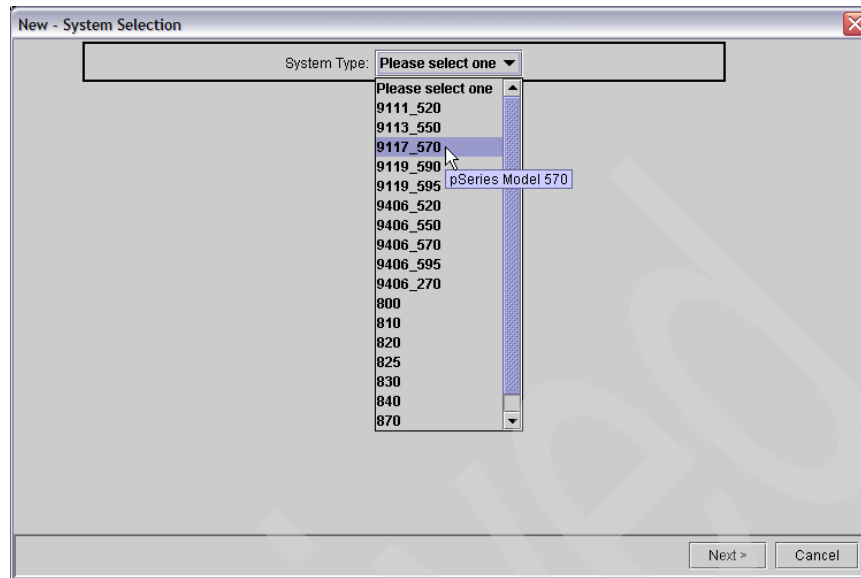


Figure 2-6 LPAR Validation Tool, System Selection dialog

Hold your cursor over a field, and additional information is provided, as shown in Figure 2-7.

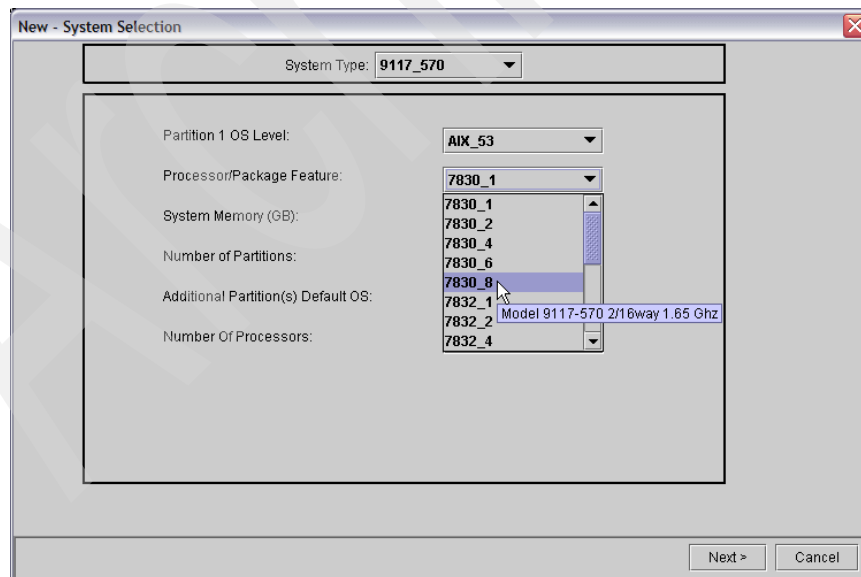


Figure 2-7 LPAR Validation Tool, System Selection processor feature selection

2.3.2 Memory Specification dialog

After you complete the System Selection fields, you enter the memory specifications for each of the logical partitions that you previously specified (Figure 2-8).

Memory Specifications

System Model: 9117_570
Processor/Package Feature: 7830_8
System Memory (GB): 16.0
Total Processors: 8

System Memory(MB): 16384.0
Configured Memory: 16384
Hypervisor Memory: 55
Unallocated Memory: 94

Partition	OS Version	Memory	Max Memory	Virtual Slots	Virtual Ethernet	Virtual Serial	Server SCSI	Client SCSI
P1	AIX_53	4096	4096	2	0	2	0	0
P2	AIX_53	4096	4096	2	0	2	0	0
P3	IO_Virtual_Ser...	8192	8192	2	3	2	0	0

The Virtual Ethernet enables inter-partition communication without the need for physical network adapters in each partition. Virtual Ethernet allows the administrator to define in-memory point to point connections between partitions. Virtual Ethernet requires a p5 system with either IBM AIX 5L Version 5.3 or the appropriate level of Linux and a Hardware Management Console (HMC) to define the Virtual Ethernet devices. Virtual Ethernet does not require the purchase of any additional features or software such as the Advanced POWER Virtualization Feature.

OS/400 License(s) Required: 0.0
AIX License(s) Required: 8.0
Linux License(s) Required: 0.0

< Back Finish Cancel

Figure 2-8 LPAR Validation Tool, Memory Specifications dialog

As you enter the memory specifications, the unallocated memory and the amount that is required by the hypervisor are shown. These values increase as the number of virtual devices defined increase. Internal validation prevents configurations that do not have enough resource.

This tool answers a common question in planning, “What is the memory usage of the POWER Hypervisor.”

2.3.3 LPAR Validation dialog

The final dialog enables you to assign features to the various slots defined, as shown in Figure 2-9.

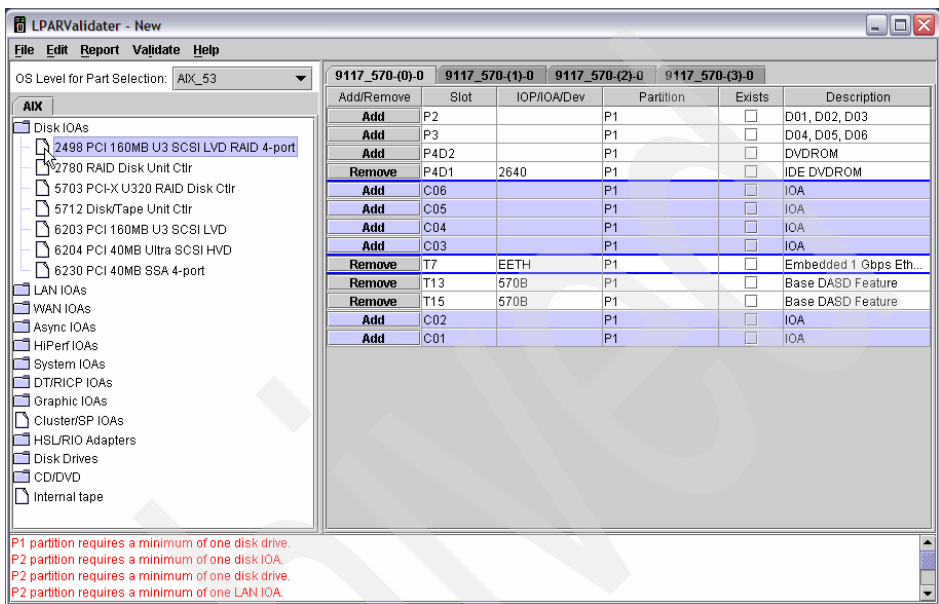


Figure 2-9 LPAR Validation Tool, slot assignments

From this screen, you can view a detailed report and select a validation engine to point out any errors in the configuration. If changes to the memory configuration are required, you can edit the configuration and change the values in error. When the configuration has been validated without error, you can feel confident that the resources that you selected will provide the configuration desired. At this point, if you choose, you can configure the system in an ordering tool or through a miscellaneous equipment specification upgrade of an existing machine, with the additional resources that are required to handle a new LPAR configuration.

2.4 I/O device assignment considerations

With the introduction of the Advanced POWER Virtualization feature, the number of available I/O options and scenarios has increased considerably. To ensure that the maximum benefits from all available partition environments are gained from the implementation or upgrade, you need a greater understanding of the hardware I/O architectural platform.

The hardware limitations of each @server p5 and OpenPower server model also influence aspects of the implementation or upgrade planning.

2.4.1 Media devices

If your installation media is removable media (for example CD-ROM, DVD-RAM, or 4 mm tape), the corresponding devices should be configured. However, the configuration of removable media devices depends on the hardware architecture of @server p5 and OpenPower servers as described in this section.

p5-520 and p5-550, OpenPower 720

These servers, including rack-mounted and desktide models, support three non-hot-swappable media bays which are used to accommodate additional devices.

Two media bays only accept slim line media devices, such as IDE DVD-ROM (FC 2640) or DVD-RAM (FC 5751) drives, and one half-height bay can be used for a tape drive. However, there are several device-reassignment operations that are required on the HMC in order to use these devices as the installation media device on these models, if you use the servers in a dedicated partition environment. You can also configure the following SCSI-attached tape devices on these models:

- ▶ (FC 6120): 8 mm 80/160 GB tape drive
- ▶ (FC 6134): 8 mm 60/150 GB tape drive
- ▶ (FC 6258): 4 mm 36/72 GB tape drive

p5-570 (Rack-mounted model)

As the hardware design of this server is based on modular building blocks, each Central Electronics Complex component supports two media bays which accept the optional slim-line media devices, such as IDE DVD-ROM (feature code (FC 2640) or DVD-RAM (FC 5751) drives.

Note: The maximum number of media devices in a fully configured p5-570 is eight.

p5-590 and p5-595

These servers can be configured with an optional storage device enclosure (FC 7212-102) which is can only be mounted in an 19-inch rack. This enclosure

contains two media bays, which can support any combination of the following IDE or SCSI devices:

- ▶ (FC 1103): DVD-RAM drive
- ▶ (FC 1104): 8 mm VXA-2 tape drive
- ▶ (FC 1105): 4 mm DAT72 DDS-4 tape drive kit
- ▶ (FC 1106): DVD-ROM drive

Note: One of the media bays must contain an optical drive.

Each of these servers can be configured with a USB diskette drive (FC 2591), which can be supported on either an integrated or external USB adapter. A USB adapter (FC 2738) is required for an external USB port connection.

Note: At the time of the writing of this book, media devices cannot be virtualized.

If your installation media is in the network, one of the following network adapters must be assigned to the partition:

- ▶ Ethernet
- ▶ Token ring

2.4.2 Boot device considerations

The following sections describe the boot device considerations for partitions.

Full system partition and dedicated partitions

When implementing either of these partitioning models, each partition requires its own separate boot device. Therefore, you must assign at least one boot device and a corresponding adapter per partition. The @server p5 and OpenPower servers support boot devices connected with SCSI, SSA, and Fibre Channel adapters. Boot over network is also available as an operating system installation option.

p5-520

Both rack-mounted and desktide models support up to eight internal SCSI disk drives which are housed in two 4-pack disk bays. In the base configuration, each 4-pack is connected to one of the two ports on the integrated SCSI controller. To an LPAR, the entire SCSI controller (including all disks attached to both ports) will be seen as P1-T10, and therefore can only be assigned to one active LPAR at a time. To provide additional drives for a second LPAR, either virtual I/O or an optional PCI SCSI adapter feature should be used. The internal disk drive bays

can be used in two different modes, depending on whether the SCSI RAID Enablement Card (FC 5709) is installed.

The other partitions must be assigned to the boot adapter and disk drive from the following options:

- ▶ A boot adapter inserted in one of six PCI-X slots in the system. A bootable external disk subsystem is connected to this adapter.
- ▶ A bootable SCSI adapter is inserted in the PCI-X slot 7, or two SCSI adapters, one in PCI-X slot 5 and PCI-X slot 7 in a 7311-D20 I/O drawer connected to the system. The adapter(s) is connected to one of a 6-pack of disk bays of the drawer that houses the boot disk drive.
- ▶ A boot adapter inserted in one of seven PCI-X slots in a 7311-D20 I/O drawer connected to the system. A bootable external disk subsystem is connected to this adapter.

Note: The p5-520 models support up to four 7311-D20 I/O drawers.

p5-550 and OpenPower 720

Both rack-mounted and desk-side models support up to eight internal SCSI disk drives, which are housed in two 4-pack disk bays. To an LPAR, the entire SCSI controller (including all disks attached to both ports) will be seen as P1-T10, and therefore can only be assigned to one active LPAR at a time. To provide additional drives for a second LPAR, either virtual I/O or an optional PCI SCSI adapter feature should be used. Assigned to this boot adapter, a boot disk drive must use one of the following options:

- ▶ A boot adapter inserted in one of five PCI-X slots in the system. The adapter could then be connected to the second internal SCSI 4-pack in the system.
- ▶ A boot adapter inserted in one of five PCI-X slots in the system. A bootable external disk subsystem is connected to this adapter.
- ▶ A bootable SCSI adapter (which can have various features) is inserted in the PCI-X slot 7, or two SCSI adapters, one in PCI-X slot 5 and PCI-X slot 7 in a 7311-D20 I/O drawer connected to the system. The adapter(s) is connected to one of a 6-pack of disk bays of drawer that houses the boot disk drive.
- ▶ A boot adapter inserted in one of seven PCI-X slots in a 7311-D20 I/O drawer connected to the system. A bootable external disk subsystem is connected to this adapter.

Note: The p5-550 and OpenPower 720 models support up to eight 7311-D20 I/O drawers.

p5-570

Each system drawer can contain up to six internal disks which are housed in one split 6-pack disk drive bay. These disks are connected to two Ultra320 internal SCSI controllers with dual ports, allowing each of the 3-packs to be assigned to a unique partition. Additional partitions must be assigned to the boot adapter and disk drive from the following options:

- ▶ A boot adapter inserted in one of five PCI-X slots in the system. A bootable external disk subsystem is connected to this adapter.
- ▶ 7311-D10, this drawer supports five hot-plug 64-bit 133 MHz 3.3 V PCI-X slots and one hot-plug 64-bit 33 MHz 5V PCI slot. All slots have the full length, blind-swap cassette.
- ▶ 7311-D11, a boot adapter inserted in one of six PCI-X slots in this drawer connected to the system. A bootable external disk subsystem can be connected to this adapter. This drawer supports six hot-plug 64-bit 133 MHz 3.3V PCI-X slots, full length, enhanced blind cassette.
- ▶ 7311-D20, a bootable SCSI adapter (which can have various features) is inserted in the PCI-X slot 7, or two SCSI adapters, one in PCI-X slot 5 and PCI-X slot 7 in a 7311-D20 I/O drawer connected to the system. The adapter(s) is connected to one of 6-pack disk bays of drawer that houses the boot disk drive.
- ▶ A boot adapter inserted in one of seven PCI-X slots in a 7311-D20 I/O drawer connected to the system. A bootable external disk subsystem is connected to this adapter.

Note: The p5-570 model can support up to a total combination of 20 7311-D10, 7311-D11, and 7311-D20 I/O drawers.

p5-590 and p5-595

Partitions must be assigned to the boot adapter and disk drive from the following options:

- ▶ An internal disk drive inserted in one of the 4-pack disk bays on I/O drawer and the SCSI controller on the drawer. The 7040-61D I/O drawer (FC 5791) can have up to 16 internal SCSI disk drives in the four 4-pack disk bays. Each of the disk bays is connected to a separate internal SCSI controller on the drawer.
- ▶ A boot adapter inserted in one of 20 PCI-X slots in a 7040-61D I/O drawer connected to the system. A bootable external disk subsystem is connected to this adapter.

You should select the adapter of the boot device from the PCI-X slot of the system or the first I/O drawer if the system is running as a full system partition.

The system locates the boot device faster. In a partitioned environment, the placement of the boot adapter does not affect the boot speed of partition.

The following points apply to the p5-590 and p5-595 models:

- ▶ The p5-590 supports up to eight 7040-61D I/O drawers, and the p5-595 supports up to 12. The minimum hardware configurations of these models require at least one I/O drawer.
- ▶ Existing 7040-61D I/O drawers may be attached to a p5-595 server as additional I/O drawers. Each 16-way processor book includes six Remote I/O-2 attachment cards for connection of the system I/O drawers.
- ▶ The parallel ports on these models are not supported in a partitioned environment.

Virtual I/O server

The Virtual I/O server complements the @server p5 server's Micro-Partitioning technology. The need to meet adequately the flexible I/O requirements of up to 254 logical partitions has driven the development of the Virtual SCSI and Virtual Ethernet. The following sections outlines what effect the Virtual I/O server has on the boot device capabilities of partitions in the shared resource pool.

Virtual SCSI disks

Virtual SCSI facilitates the sharing of physical disk resources (I/O adapters and devices) between the VIOS and the client partitions. Partitions must be assigned a SCSI adapter and disk drive(s) as follows:

- ▶ One or more client SCSI adapter(s) from the available candidates on the HMC.
- ▶ One or more logical volumes which appears as a real disk devices (hdisks).

For redundancy and high availability, consider mirroring AIX and Linux partition operating system disks across multiple virtual disks.

Note: Once a virtual disk is assigned to a client partition, the Virtual I/O server must be available before the client partitions are able to boot.

Virtual Ethernet

When installing or maintaining partitions with a Virtual Ethernet adapter, the AIX Network Installation Manager, Network Installation Manager on Linux, and Cluster Server Manager application operate in the same manner as they would with a dedicated Ethernet adapter assigned to the partition(s). Virtual Ethernet does not require the Virtual I/O server.

2.4.3 Network devices

It is mandatory to assign a network adapter to each partition. In addition to providing network access to client systems of a partition, the connection is also needed to provide the capability to manage the operating system and the applications in the partition remotely, either with a telnet session or a graphical user interface, such as the Web-based System Manager. An Ethernet network connection between partitions and the HMC must be available if you want to use the following services:

- ▶ Service Agent
- ▶ Service Focal Point
- ▶ Inventory Scout
- ▶ Dynamic logical partitioning
- ▶ Partition Load Manager

These services communicate over the TCP/IP network between the partitions and the HMC.

2.4.4 Graphics console

If you need direct console access to a partition without using the network, the partition must be assigned a graphics console. A graphics console is available on a partition by configuring the following features on the partition:

- ▶ A graphics adapter (FC 2849) with a graphics display
- ▶ A USB keyboard and mouse adapter (FC 2738) with a USB keyboard and a USB mouse attached

Only one graphics console is supported per partition. The graphics console is functional only when AIX is running. For any installation or service processor support functions, you have to use the virtual terminal function on the HMC.

2.4.5 High availability

You should place redundant devices of a partition in separate I/O drawers, where possible, for highest availability. For example, if two Fibre Channel adapters support multipath I/O to one logical unit number, and if one path fails, the device driver chooses another path using another adapter in another I/O drawer automatically.

Some PCI adapters do not have enhanced error handling capabilities built in to their device drivers. If these devices fail, the PCI host bridge in which they are placed and the other adapters in this PCI host bridge are affected. Therefore, it is strongly recommended that you place all adapters without enhanced error

handling capabilities on their own PCI host bridge and that you do not assign these adapters on the same PCI host bridge to different partitions.

2.5 LPAR limitations and considerations

Consider the following limitations when implementing shared processor partitions:

- ▶ The limitation for a shared processor partition is 0.1 processing units of a physical processor. So, the number of shared processor partitions you can create for a system depends mostly on the number of processors of a system.
- ▶ The system architecture is designed to support a maximum number of 254 partitions.
- ▶ In a partition, there is a maximum number of 64 virtual processors
- ▶ A mix of dedicated and shared processors within the same partition is not supported.
- ▶ If you dynamically remove a virtual processor you cannot specify a particular virtual CPU to be removed. The operating system will choose the virtual CPU to be removed.
- ▶ Shared processors can make AIX affinity management less effective. AIX continues to utilize affinity domain information as provided by firmware to build associations of virtual processors to memory and continues to show preference to re-dispatching a thread to the virtual CPU that it last ran on.

You should carefully consider the capacity requirements of online virtual processors before choosing values for their attributes. Virtual processors have dispatch latency, because they are scheduled. When a virtual processor is made runnable, it is placed on a run queue by the hypervisor, where it waits until it is dispatched. The time between these two events is referred to as *dispatch latency*.

The dispatch latency of a virtual processor depends on the partition entitlement and the number of virtual processors that are online in the partition. The capacity entitlement is equally divided amongst these online virtual processors, so the number of online virtual processors impacts the length of each virtual processor's dispatch. The smaller the dispatch cycle, the greater the dispatch latency.

At the time of the writing of this book, the worst case virtual processor dispatch latency is 18 milliseconds, because the minimum dispatch cycle that is supported at the virtual processor level is one millisecond. This latency is based on the minimum partition entitlement of 1/10 of a physical processor and the 10 millisecond rotation period of the hypervisor's dispatch wheel. It can be easily visualized by imagining that a virtual processor is scheduled in the first and last

portions of two 10 millisecond intervals. In general, if these latencies are too great, then clients may increase entitlement, minimize the number of online virtual processors without reducing entitlement, or use dedicated processor partitions.

In general, the value of the minimum, desired, and maximum virtual processor attributes should parallel those of the minimum, desired, and maximum capacity attributes in some fashion. A special allowance should be made for uncapped partitions, because they are allowed to consume more than their entitlement.

If the partition is uncapped, then the administrator may want to define the desired and maximum virtual processor attributes x percent above the corresponding entitlement attributes. The exact percentage is installation specific, but 25 to 50 percent is a reasonable number.

Table 2-2 lists several reasonable settings of number of virtual processor, processing units, and the capped and uncapped mode.

Table 2-2 Reasonable settings for shared processor partitions

Min VPs ^a	Desired VPs	Max VPs	Min PU ^b	Desired PU	Max. PU	Capped
1	2	4	0.1	2.0	4.0	Y
1	3 or 4	6 or 8	0.1	2.0	4.0	N
2	2	6	2.0	2.0	6.0	Y
2	3 or 4	8 or 10	2.0	2.0	6.0	N

a - Virtual processors

b - Processing units

Operating systems and applications that are running in shared partitions need not be aware that they are sharing processors. However, overall system performance can be significantly improved by minor operating system changes. AIX 5L Version 5.3 provides support for optimizing overall system performance of shared processor partitions.

In a shared partition, there is not a fixed relationship between the virtual processor and the physical processor. The hypervisor tries to use a physical processor with the same memory affinity as the virtual processor, but it is not guaranteed. Virtual processors have the concept of a home physical processor. If it cannot find a physical processor with the same memory affinity, then it gradually broadens its search to include processors with weaker memory affinity, until it finds one that it can use. As a consequence, memory affinity is expected to be weaker in shared processor partitions.

Workload variability is also expected to be increased in shared partitions, because there are latencies associated with the scheduling of virtual processors and interrupts. SMT may also increase variability, because it adds another level of resource sharing, which could lead to a situation where one thread interferes with the forward progress of its sibling.

Therefore, if an application is cache sensitive or cannot tolerate variability, then it should be deployed in a dedicated partition with SMT disabled. In dedicated partitions, the entire processor is assigned to a partition. Processors are not shared with other partitions, and they are not scheduled by the hypervisor. Dedicated partitions must be explicitly created by the system administrator using the HMC.

Processor and memory affinity data is only provided in dedicated partitions. In a shared processor partition, all processors are considered to have the same affinity. Affinity information is provided through RSET APIs, which contain discovery and bind services.

Basic partition management

This chapter describes the tools used to configure and control @server p5 systems and includes the following sections:

- ▶ 3.1, “Hardware Management Console” on page 58
- ▶ 3.2, “Advanced System Management Interface” on page 64
- ▶ 3.3, “Resetting a server” on page 69
- ▶ 3.4, “Partition Load Manager” on page 71

3.1 Hardware Management Console

In order to configure and administer a partitioning-capable pSeries server, you must attach at least one IBM Hardware Management Console (HMC) for pSeries to the system. You can find a general overview of HMC in 1.4, “IBM Hardware Management Console” on page 11.

One HMC is capable of controlling multiple pSeries servers. At the time of the writing of this book, a maximum of 32 non-clustered pSeries servers and a maximum of 254 partitions are supported by one HMC. You can add a second redundant HMC to the configuration.

You can add, remove, or move resources between partitions. When moving resources in partitions, you can use the Resource Monitoring and Controlling infrastructure to provide a secure and reliable connection channel between the HMC and the partitions. This connection channel is configured automatically by the HMC and by each AIX partition when the AIX partition is started. The HMC uses the open network LAN connection for this connection channel.

POWER5 processor-based system HMCs require Ethernet connectivity. Sufficient Ethernet adapters must be available to enable public and private networks if you need both.

The 7310 Model C04 is a desktop model with one native 10/100/1000 Ethernet port, two additional PCI slots for additional Ethernet adapters, two PCI-Express slots, and six USB ports.

The 7310 Model CR3 is a 1U, 19-inch rack-mountable drawer that has two native Ethernet ports, two additional PCI slots for additional Ethernet adapters, and three USB ports.

When an HMC is connected to @server p5 systems, the p5-570 integrated serial ports are disabled. If you need serial connections, for example for a non-Ethernet HACMP heartbeat, you must provide an async adapter.

Note: It is not possible to connect POWER4 and POWER5 processor-based systems to the same HMC simultaneously.

To extend this functionality, you can use:

- ▶ Another HMC for remote access. This remote HMC must have a network connection to the HMC, which is connected to the servers (see Figure 3-1 on page 59).

- An AIX 5L Web-based System Manager client to connect to the HMC over the network or the Web-based System Manager PC client, which runs on a Windows or Linux operating systems.

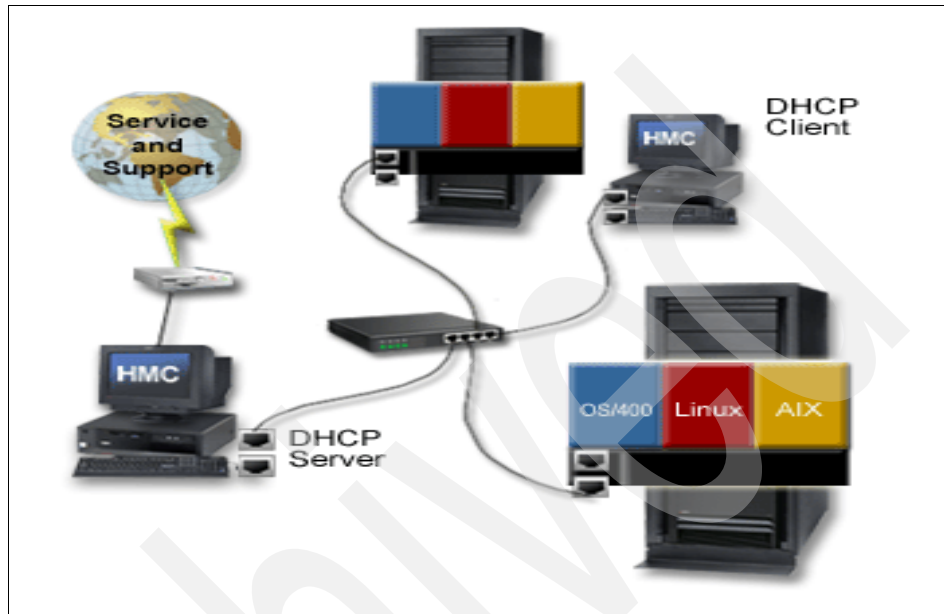


Figure 3-1 Dual HMC configuration

The resources that the HMC manipulates are I/O devices and slots, memory, and processors.

This book does not discuss how to create a basic partition and the full range of functions.

3.1.1 Managing I/O devices and slots

Logical partitions can have desired or required I/O devices or slots. When you specify that an I/O device or slot is desired (or shared), either the I/O device or slot is meant to be shared with other logical partitions or the I/O device or slot is optional. When you specify that an I/O device or slot is required (or dedicated), then you cannot activate the logical partition if the I/O device or slot is unavailable or in use by another logical partition.

Note: If resources are moved dynamically, the configuration change is temporary and is not reflected in the partition profile. Thus, all configuration changes are lost the next time that you activate the partition profile. If you want to save your new partition configuration, you must change the partition profile.

In Figure 3-2 and Figure 3-3 on page 61, the virtual adapters are examined.

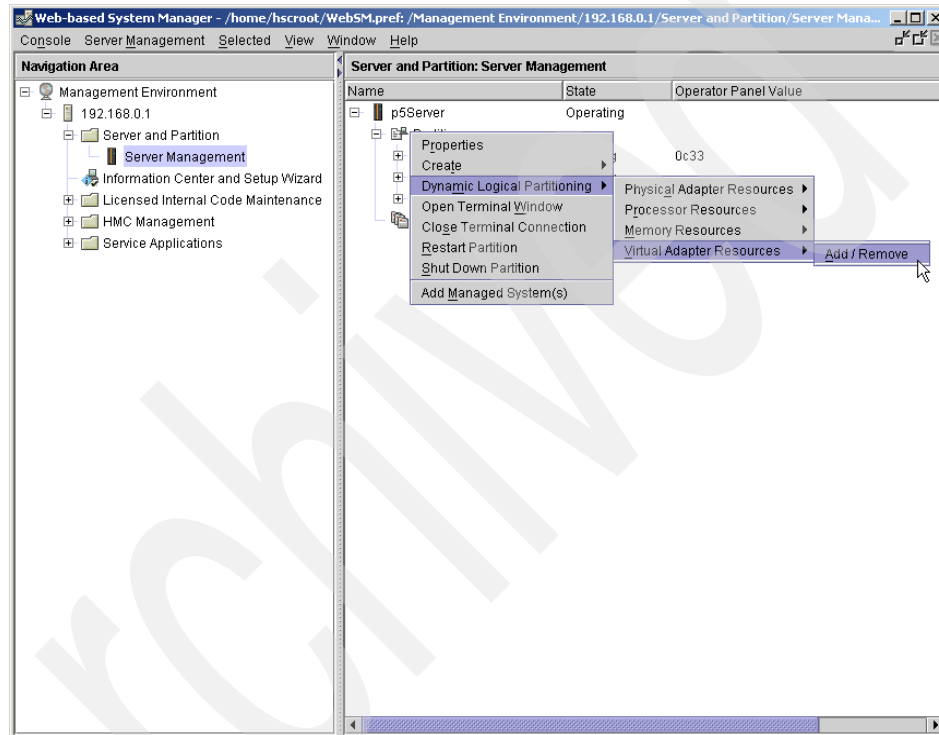


Figure 3-2 DLPAR Virtual Adapter menu

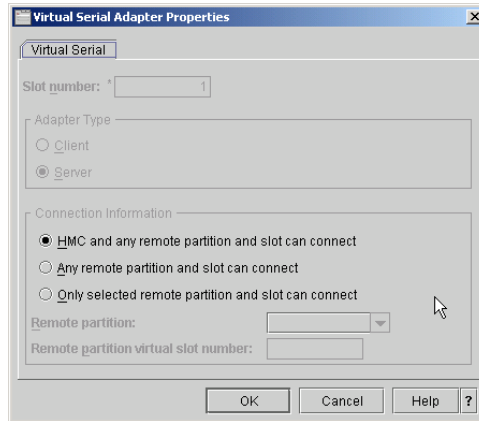


Figure 3-3 Virtual Adapter capabilities

3.1.2 Managing memory

Memory in each logical partition operates within its assigned minimum and maximum values. The full amount of memory that you assign to a logical partition might not be available for the partition's use. Static memory overhead that is required to support the assigned maximum memory affects the reserved or hidden memory amount. This static memory overhead also influences the minimum memory size of a partition (see Figure 3-4 and Figure 3-5 on page 62).

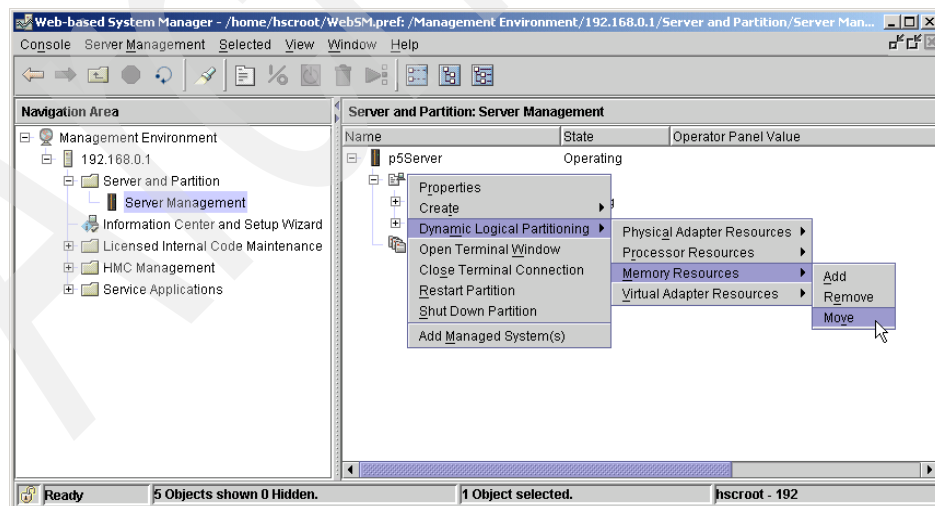


Figure 3-4 Move memory resources - step 1

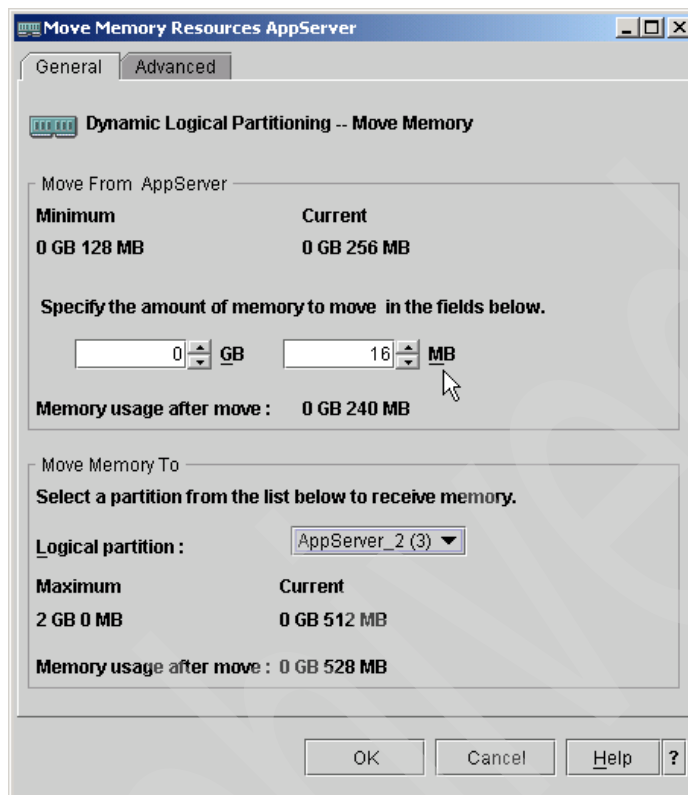


Figure 3-5 Move memory resources - step 2

3.1.3 Managing processing power

The ability to move processor capacity dynamically becomes important when you need to adjust to changing workloads. You can move processing capacity based on the desired, minimum, and maximum values that you created for the profile. The desired processing value that you establish is the amount of processing resources that you get if you do not overcommit the processing capacity. The minimum and maximum values enable you to establish a range within which you can move the processors dynamically.

For both shared and dedicated processors, you can specify a minimum value that is equal to the minimum amount of processing capacity that you need to support the logical partition. The maximum value must be less than the amount of processing capacity that is available on the system (see Figure 3-6 on page 63 and Figure 3-7 on page 63).

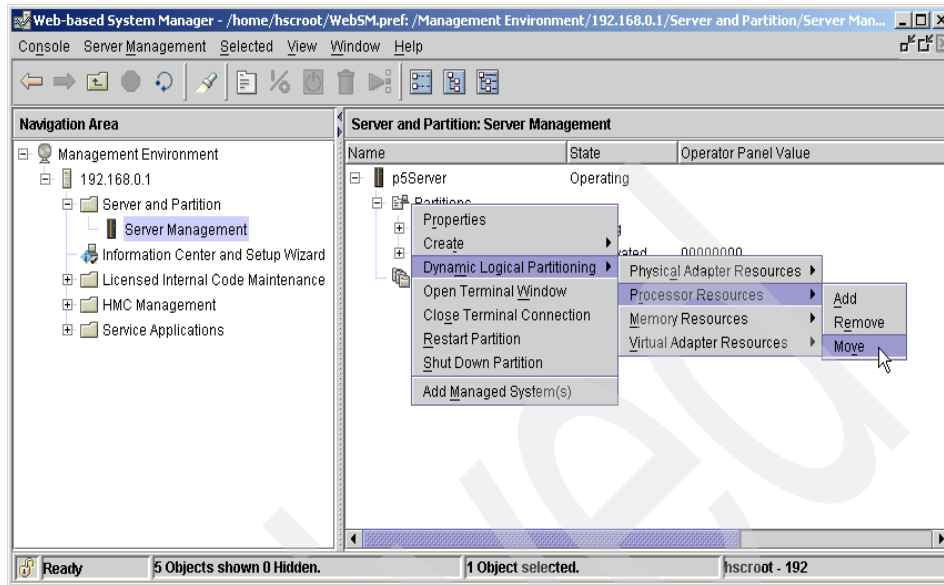


Figure 3-6 Move processor resources

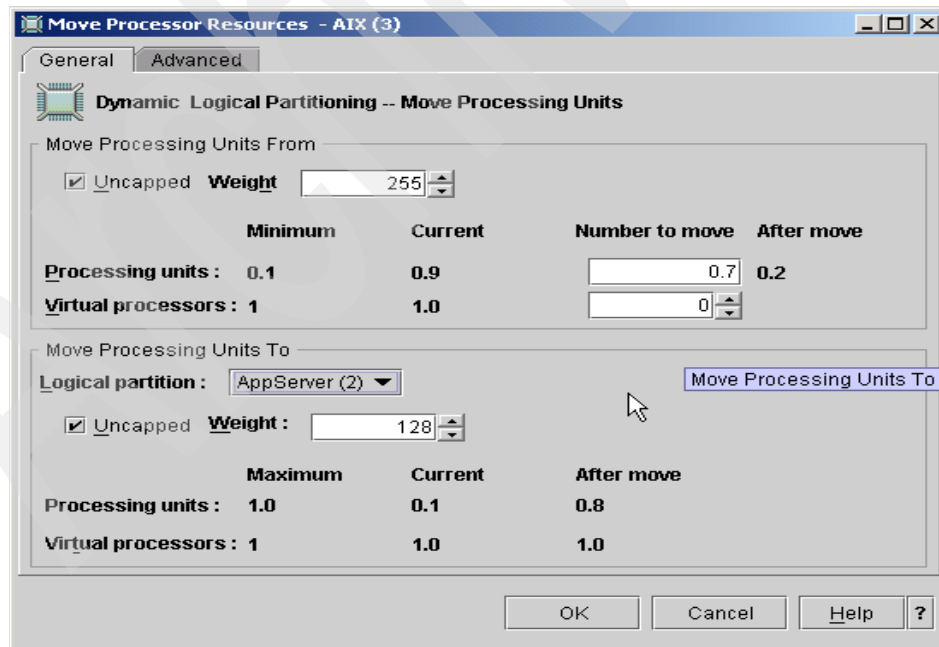


Figure 3-7 Move processing units

3.1.4 Scheduling movement of resources

You can schedule the dynamic movement of resources to and from logical partitions that are running on your managed system. When this procedure is completed, the managed system is set to perform the dynamic logical partitioning task at the date and time that you specify. You can set the managed system to add resources to a logical partition, remove resources from a logical partition, or move resources from one logical partition to another.

You can schedule the movement of memory and of dedicated processors. (You are not able to schedule the movement of shared processors.)

For more information about dynamic resources, see Chapter 5, “Dynamic logical partitioning” on page 139.

3.2 Advanced System Management Interface

The Advanced System Management Interface (ASMI) is the interface to the service processor that is required to perform general and administrator-level service tasks, such as reading service processor error logs, reading vital product data, setting up the service processor, and controlling the system power. You can access the ASMI through a Web browser, an ASCII console, or the HMC.

This interface is accessible using a Web browser on a client system that is connected to the service processor on an Ethernet network. You can also access it using a terminal that is attached to a serial port on the server.

This section covers two important system management topics in detail: network configuration and power/restart control.

With the system in power standby mode, or with an operating system in control of the machine or controlling the related partition, the service processor is working and checking the system for errors, ensuring the connection to the HMC for manageability purposes.

When the system status is standby, the service processor provides a System Management Interface (that you can access by pressing any key on an attached serial console keyboard, or the ASMI using a Web browser on a client system that is connected to the service processor on an Ethernet network.

The service processor and the ASMI are standard on all @server p5 processor-based hardware. Both system management interfaces require you to enter the general or admin ID password. They also allow you to set flags that affect the operation of the system according to the provided password (such as auto power restart), to view information about the system (such as the error log

and virtual product data) and the network environment access setup, and to control the system power.

3.2.1 Accessing the ASMI using a Web browser

The ASMI requires password authentication and provides a Web connection to the service processor over the Ethernet using the secure sockets layer (SSL). To establish an SSL connection, open your browser using `https://`.

Supported browsers are Netscape (version 7.1), Internet Explorer (version 6.0), and Opera (version 7.23). Later versions of these browsers are not supported. JavaScript and cookies must be enabled.

The browser-based ASMI is available during all phases of the system operation, including initial program load (IPL) and run time. Some menu options are blocked during the system IPL or run time to prevent usage or ownership conflicts if corresponding resources are in use during that phase.

If accessed on a terminal, ASMI is only available if the system is powered off.

All requested input must be provided in English-language characters regardless of the language selected to view the interface.

3.2.2 Accessing the ASMI using the HMC

To access the ASMI using the HMC, following these steps:

1. In the navigation area, expand the managed system with which you want to work.
2. Expand Service Applications and click **Service Focal Point**.
3. In the content area, click **Service Utilities**.
4. From the Service Utilities window, select the managed system with which you want to work.
5. From the Selected menu on the Service Utilities window, select **Launch ASM menu**.

3.2.3 Network configuration

You can initialize network addresses and attributes for the managed system with the ASMI using the following techniques:

- Dynamic configuration, using DHCP
- Manual configuration, if a specific address is required

Note: You can only use the ASMI to configure these network attributes when the system is powered off (see Figure 3-8 and Figure 3-9).

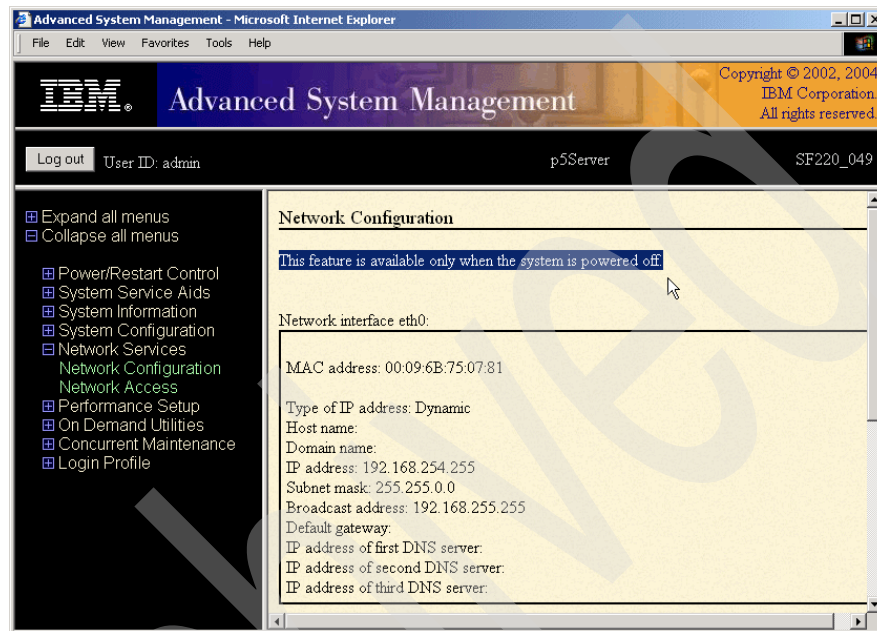


Figure 3-8 ASMI network configuration, powered on

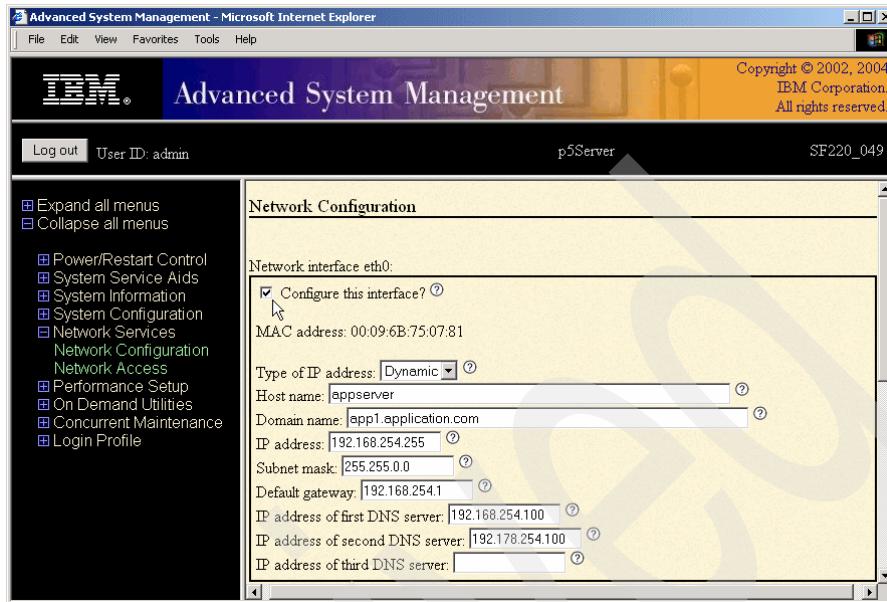


Figure 3-9 ASMI network configuration

3.2.4 Service processor

The service processor has a permanent firmware boot side, or A side, and a temporary firmware boot side, or B side. You should install new levels of firmware on the temporary side first in order to test the update's compatibility with your applications. When the new level of firmware has been approved, you can copy it to the permanent side.

With the system running, the service processor provides the ability to view and change the power-on settings using the ASMI. Also, the surveillance function of the service processor is monitoring the operating system to confirm that it is still running and has not stalled.

3.2.5 Power/Restart control

You can start and shut down the system in addition to setting IPL options. In ASMI, you can view and change the following IPL options:

- System boot speed
 - Fast or Slow. Fast boot results in skipped diagnostic tests and shorter memory tests during the boot.

- ▶ Firmware boot side for next boot
Permanent or Temporary. Firmware updates should be tested by booting from the temporary side before being copied into the permanent side.
- ▶ System operating mode
Manual or Normal. Manual mode overrides various automatic power-on functions, such as auto-power restart, and enables the power switch button.
- ▶ AIX/Linux partition mode boot (available only if the system is not managed by the HMC)
Service mode boot from saved list. This is the preferred way to run concurrent AIX diagnostics
Service mode boot from default list. This is the preferred way to run stand-alone AIX diagnostics
- ▶ Boot to open firmware prompt
Boot to System Management Service (SMS) to further select the boot devices or network boot options.
- ▶ Boot to server firmware
Select the state for the server firmware: Standby or Running. When the server is in the server firmware standby state, partitions can be set up and activated.

Refer to Figure 3-10 on page 69 for an example of the power control modes.

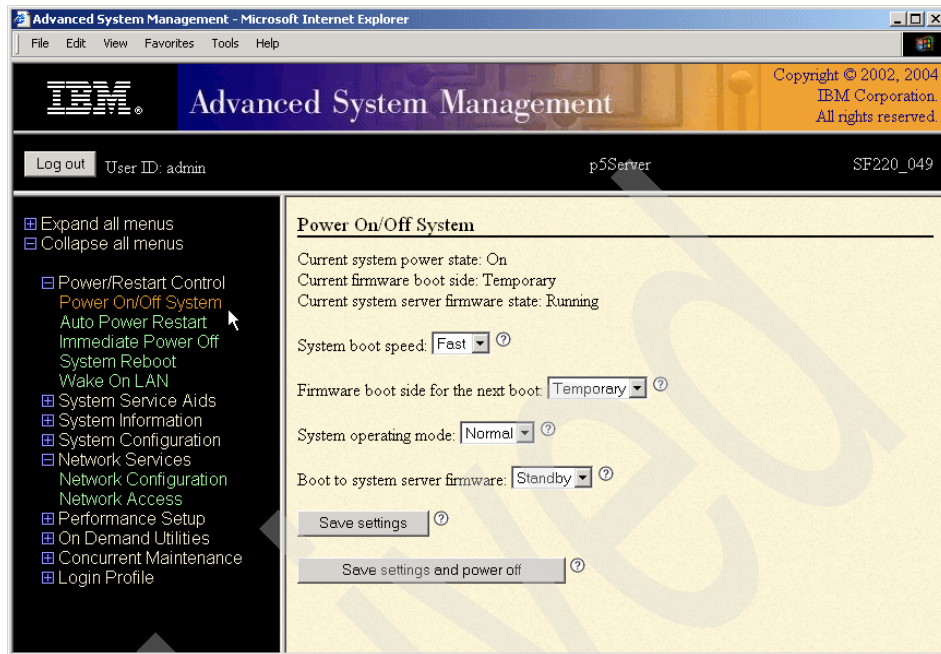


Figure 3-10 ASMI, Power On/Off System

3.3 Resetting a server

If you need support for an adapter that does not use extended error handling (EEH), and there is no possible solution to obtain EEH support for it or to provide an alternative hardware solution, there might be no other choice than to reset your server to the factory settings if the adapter is supported in this mode. This is also the case if you no longer wish to manage a system with an HMC.

Note: The p5-590 and p5-595 must be managed by an HMC.

This section describes how to reset a server in both of these cases.

3.3.1 EEH adapters and partitioning

Currently, you can order POWER5-based systems only with adapters that support EEH. Support of a non-EEH adapter (OEM adapter) is only possible when the system has not been configured for partitioning. This is the case when a new system is received, for example, and it is in full system partition which you

plan to use without an HMC. EEH is disabled for that adapter upon system initialization.

When the platform is prepared for partitioning or is partitioned, the hypervisor prevents disabling EEH upon system initialization. Firmware in the partition detects any non-EEH device drivers that are installed and that are not configured. Therefore, all adapters in @server p5 systems must be EEH capable in order to be used by a partition. This applies to I/O installed in I/O drawers attached to a @server p5 system and I/O installed in planar adapter slots found in @server p5 system units.

You do not need to actually create more than a single partition to put the platform in a state where the hypervisor considers it to be partitioned. The platform becomes partitioned (in general, but also in specific reference to EEH enabled by default) as soon as you attach an HMC and perform any function that relates to partitioning. Simple hardware service operations do not partition the platform, so it is not simply connecting an HMC that has this affect. However, modifying any platform attributes that are related to partitioning (such as booting under HMC control to only PHYP standby and suppressing autoboot to the preinstalled operating system partition) results in a partitioned platform, even if you do not actually create additional partitions.

All @server p5 platform IO slots (Sacs and drawers) are managed the same with respect to EEH. To return a system to a non-partitioned state, you must perform a reset.

3.3.2 Restoring a server to factory settings

You can reset a server to the factory default settings. It is recommended that you perform this task only when directed to do so by your service provider.

Attention: Before resetting a server, make sure that you have manually recorded all settings that you need to preserve. You can reset a server only if the identical level of firmware exists on both the permanent firmware boot side, also known as the P side, and the temporary firmware boot side, also known as the T side.

Resetting a server results in the loss of all system settings (such as the HMC access and ASMI passwords, time of day, network configuration, and hardware de configuration policies) that you may have set through user interfaces. Also, you lose the system error logs and partition-related information.

To reset a server, your authority level must be one of the following:

- Administrator

- Authorized service provider

To restore server settings to factory settings, do the following:

1. On the ASMI Welcome pane, specify your user ID and password, and click **Log In**.
2. In the navigation area, expand System Service Aids.
3. Select **Factory Configuration**.
4. Select the options that you want to restore to factory settings.
5. Click **Continue**. The service processor reboots.

3.4 Partition Load Manager

The Partition Load Manager (PLM) is a utility that can redistribute processors and memory resources automatically between partitions that are running AIX 5L Version 5.3. The PLM server monitors the processor and memory load in the managed partitions using the AIX Resource Management and Control subsystem. Based on an administrator defined policy, the PLM server orchestrates the movement of processor and memory resources between the partitions by communicating with the Resource Management and Control client and the HMC.

The PLM software is part of the Advanced POWER Virtualization feature on @server p5 servers and helps you maximize the dynamic utilization of processor and memory resources of partitions.

PLM provides automated processor and memory resource management across DLPAR capable logical partitions running AIX 5L. PLM allocates resources to partitions on-demand, within the constraints of a user-defined policy. Partitions with a high demand for resources are given resources from partitions with a lower demand, improving the overall resource utilization of the system. Resources that would otherwise be unused, if left allocated to a partition that was not utilizing them, can now be used to meet resource demands of other partitions in the same system. Figure 3-11 on page 72 shows how PLM functionality can improve partition resource utilization.

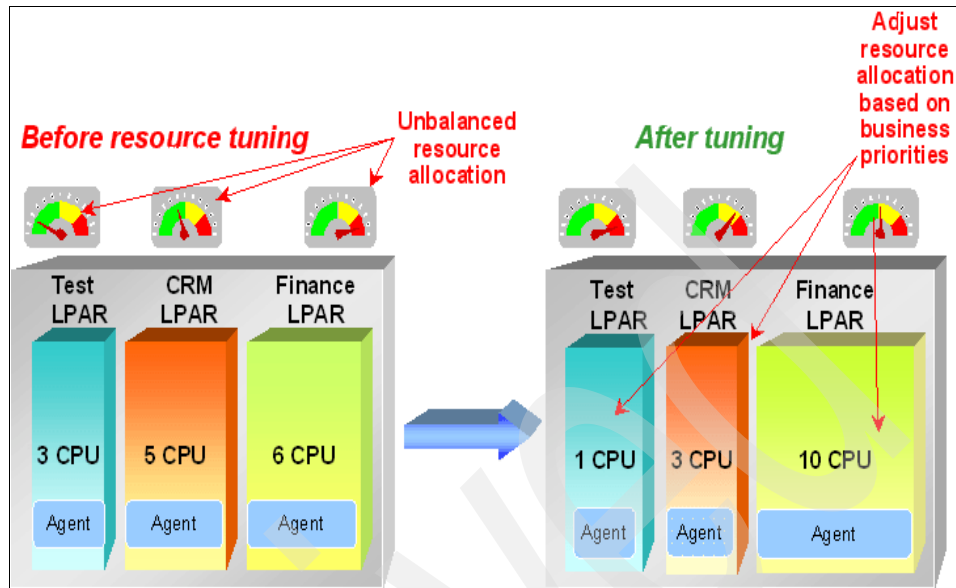


Figure 3-11 Partition Load Manager functionality

PLM uses a client-server model to report and manage resource utilization. The clients, or managed partitions, notify the PLM server when resources are either not used enough or are overused. Upon notification of one of these events, the PLM server makes resource allocation decisions based on a user-defined resource management policy. This policy determines how much of the available resources are to be allocated to each partition.

PLM works much like any other system management software in that it allows you to view the resources across your partitions, group those resources into manageable chunks, allocate and reallocate those resources within or across the groups, and maintain local logs of activity on the partitions.

PLM is a resource manager that assigns and moves resources based on defined policies and utilization of the resources. PLM manages memory, both dedicated processors and partitions, using Micro-Partitioning technology to readjust the resources. This adds additional flexibility on top of the micro-partitions flexibility that is added by the hypervisor.

PLM, however, has no knowledge about the importance of a workload running in the partitions and cannot re-adjust priority based on the changes of types of workloads. PLM does not manage Linux and i5/OS partitions. Figure 3-12 on page 73 shows a comparison of features between PLM and the hypervisor.

PLM Differentiation	Capability	PLM	P5 PHYP
HW Support	POWER4 PLM automates DLPAR adjustment for P4 install base	X	
	POWER5	X	X
OS Support	AIX 5.2 PLM runs on AIX 5.2 on P4 and P5 systems (through PRPQ)	X	
	AIX 5.3	X	X
	pLinux		X
Physical Processor Management	Dedicated PLM runs on AIX 5.2 and/or P4 systems	X	
	Capped shared	X	
	Uncapped shared	X	X
Virtual Processor Management	Virtual processor minimization for efficiency	X	
	Virtual processor adjustment for physical processor growth	X	
Physical Memory Management	Share-based	X	
	Minimum and maximum entitlements	X	
Management Policy	Entitlement-based	X	X
	Goal-based		
	Application/middleware instrumentation required		
Management Domains	Multiple management domains on a single CEC	X	
	Cross platform (CEC)		
Administration	Simple administration	X	X
	Centralized LPAR monitoring (PLM command provides usage stats)	X	
	TOD-driven policy adjustment (PLM command supports new policy load based as TOD)	X	

Figure 3-12 Comparison of features of PLM and hypervisor

PLM is set up in a partition or on another system running AIX 5L Version 5.2 ML4 or AIX 5L Version 5.3. Linux for PLM and the clients is not available. You can have other installed applications on the partition or system running PLM as well. A single instance of PLM can only manage a single server.

To configure PLM, you can use the command line interface or the Web-based System Manager for graphical setup.

Figure 3-13 on page 74 shows an overview of the components of PLM.

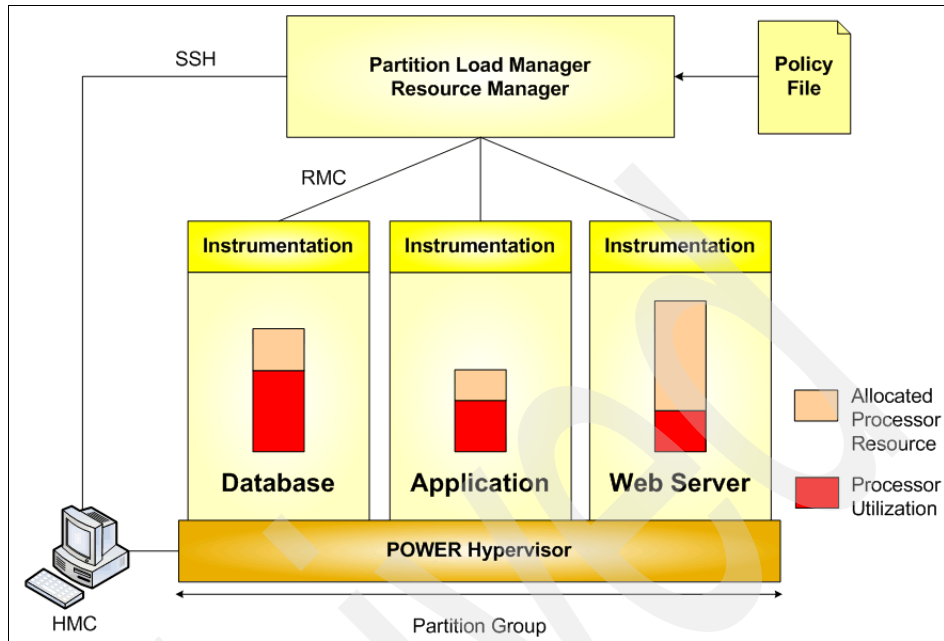


Figure 3-13 PLM overview

The policy file defines managed partitions, their entitlements, and their thresholds and organizes the partitions into groups. Every node managed by PLM must be defined in the policy file along with several associated attribute values:

- ▶ Optional maximum, minimum, and guaranteed resource values
- ▶ The relative priority or weight of the partition
- ▶ Upper and lower load thresholds for resource event notification

For each resource (processor and memory), the administrator specifies an upper and a lower threshold for which a resource event should be generated. You can also choose to manage only one resource.

Partitions that have reached an upper threshold become resource *requesters*. Partitions that have reached a lower threshold become resource *donors*. When a request for a resource is received, it is honored by taking resources from one of three sources when the requester has not reached its maximum value:

- ▶ A pool of free, unallocated resources
- ▶ A resource donor
- ▶ A lower priority partition with excess resources over entitled amount

As long as there are resources available in the free pool, they are given to the requester. If there are no resources in the free pool, the list of resource donors is checked. If there is a resource donor, the resource is moved from the donor to the requester. The amount of resource moved is the minimum of the delta values for the two partitions, as specified by the policy. If there are no resource donors, the list of excess users is checked.

When determining if resources can be taken from an excess user, the weight of the partition is determined to define the priority. Higher priority partitions can take resources from lower priority partitions. A partition's priority is defined as the ratio of its excess to its weight, where excess is expressed with the formula (current amount - desired amount) and weight is the policy-defined weight. A lower value for this ratio represents a higher priority. Figure 3-14 shows an overview of the process for partitions.

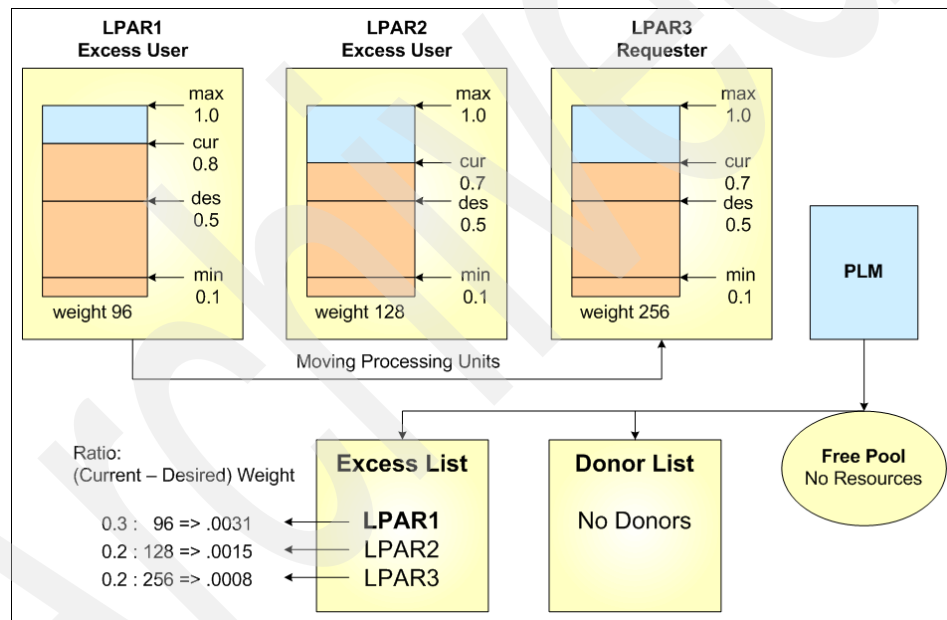


Figure 3-14 PLM resource distribution for partitions

In Figure 3-14, all partitions are capped partitions. LPAR3 is under heavy load and over its high processor average threshold value becoming a requester. There are no free resources in the free pool and no donor partitions available. PLM then checks the excess list to find a partition that has resources allocated over its guaranteed value and with a lower priority. Calculating the priority, LPAR1 has the highest ratio number and therefore the lowest priority. PLM de-allocates resources from LPAR1 and allocates them to LPAR3.

If the request for a resource cannot be honored, it is queued and re-evaluated when resources become available. A partition cannot fall below or rise above its maximum definition for each resource.

The policy file, once loaded, is static and has no knowledge of the nature of the workload on the managed partitions. A partition's priority does not change with the arrival of high priority work. The priority of partitions can only be changed by an action external to PLM by loading a new policy.

PLM handles memory and both types of processor partitions (dedicated and shared processor). All the partitions in a group must be of the same processor type.

3.4.1 Managing memory

PLM manages memory by moving logical memory blocks across partitions. To determine when there is demand for memory, PLM uses two metrics:

- ▶ Utilization percentage (ratio of memory in use to available memory)
- ▶ The page replacement rate

For workloads that result in significant file caching, the memory utilization on AIX can never fall below the specified lower threshold. With this type of workload, a partition can never become a memory donor, even if the memory is not currently being used.

In the absence of memory donors, PLM can only take memory from excess users. Because the presence of memory donors cannot be guaranteed and is unlikely with some workloads, memory management with PLM may only be effective if there are excess users present. One way to ensure the presence of excess users is to assign each managed partition a low guaranteed value, such that it will always have more than its guaranteed amount. With this sort of policy, PLM can redistribute memory to partitions based on their demand and priority.

3.4.2 Managing processors

For dedicated processor partitions, PLM moves physical processors, one at a time, from partitions that are not using them to partitions that have demand for them. This movement enables dedicated processor partitions that are running AIX 5L Version 5.2 and AIX 5L Version 5.3 to better use their resources. If one partition needs more processor capacity, PLM automatically moves processors from a partition that has idle capacity.

For shared processor partitions, PLM manages the entitled capacity and the number of virtual processors for capped or uncapped partitions. When a partition has requested more processor capacity, PLM increases the entitled capacity for

the requesting partition if additional processor capacity is available. For uncapped partitions, PLM can increase the number of virtual processors to increase the partition's potential to consume processor resources under high load conditions. Conversely, PLM also decreases entitled capacity and the number of virtual processors under low-load conditions to more efficiently use the underlying physical processors.

With the goal of maximizing a partition's and the system's ability to consume available processor resources, the administrator now can:

1. Configure partitions that have high workload peaks as uncapped partitions with a large number of virtual processors. This approach has the advantage of allowing these partitions to consume more processor resource when it is needed and available, with very low latency and no dynamic reconfiguration. For example, consider a 16-way system that uses two highly loaded partitions that are configured with eight virtual processors each, in which case, all physical processors could have been fully used. The disadvantage of this approach is that when these partitions are consuming at or below their desired capacity, there is an overhead that is associated with the large number of virtual processors defined.
2. Use PLM to vary the capacity and number of virtual processors for the partitions. This approach has the advantages of allowing partitions to consume all of the available processor resource on demand, and it maintains a more optimal number of virtual processors. The disadvantage to this approach is that since PLM performs dynamic reconfiguration operations to shift capacity to and from partitions, there is a much higher latency for the reallocation of resources. Though this approach offers the potential to more fully use the available resource in some cases, it significantly increases the latency for redistribution of available capacity under a dynamic workload, because dynamic reconfiguration operations are required.

3.4.3 Limitations and considerations

Consider the following limitations when managing your system with PLM:

- ▶ You can use PLM in partitions that are running AIX 5L Version 5.2 ML4 or AIX 5L Version 5.3. Linux or iOS support is not available.
- ▶ A single instance of PLM can only manage a single server. However, you can run multiple instances of PLM on a single system, each managing a different server.
- ▶ PLM cannot move I/O resources between partitions. Only processor and memory resources can be managed by PLM.
- ▶ PLM requires HMC Release 3 Version 2.6 or higher on an HMC and a @server p5 system.

3.4.4 Installing Partition Load Manager

To install PLM, complete the following steps:

1. Mount the PLM CD to your system.
2. Using either the **installp** command or the smitty **install_latest** fastpath, install the following filesets:
 - plm.server
 - plm.sysmgt
3. When PLM is installed, install and configure OpenSSH.
4. Run **p1msetup** for the managed partitions.

3.4.5 Querying partition status

Any user can run the **x1p1m** command to obtain status information for running instances of PLM. To query the status of all running instances of PLM, type the following command:

```
x1p1m -Q
```

A list of the instances that are running is displayed (see Figure 3-15). If there are no instances running, no output is displayed.

Name	Minimum	Guaranteed	Maximum	Share	Current	Use %	Load average
Server-9117-570-SN...			4.00				
--- Benchmark1			4.00				
----- basil.austin.ib...	0.80	2.00	4.00	128	0.81	1.23	0.23
----- sage.austin.ib...	0.80	2.00	4.00	128	0.81	1.23	0.13

Figure 3-15 PLM, Show LPAR Statistics

You can allocate resources to specific partitions and even reserve resources for specific partitions regardless of when those partitions will use the resources. The **x1p1m -R** command allows you to reserve and allocate resources from a group of managed partitions. Those resources that are reserved can be used to create a

new unmanaged partition or to make room for a new partition to enter the managed group.

Reserved resources will not be allocated to any existing partition in a group unless they are first released. If a previously offline partition comes online and enters a managed group, any reserved resources within that group are removed automatically from the collection of reserved resources, called the *free pool*, and are assigned to the new partition. If the reserved resources are used instead to create a new, unmanaged partition, they can be released back to the group after the new partition has booted and can then be reclaimed automatically by the managed group if they later become available and are needed.

The requested reservation amount is absolute, so a reserve command can result in either a reserve or a release, depending on the current reservation amount. The minimum allowed changes in the reservation amounts are:

- ▶ 1 MB for memory
- ▶ 1 processor unit for a dedicated processor group
- ▶ 0.01 processor unit for a share processor group

When you reserve resources, the free pool for the target group is first checked for available resources. If the free pool has enough resources to satisfy the request, the requested amount is removed from the free pool. If the free pool does not have enough resources to satisfy the request, resources are taken from one or more partitions with the lowest workload or the least need for the resources. A reservation request fails if the requested amount is more than the minimum that is allowed for the group.

3.4.6 Managing memory resource requests

The following is an example of how to use PLM to manage memory resource requests. This example shows how PLM responds to memory resource requests between two partitions:

The two partitions, LP0 and LP1, have these attributes:

LP0: Minimum = 1024 MB
Guaranteed = 1024 MB
Maximum = 4096 MB
Weight = 2
Current Entitlement = 1024 MB

LP1: Minimum = 1024 MB
Guaranteed = 1024 MB
Maximum = 4096 MB
Current Entitlement = 1024 MB
Weight = 1

The total amount of memory that the PLM manages is 5120 MB. With each partition's current memory allocation, shown as Current Entitlement = 1024 MB, that leaves 3072 MB that the PLM assumes is unallocated and available.

If both partitions become loaded in terms of memory use, then events that demand more memory resources are generated and sent to the PLM server. For each event received, PLM tags the partition as a taker. At the same time, PLM checks whether the partition is currently using more than its guaranteed amount. If so, the partition is tagged as an excess user. Because there are available resources, PLM satisfies the request immediately and allocates memory in the amount of mem_increment (defined either in the PLM policy or by the internal default value) to the partition from the available memory. After the available memory is depleted, the new entitlement allocations are:

LP0: Current Entitlement = 2560 MB

LP1: Current Entitlement = 2560 MB

Even with the current allocations, the partitions continue to generate events that demand more memory resources.

For each event, PLM continues to tag the partition as a taker and excess user because the partition has more resources allocated than are shown as its guaranteed entitlement. However, because there are no available resources, the request is queued if there are no other resource donors or any other excess users. When the request from the second partition is received, it is also marked as a taker and an excess user. Because there is an excess user already queued, PLM can satisfy the resource request.

Because both LP0 and LP1 are takers and excess users, PLM uses the weight that is associated with each as the determining factor of how the extra entitlement (the sum of the current entitlement for each partition minus the sum of each partition's guaranteed allotment) will be distributed between the two partitions.

In this example, of the extra 3072 MB, the LP0 partition should be allocated 2048 MB and the LP1 partition should be allocated 1024 MB. PLM assigns the mem_increment MB of memory from the LP1 partition to the LP0 partition.

With constant memory requests from each partition, PLM eventually distributes the memory so that current entitlements become the following:

LP0: Current Entitlement = 3072 MB

LP1: Current Entitlement = 2048 MB

3.4.7 Processor resources in a shared partition environment

The following example describes how PLM manages processor resources in a shared partition environment. The two partitions are configured as follows:

LP0: Minimum = 0.1
Guaranteed = 0.5
Maximum = 2.0
Max entitlement per virtual processor = 0.8
Weight = 3
Current entitlement = 0.1
Current number of virtual processors = 1

LP1: Minimum = 0.1
Guaranteed = 0.5
Maximum = 2.0
Max entitlement per virtual processor = 0.8
Weight = 1
Current entitlement = 0.1
Current number of virtual processors = 1

The total amount of processor entitlement managed by PLM is 2.0. The amount that is currently allocated to each partition, 0.1, leaves 1.8 of unallocated processor entitlement that PLM can distribute.

If both partitions begin running processor-intensive jobs, they request more processor entitlement by sending requests to PLM. PLM then tags the demanding partitions as takers and as excess users if the current entitlement is above its guaranteed value.

In addition to managing processor entitlement, PLM also manages the number of virtual processors. When either partition's current entitlement exceeds 0.8, a virtual processor is also added.

In this example, PLM assigns the available entitlement until the partitions reach the following state:

LP0: Current entitlement = 1.0
Current number of virtual processors = 2

LP1: Current entitlement = 1.0
Current number of virtual processors = 2

If the partitions continue to demand more resource, then PLM redistributes the assigned entitlement based on the weight and excess entitlement. In this example, between the LP0 partition and the LP1 partition, the total excess amount is 1.5. Because LP0 has a weight of 3 and LP1 has a weight of 1, PLM removes processor entitlement from the LP1 partition and reassigns it to the LP0

partition. If both partitions remain busy, then the resource allocation becomes the following:

- LP0:** Current entitlement = 1.25
 Current number of virtual processors = 2
- LP1:** Current entitlement = 0.75
 Current number of virtual processors = 2

Virtualized resource management

This chapter provides detailed information about the new features and their capabilities that are available in IBM @server p5 servers. It discusses the following topics:

- ▶ 4.1, “Micro-Partitioning technology” on page 84
- ▶ 4.2, “Advanced Virtualization” on page 97
- ▶ 4.3, “Introduction to Virtual I/O Server” on page 104
- ▶ 4.4, “Virtual I/O Server and virtualization configuration” on page 117

If you are a system administrator who has responsibility for configuration and management of POWER5-based servers with these advanced capabilities, it is imperative that you become familiar with the aspects that this chapter describes before you run the system with these features enabled.

4.1 Micro-Partitioning technology

Micro-Partitioning technology allows the resource definition of a partition to allocate fractions of processors to the partition. On POWER4 systems, all partitions are considered *dedicated*. The processors that are assigned to a partition can only be in whole multiples and only used by that partition. On POWER5 systems, you can choose between *dedicated* processor partitions and *shared* processor partitions using Micro-Partitioning technology. You can have both dedicated and shared processor partitions running on the same system at the same time. At the time of the writing of this book, you can have only one shared processor pool per system.

Micro-Partitioning technology allows for increased overall use of system resources by applying automatically only the required amount of processor resources that each partition needs. Resources can also be defined as increments greater than a single processor.

The hypervisor continually adjusts the amount of processor capacity that is allocated to each shared processor partition and any excess capacity that is unallocated based on current partition profiles within a shared pool. Tuning parameters allow the administrator extensive control over the amount of processor resources that each partition can use.

This section discusses the following topics of Micro-Partitioning technology:

- ▶ “Shared processor partitions” on page 84
- ▶ “Processing units of capacity” on page 86
- ▶ “Capped and uncapped mode” on page 88
- ▶ “Virtual processors” on page 89
- ▶ “Dedicated processors” on page 91
- ▶ “Capped and uncapped processing units” on page 93
- ▶ “Dynamic processor deallocation and sparing” on page 96

4.1.1 Shared processor partitions

The virtualization of processors enables the creation of a partitioning model which is fundamentally different from the POWER4 systems where whole processors are assigned to partitions and are owned by them. In the new model, physical processors are abstracted into virtual processors that are then assigned to partitions. However, the underlying physical processors are shared by these partitions.

Virtual processor abstraction is implemented in the hardware and microcode. From an operating system perspective, a virtual processor is indistinguishable from a physical processor. The key benefit of implementing partitioning in the

hardware allows any operating system to run on POWER5 technology with little or no changes. Optionally, for optimal performance, the operating system can be enhanced to exploit shared processor pools more in-depth (for instance, by voluntarily relinquishing processor cycles to the hardware when they are not needed). AIX 5L Version 5.3 is the first version of AIX 5L that includes such enhancements.

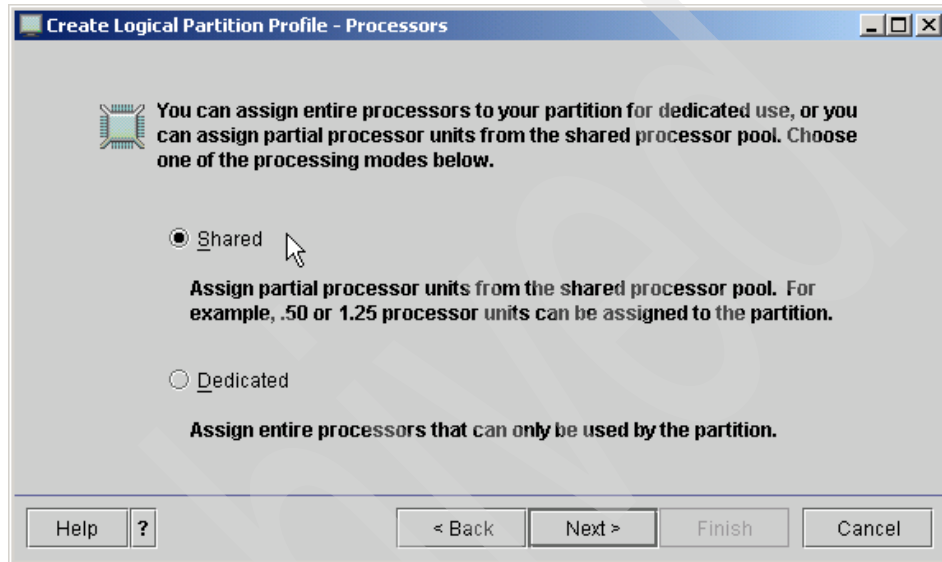


Figure 4-1 Create Logical Partition Profile - shared processors

Micro-Partitioning technology allows for multiple partitions to share one physical processor. Partitions using Micro-Partitioning technology are referred to as shared processor partitions.

A partition may be defined with a processor capacity as small as 10 processor units. This represents 1/10 of a physical processor. Each processor can be shared by up to 10 shared processor partitions. The shared processor partitions are dispatched and time-sliced on the physical processors that are under control of the hypervisor.

Micro-Partitioning technology is supported across the entire POWER5 product line from the entry-level to the high-end systems. Table 4-1 on page 86 shows the maximum number of logical partitions and shared processor partitions of the different models.

Table 4-1 Micro-Partitioning technology overview on @server p5 systems

@server p5 servers	p5-510	p5-520	p5-550	p5-570	p5-590	p5-595
Processors	2	2	4	16	32	64
Dedicated processor partitions	2	2	4	16	32	64
Shared processor partitions	10	20	40	160	254	254

Note: The maximums listed in Table 4-1 are supported by the hardware. However, the practical limits based on production workload demands might be significantly lower.

Shared processor partitions still need dedicated memory, but the partition I/O requirements can be supported through Virtual Ethernet and Virtual SCSI. Using all virtualization features, up to 254 shared processor partitions are supported in p5-590 and p5-595 systems.

The shared processor partitions are created and managed by the HMC. When you start creating a partition, you have to choose between a shared processor partition and a dedicated processor partition (see Figure 4-1 on page 85).

When setting up a partition, you have to define the resources that belong to the partition, such as memory and IO resources. For shared processor partitions you have to configure the following additional options:

- ▶ Minimum, desired, and maximum processing units of capacity.
- ▶ The processing sharing mode to capped or uncapped. If the partition is uncapped, you must set its variable capacity weight also.
- ▶ Minimum, desired and maximum virtual processors.

4.1.2 Processing units of capacity

Processing capacity can be configured in fractions of 1/100 of a processor. The minimum amount of processing capacity which has to be assigned to a partition is 1/10 of a processor.

On the HMC, processing capacity is specified in terms of *processing units*. The minimum capacity of 1/10 of a processor which has to be assigned to a partition is specified as 0.1 processing units. To assign a processing capacity representing 75% of a processor, 0.75 processing units are specified on the HMC.

On a system with two processors a maximum of 2.0 processing units can be assigned to partition. Processing units specified on the HMC are used to quantify the minimum, desired, and maximum amount of processing capacity for a partition (see Figure 4-2).

Create Logical Partition Profile - Processing Settings

Specify the desired, minimum, and maximum processing settings in the fields below.

Total usable processing units: 2.00

Desired processing units: 0.5

Minimum processing units: 0.1

Maximum processing units: 1

(Advanced...)

Help ? < Back Next > Finish Cancel

Figure 4-2 Choose desired, minimum, and maximum processing units

After a partition is activated, processing capacity is usually referred to as capacity entitlement or entitled capacity. Figure 4-3 on page 88 shows a graphic representation of the definitions of processor capacity.

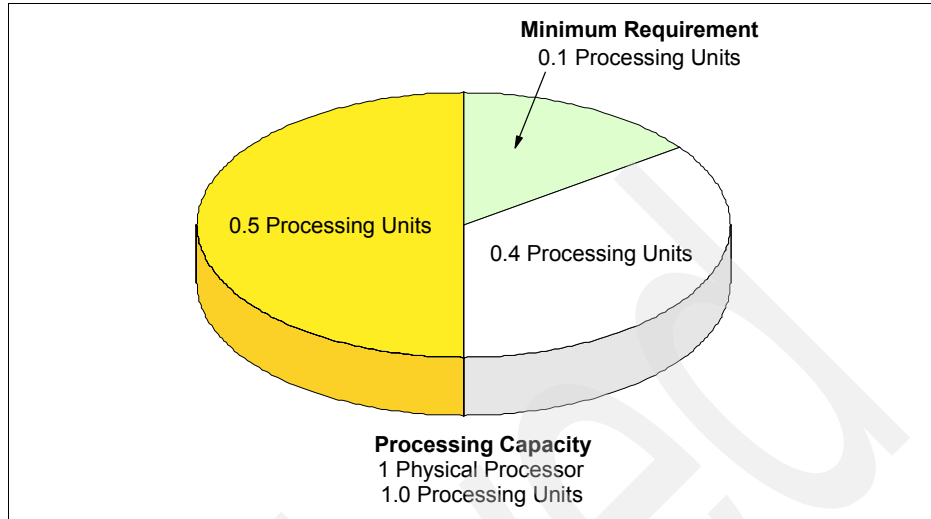


Figure 4-3 Processing units of capacity

4.1.3 Capped and uncapped mode

The next step in defining a shared processor partition is to define whether the partition is running in a capped or uncapped mode (see Figure 4-4 on page 89).

Capped mode The processing units never exceed the assigned processing capacity.

Uncapped mode The processing capacity can be exceeded when the shared pool has available resources.

When a partition is running in an uncapped mode, you must specify the uncapped weight of that partition.

If multiple uncapped logical partitions require idle processing units, the managed system distributes idle processing units to the logical partitions in proportion to each logical partition's uncapped weight. The higher the uncapped weight of a logical partition, the more processing units the logical partition gets.

The uncapped weight must be a whole number from 0 to 255. The default uncapped weight for uncapped logical partitions is 128. A partition's share is computed by dividing its variable capacity weight by the sum of the variable capacity weights for all uncapped partitions. If you set the uncapped weight at 0, the managed system treats the logical partition as a capped logical partition. A logical partition with an uncapped weight of 0 cannot use more processing units than those that are committed to the logical partition.

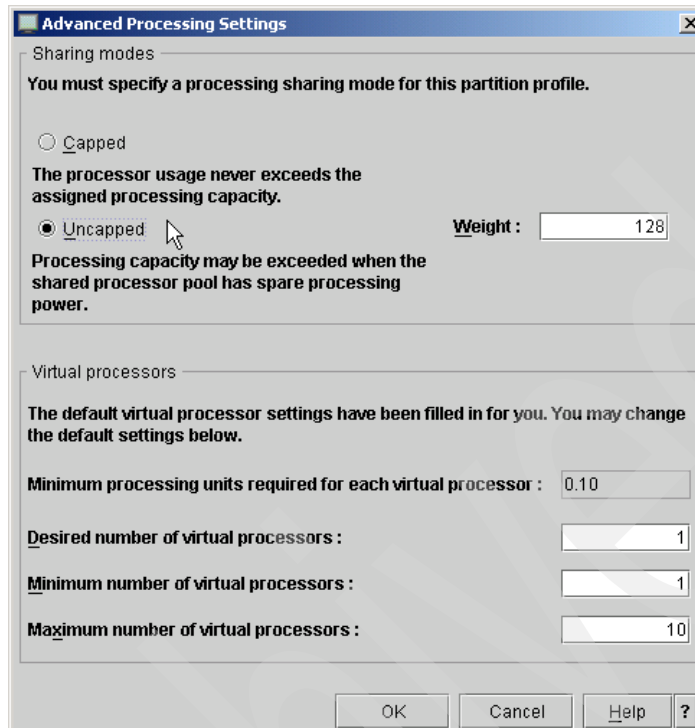


Figure 4-4 Specify processing sharing mode and weight

4.1.4 Virtual processors

Virtual processors are the whole number of concurrent operations that the operating system can use. The processing power can be conceptualized as being spread equally across these virtual processors. Selecting the optimal number of virtual processors depends on the workload in the partition. Some partitions benefit from greater concurrence, where other partitions require greater power.

By default, the number of processing units that you specify is rounded up to the minimum number of virtual processors that are needed to satisfy the assigned number of processing units. The default settings maintains a balance of virtual processors to processor units. For example:

- ▶ If you specify 0.50 processing units, one virtual processor are assigned.
- ▶ If you specify 2.25 processing units, three virtual processors are assigned.

You also can use the advanced tab in your partitions profile to change the default configuration and to assign more virtual processors (see Figure 4-5 on page 90).

At the time of the writing of this book, the maximum number of virtual processors per partition is 64.

A logical partition in the shared processing pool has at least as many virtual processors as its assigned processing capacity. By making the number of virtual processors too small, you limit the processing capacity of an uncapped partition. If you have a partition with 0.50 processing units and 1 virtual processor, the partition cannot exceed 1.00 processing units, because it can only run one job at a time, which cannot exceed 1.00 processing units. However, if the same partition with 0.50 processing units was assigned two virtual processors and processing resources were available, the partition could use an additional 1.50 processing units.

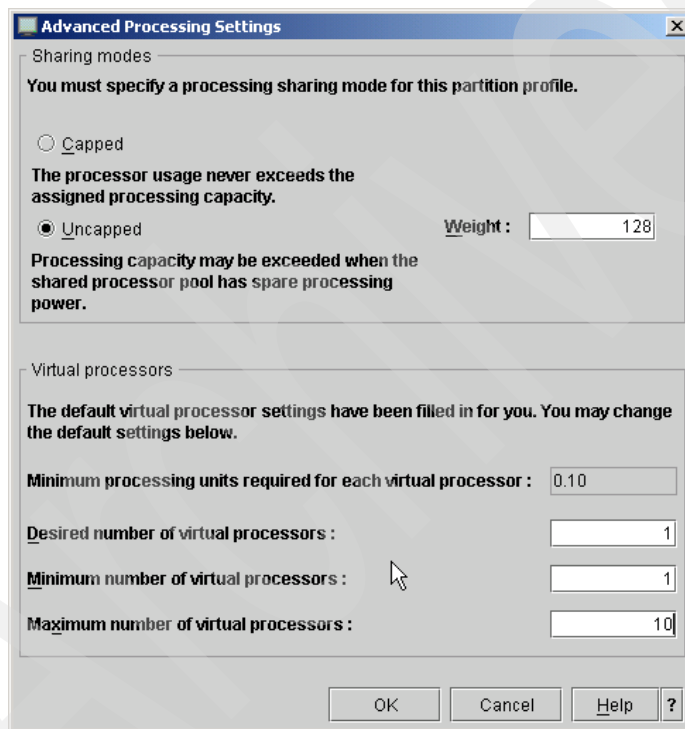


Figure 4-5 Specify number of virtual processors

4.1.5 Dedicated processors

Dedicated processors are whole processors that are assigned to a single partition. If you choose to assign dedicated processors to a logical partition, you must assign at least one processor to that partition (see Figure 4-6).

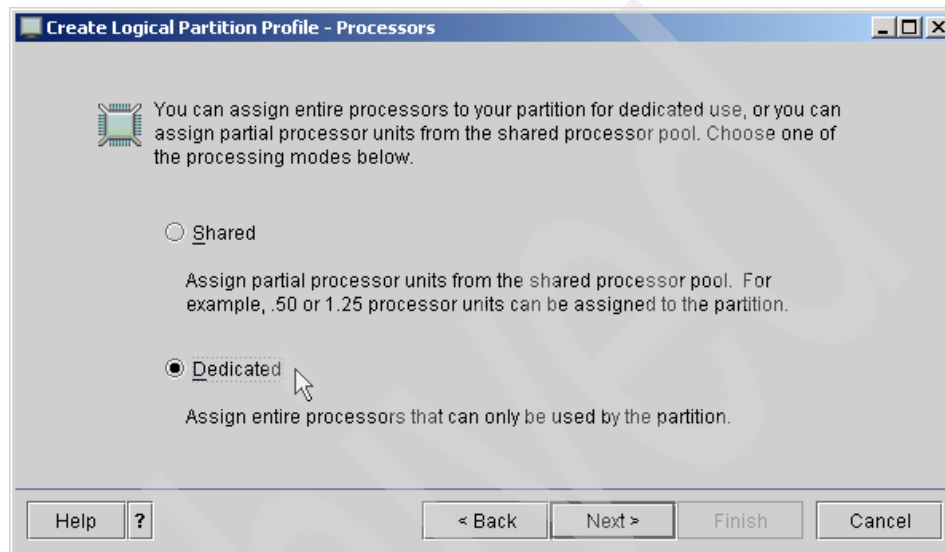


Figure 4-6 Create Logical Partition Profile, dedicated processors

You cannot mix shared processors and dedicated processors in one partition.

By default, a powered-off logical partition using dedicated processors has its processors available to the shared processing pool. When the processors are in the shared processing pool, an uncapped partition that needs more processing power can use the idle processing resources. However, when you power on the dedicated partition while the uncapped partition is using the processors, the activated partition regains all of its processing resources. If you want to allow this partition the ability to collect shared pool statistics, you can do so under the hardware tab (see Figure 4-7 on page 92).

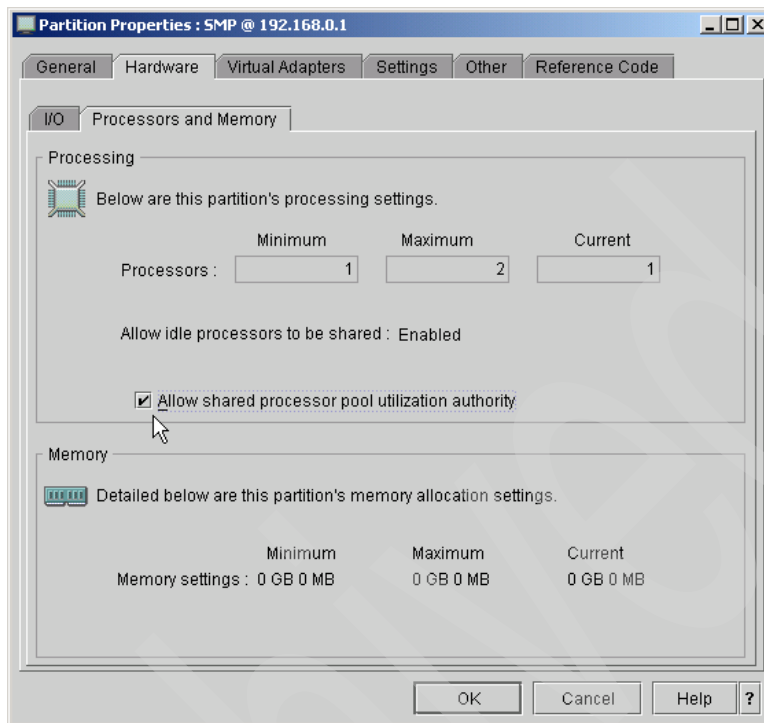


Figure 4-7 Allow shared processor pool statistics access to a partition

To set if a partition is to use resources from the shared processor pool, this is done in the profile properties under the processors tab.

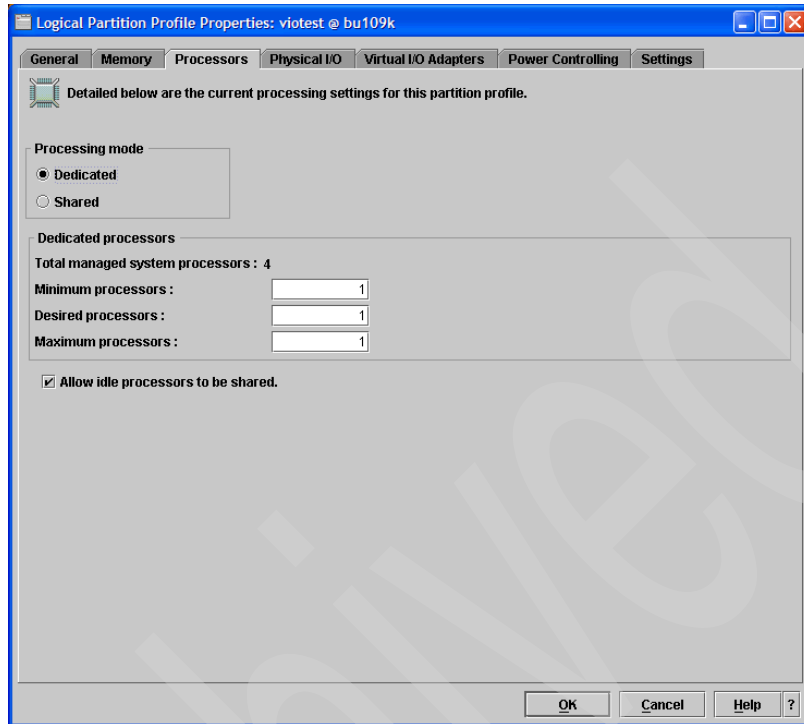


Figure 4-8 Allow idle processors to be shared

Note: You cannot disable the *Allow idle processor to be shared* function when you create a partition. You need to open the properties for the created partition and change it on the processor tab.

4.1.6 Capped and uncapped processing units

The hypervisor schedules shared processor partitions from a set of physical processors that is called the *shared processor pool*. By definition, these processors are not associated with dedicated partitions.

In shared partitions, there is no fixed relationship between virtual processors and physical processors. The hypervisor can use any physical processor in the shared processor pool when it schedules the virtual processor. By default, it attempts to use the same physical processor, but this cannot always be guaranteed. The hypervisor uses the concept of a home node for virtual processors, enabling it to select the best available physical processor from a memory affinity perspective for the virtual processor that is to be scheduled.

Affinity scheduling is designed to preserve the content of memory caches, so that the working data set of a job can be read or written in the shortest time period possible. Affinity is actively managed by the hypervisor, because each partition has a completely different context. Currently, there is one shared processor pool, so all virtual processors are implicitly associated with the same pool.

Figure 4-9 shows the relationship between two partitions using a shared processor pool of a single physical processor. One partition has two virtual processors, and the other, a single one. The figure also shows how the capacity entitlement is evenly divided over the number of virtual processors.

When you set up a partition profile, you set up the desired, minimum, and maximum values that you want for the profile. When a partition is started, the system chooses the partition's entitled processor capacity from this specified capacity range. The value that is chosen represents a commitment of capacity that is reserved for the partition. This capacity cannot be used to start another shared partition. Otherwise, capacity could be overcommitted.

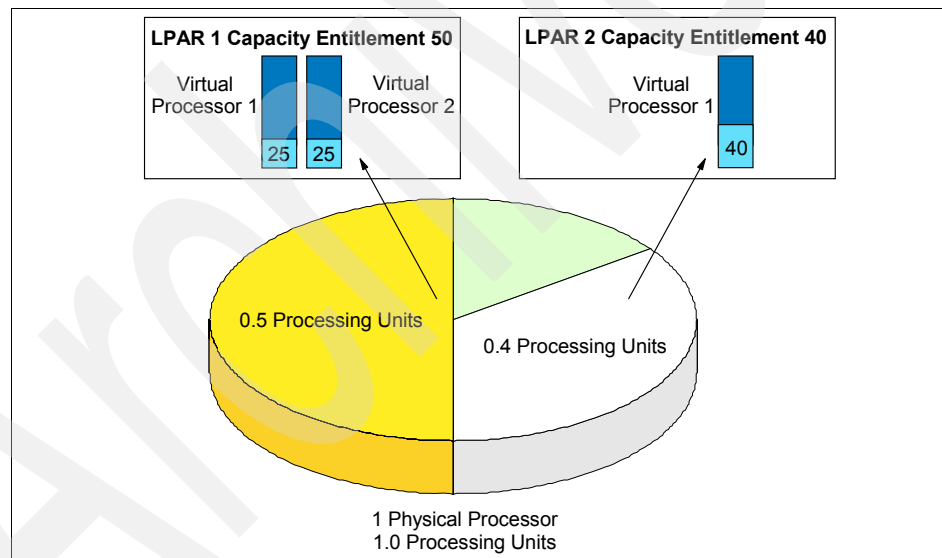


Figure 4-9 Distribution of capacity entitlement on virtual processors

When starting a partition, preference is given to the desired value, but these values cannot always be used, because there may not be enough unassigned capacity in the system. In that case, a different value is chosen, which must be greater than or equal to the minimum capacity attribute. Otherwise, the partition cannot be started. The entitled processor capacity is distributed to the partitions

in sequence the partitions are started. For example a shared pool has 2.0 processing units available.

Partitions 1, 2, and 3 are activated in sequence

1. Partition 1 activated
Min. = 1.0, max = 2.0, desired = 1.5
Allocated capacity entitlement: 1.5
2. Partition 2 activated
Min. = 1.0, max = 2.0, desired = 1.0
Partition 2 does not start because the minimum capacity is not met
3. Partition 3 activated
Min. = 0.1, max = 1.0, desired = 0.8
Allocated capacity entitlement: 0.5

The maximum value is only used as an upper limit for dynamic operations.

Figure 4-10 shows the usage of a capped partition of the shared processor pool. Partitions using the capped mode are not able to assign more processing capacity from the shared processor pool than the capacity entitlement will allow.

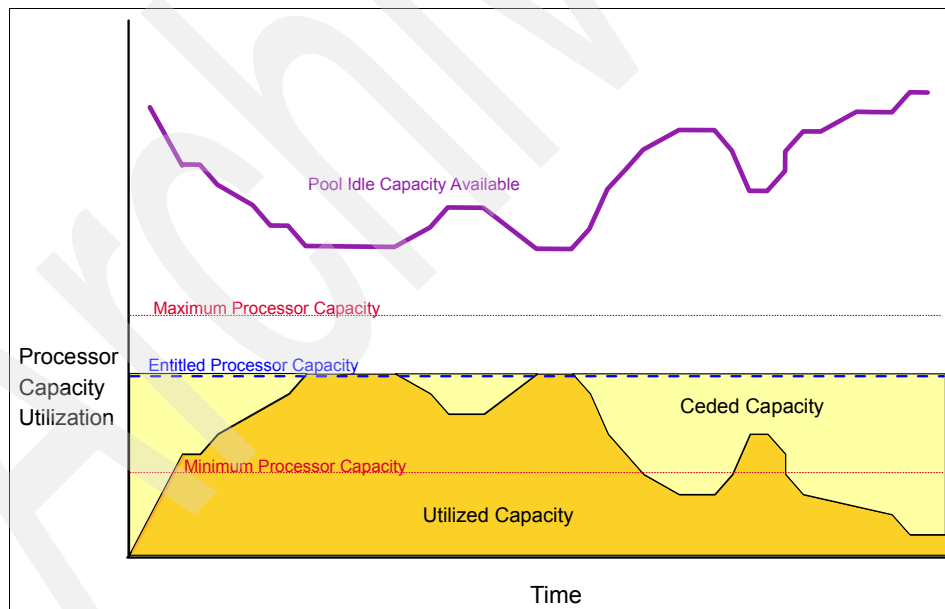


Figure 4-10 Shared capped processor partition utilization

Figure 4-11 on page 96 shows the usage of the shared processor pool by an uncapped partition. The uncapped partition is able to assign idle processing capacity if it needs more than the entitled capacity.

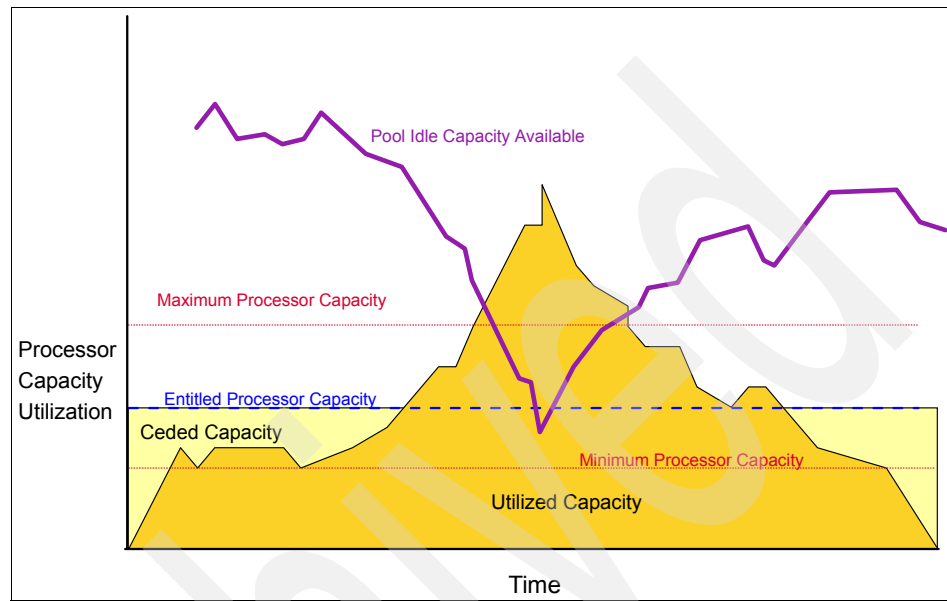


Figure 4-11 Shared uncapped processor partition utilization

4.1.7 Dynamic processor deallocation and sparing

Dynamic Processor Deallocation and Dynamic Processor Sparing is supported in the shared processor pool.

If a physical processor reaches a failure threshold and needs to be taken offline (guarded out), the hypervisor analyzes the system environment to determine what action it should take to replace the processor resource. The options for handling this condition can be one of the following:

- ▶ If there is a Capacity on Demand processor available, the hypervisor switches the processor to the shared pool transparently, and no partition loss of capacity results.
- ▶ If there is at least 1.0 unallocated processor capacity available, it can be used to replace the capacity lost due to the failing processor.

If not enough unallocated resource exists, the hypervisor determines the capacity that each partition must lose to eliminate the 1.00 processor units from the shared pool. As soon as each partition varies off the processing capacity or

virtual processors, the failing processor is taken offline by the service processor and the hypervisor.

The amount of capacity that each shared partition is requested to vary off is proportional to the total amount of entitled capacity in the partition. This amount is based on the amount of capacity that can be varied off, which is controlled by the *min* capacity of the partition. The larger the difference between the current entitled capacity of the partition and the minimum entitled capacity, the more the partition will be asked to vary off.

Note: Dynamic Memory allocation or de-allocation is not supported by Linux on @server p5 servers.

4.2 Advanced Virtualization

With the usage of virtual partitions in POWER5-based servers, the number of partitions that can be concurrently instantiated on a @server p5 server can be greater than the number of physical I/O slots in the Central Electronic Complex and its remote I/O drawers. For example, at the time of this publication, @server p5 processor-based servers support 254 logical partitions, while p5-595 with remote I/O drawers can have up to 240 I/O slots.

Typically, a small operating system instance needs at least one slot for a network interface connector and one slot for a disk adapter (SCSI, Fibre Channel), while more robust configuration often consist of two redundant network interface connector adapters and two disk adapters.

To be able to connect enough I/O devices to each partition that is configured on a @server p5 server, IBM introduced virtual I/O technology for POWER5-based servers. Virtual I/O devices are an optional feature of a partition and can be used on POWER5 based systems in conjunction with AIX 5L Version 5.3 or Linux. Virtual I/O devices are intended as a complement to physical I/O adapters (also known as dedicated or local I/O devices). A partition can have any combination of local and virtual I/O adapters.

The generic term *Virtual I/O* relates to five different concepts:

- ▶ Three adapters: Virtual SCSI, Virtual Ethernet, and Virtual Serial
- ▶ A special AIX partition, called the Virtual I/O Server
- ▶ A mechanism to link virtual network devices to real devices, called Shared Ethernet Adapter (SEA).

4.2.1 Virtual LAN

Virtual LAN (VLAN) is a technology used for establishing virtual network segments on top of physical switch devices. If configured appropriately, a VLAN definition can straddle multiple switches. Typically, a VLAN is a broadcast domain that enables all nodes in the VLAN to communicate with each other without L3 routing or inter-VLAN bridging. In Figure 4-12, two VLANs (1 and 2) are defined on three switches (A, B, and C). Although nodes C-1 and C-2 are physically connected to the same switch C, traffic between the two nodes can be blocked. To enable communication between VLAN 1 and 2, L3 routing or inter-VLAN bridging should be established between them, typically by an L3 device.

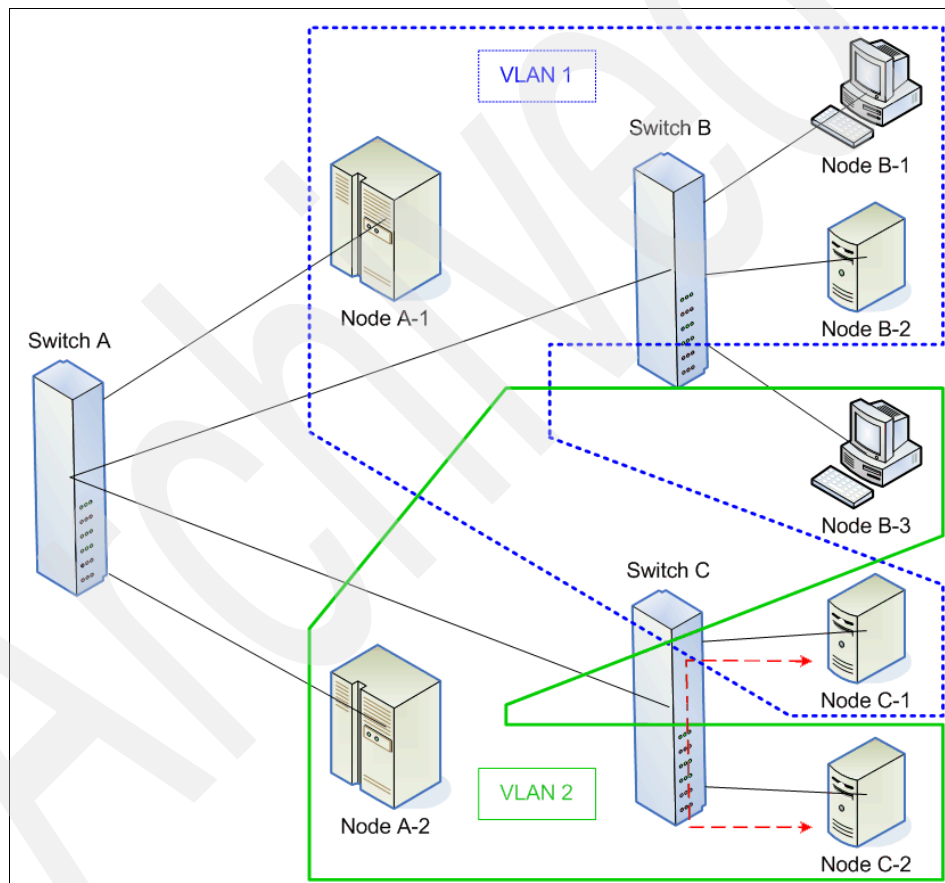


Figure 4-12 Example of a VLAN

The use of VLAN provides increased LAN security and flexible network deployment over traditional network devices. VLANs provide additional security by allowing an administrator to block packets to a domain from a separate

domain on the same switch, therefore providing an additional control on what LAN traffic is visible to specific Ethernet ports on the switch.

AIX virtual LAN support

Some of the various technologies for implementing VLANs include:

- ▶ Port-based VLAN
- ▶ Layer 2 VLAN
- ▶ Policy-based VLAN
- ▶ IEEE 802.1Q VLAN

VLAN support in AIX is based on the IEEE 802.1Q VLAN implementation. The IEEE 802.1Q VLAN is achieved by adding a VLAN ID tag to an Ethernet frame, and the Ethernet switches restrict the frames to ports that are authorized to receive frames with that VLAN ID. Switches also restrict broadcasts to the logical network by ensuring that a broadcast packet is delivered to all ports which are configured to receive frames with the VLAN ID with which the broadcast frame was tagged.

A port on a VLAN capable switch has a default Port virtual LAN ID (PVID) that indicates the default VLAN to which the port belongs. The switch adds the PVID tag to untagged packets that are received by that port. In addition to a PVID, a port may belong to additional VLANs and have those VLAN IDs assigned to it that indicates the additional VLANs to which the port belongs.

A port only accepts untagged packets or packets with a VLAN ID (PVID or additional VIDs) tag of the VLANs to which the port belongs. A port configured in the untagged mode is only allowed to have a PVID and receives untagged packets or packets tagged with the PVID. The untagged port feature helps systems that do not understand VLAN tagging to communicate with other systems using standard Ethernet.

Each VLAN ID is associated with a separate Ethernet interface to the upper layers (for example IP) and creates unique logical Ethernet adapter instances per VLAN (for example ent1 or ent2).

You can configure multiple VLAN logical devices on a single system. Each VLAN logical devices constitutes an additional Ethernet adapter instance. These logical devices can be used to configure the same Ethernet IP interfaces as are used with physical Ethernet adapters.

4.2.2 VLAN communication by example

This section discusses in more detail VLAN communication between partitions and with external networks and uses the sample configuration that is shown in

- Only packets for the VLAN that is specified as PVID are received by the network interfaces en0

Table 4-2 on page 101 lists which client partitions can communicate with each other and the network interfaces that they can use.

Table 4-2 Interpartition of VLAN communication

VLAN	Partition / Network interface
1	Partition 1 / en0 Partition 2 / en0
2	Partition 3 / en0 Partition 4 / en0
10	Partition 1 / en1 Partition 3 / en1

Communication with external networks

The SEA is configured with PVID 1 and VLAN 10. Untagged packets that are received by the SEA are tagged for VLAN 1. Handling of outgoing traffic depends on the VLAN tag of the outgoing packets. For example:

- Packets tagged with the VLAN which match the PVID of the SEA are untagged before being sent out to the external network
- Packets tagged with a VLAN *other* than that of the PVID of the SEA are sent out with the VLAN tag unmodified

In the example shown in Figure 4-13 on page 100, Partition 1 and Partition 2 have access to the external network through network interface en0 using VLAN 1. Because these packets are using the PVID, the SEA removes the VLAN tags before sending the packets to the external network.

Partition 1 and Partition 3 have access to the external network using network interface en1 and VLAN 10. Packets are sent out by the SEA with the VLAN tag. Therefore, only VLAN-capable destination devices are able to receive the packets. Table 4-3 lists this relationship.

Table 4-3 VLAN communication to external network

VLAN	Partition / Network interface
1	Partition 1 / en0 Partition 2 / en0
10	Partition 1 / en1 Partition 3 / en1

Virtual Ethernet connections

Virtual Ethernet connections supported in POWER5 systems use VLAN technology to ensure that the partitions can access only data that is directed to them. The hypervisor provides a Virtual Ethernet switch function based on the IEEE 802.1Q VLAN standard that allows partition communication within the same server. The connections are based on an implementation that is internal to the hypervisor that moves data between partitions. Figure 4-14 is an example of an inter-partition VLAN.

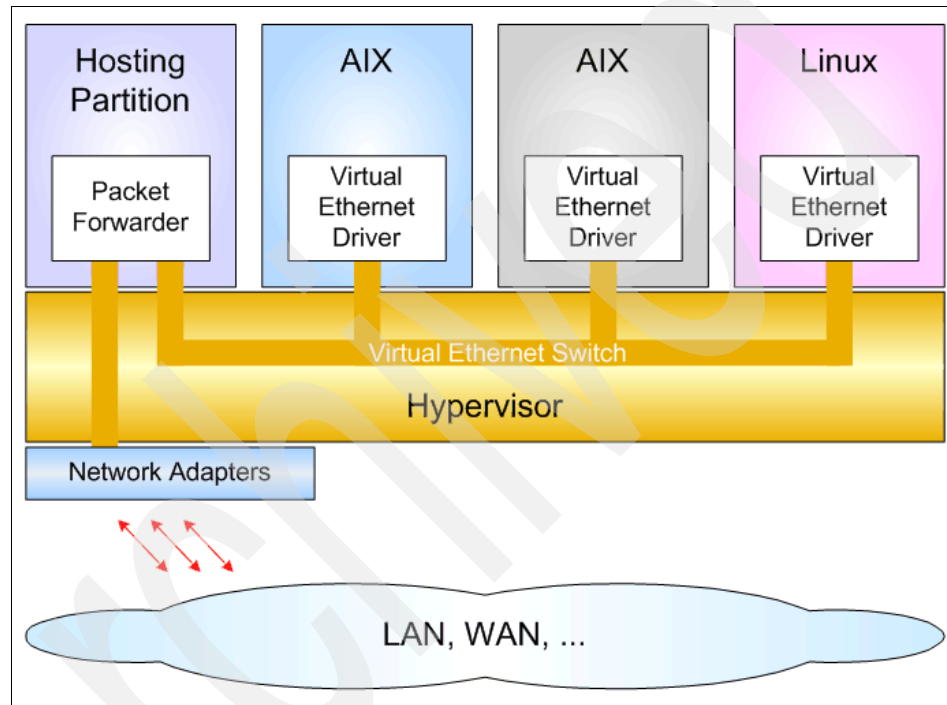


Figure 4-14 logical view of an inter-partition VLAN

Virtual Ethernet adapter concepts

Partitions that communicate through a Virtual Ethernet channel need to have an additional in-memory channel. This additional in-memory channel requires the creation of an in-memory channel between partitions on the HMC. The kernel creates a virtual device for each memory channel indicated by the firmware. The AIX configuration manager creates the device special files. A unique media access control (MAC) address is also generated when the Virtual Ethernet device is created. You can assign a prefix value for the system so that the generated MAC addresses in a system consists of a common system prefix plus an algorithmically-generated unique part per adapter.

The Virtual Ethernet can also be used as a bootable device to allow such tasks as operating system installations to be performed using network installation management (NIM).

Performance considerations

The transmission speed of Virtual Ethernet adapters is in the range of 1 to 3 Gigabits per second, depending on the transmission (maximum transmission unit) size. A partition can support up to 256 Virtual Ethernet adapters with each Virtual Ethernet capable to be associated with up to 21 VLANs (20 VID and 1 PVID).

The Virtual Ethernet connections generally take up more processor time than a local adapter to move a packet (DMA versus copy). For shared processor partitions, performance is gated by the partition definitions (for example, entitled capacity and number of processors). Small partitions communicating with each other experience more packet latency due to partition context switching. In general, high bandwidth applications *should not* be deployed in small shared processor partitions. For dedicated partitions, throughput *should be* comparable to a 1 Gigabit Ethernet for small packets providing much better performance than 1 Gigabit Ethernet for large packets. For large packets, the Virtual Ethernet communication is copy bandwidth limited.

Benefits of virtual Ethernet

Due to the number of partitions possible on many systems being greater than the number of I/O slots, Virtual Ethernet is a convenient and cost saving option to enable partitions within a single system to communicate with one another through a VLAN. The VLAN creates logical Ethernet connections between one or more partitions and is designed to help avoid a failed or malfunctioning operating system from being able to impact the communication between two functioning operating systems. The Virtual Ethernet connections may also be bridged to an external network to permit partitions without physical network adapters to communicate outside of the server.

Dynamic partitioning for virtual Ethernet devices

Virtual Ethernet resources can be assigned and removed dynamically. On the HMC, Virtual Ethernet target and server adapters can be assigned and removed from a partition using dynamic logical partitioning. The mapping between physical and virtual resources on the Virtual I/O Server can also be done dynamically.

Limitations and considerations

The following are limitations that you should consider when implementing a Virtual Ethernet:

- ▶ A maximum of up to 256 Virtual Ethernet adapters are permitted per partition.
- ▶ Virtual Ethernet can be used in both shared and dedicated processor partitions, provided that the partition is running AIX 5L Version 5.3 or Linux with the 2.6 kernel or a kernel that supports virtualization.
- ▶ A mixture of Virtual Ethernet connections, real network adapters, or both are permitted within a partition.
- ▶ Virtual Ethernet can only connect partitions within a single system.
- ▶ Virtual Ethernet requires a POWER5 system and an HMC to define the Virtual Ethernet adapters.

Virtual Ethernet uses the system processors for all communication functions instead of off loading to processors on network adapter cards. As a result there is an increase in system processor load generated by the use of Virtual Ethernet.

4.3 Introduction to Virtual I/O Server

The Virtual I/O Server is an appliance that provides virtual storage and shared Ethernet capability to client logical partitions on a POWER5 system. It allows a physical adapter with attached disks on the Virtual I/O Server partition to be shared by one or more partitions, enabling clients to consolidate and potentially minimize the number of physical adapters.

The Virtual I/O Server is the link between the virtual and the real world. It can be seen as an AIX-based appliance, and it is supported on POWER5 servers only. The Virtual I/O Server runs in a special partition which cannot be used for execution of application code.

It mainly provides two functions:

- ▶ Server component for Virtual SCSI devices (VSCSI target)
- ▶ Support of shared Ethernet adapters for Virtual Ethernet

The Virtual I/O Server is shipped as a mksysb image. Although the Virtual I/O Server is based in AIX5L Version 5.3, it is not accessible as a standard partition. Administrative access to the I/O Server partition is only possible as user padmin, not as id root. After login, user padmin gets a restricted shell, which is not escapable, called the I/O Server command line interface.

The operating system of the Virtual I/O Server is hidden to simplify transitions to later versions. Additionally, this product supports Linux and AIX 5L Version 5.3 client partitions. No specific operating system skill is required for administration of the Virtual I/O Server.

4.3.1 Shared Ethernet Adapter

You can use a Shared Ethernet Adapter (SEA) to connect a physical Ethernet to the Virtual Ethernet. An SEA also provides the possibility for several client partitions to share one physical adapter.

Connecting a virtual Ethernet to external networks

There are two ways you can connect the Virtual Ethernet that enables the communication between logical partitions on the same server to an external network.

Enabling routing capabilities

By enabling the AIX routing capabilities (ipforwarding network option), one partition with a physical Ethernet adapter connected to an external network can act as router. Figure 4-15 shows a sample configuration. In this type of configuration the partition that routes the traffic to the external work does not necessarily have to be the Virtual I/O Server as in the figure. It could be any partition with a connection to the outside world. The client partitions would have their default route set to the partition which routes traffic to the external network.

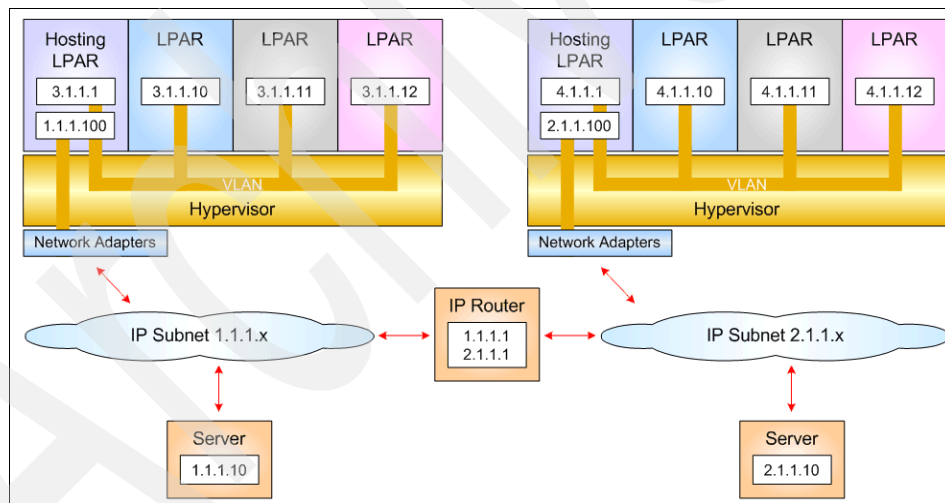


Figure 4-15 Connection to external network using AIX routing

Using Shared Ethernet Adapter

Using SEA, you can connect internal and external VLANs using one physical adapter. The SEA hosted in the Virtual I/O Server acts as a layer two switch between the internal and external network.

SEA is a new service that securely transports network traffic from a virtual Ethernet to a real network adapter. The SEA service runs in the Virtual I/O Server. It cannot be run in a general purpose AIX partition.

SEA requires the hypervisor component of POWER5 systems and therefore cannot be used on POWER4 systems. It also cannot be used with AIX 5L Version 5.2, because the device drivers for virtual Ethernet are only available for AIX 5L Version 5.3 and Linux. Thus, there is no way to connect an AIX 5L Version 5.2 system to an SEA.

The SEA allows partitions to communicate outside the system without having to dedicate a physical I/O slot and a physical network adapter to a client partition. The SEA has the following characteristics:

- ▶ Virtual Ethernet MAC addresses that are visible to outside systems
- ▶ Broadcast and multicast are supported
- ▶ ARP and NDP can work across a shared Ethernet

To bridge network traffic between the Virtual Ethernet and external networks, the Virtual I/O Server has to be configured with at least one physical Ethernet adapter. One SEA can be shared by multiple VLANs, and multiple subnets can connect using a single adapter on the Virtual I/O Server. Figure 4-16 shows a configuration example. An SEA can include up to 16 Virtual Ethernet adapters that share the physical access.

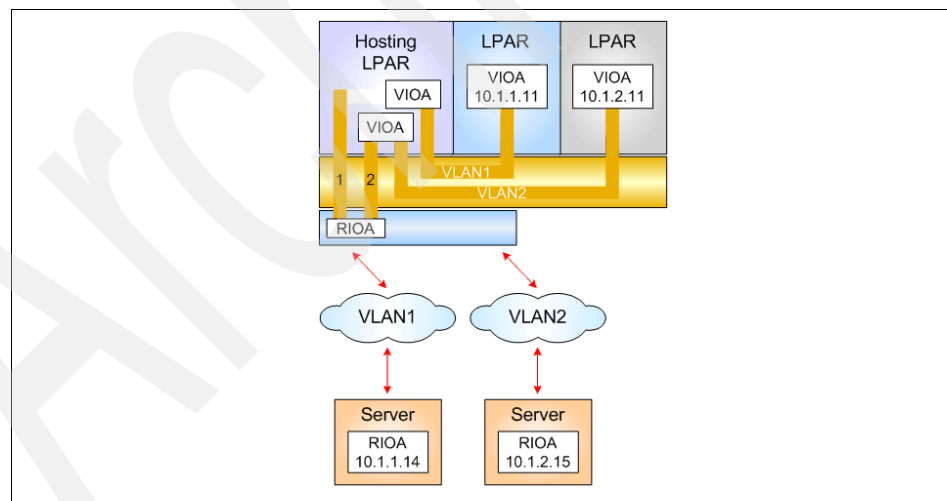


Figure 4-16 SEA configuration

A Virtual Ethernet adapter connected to the SEA must have the trunk flag set. Once an Ethernet frame is sent from the Virtual Ethernet adapter on a client partition to the hypervisor, the hypervisor searches for the destination MAC

address within the VLAN. If no such MAC address exists within the VLAN, it forwards the frame to the trunk Virtual Ethernet adapter that is defined on the same VLAN. The trunk virtual Ethernet adapter enables a layer two bridge to a physical adapter.

The shared Ethernet directs packets based on the VLAN ID tags, based on observing the packets that originate from the virtual adapters. One of the virtual adapters in the SEA is designated as the default PVID adapter. Ethernet frames without any VLAN ID tags are directed to this adapter and assigned the default PVID.

When the shared Ethernet receives IP (or IPv6) packets that are larger than the maximum transmission unit of the adapter that the packet is forwarded through, either IP fragmentation is performed and the fragments are forwarded, or an ICMP packet too big message is returned to the source when the packet cannot be fragmented.

Theoretically, one adapter can act as the only contact with external networks for all client partitions. Because the SEA is dependant on Virtual I/O, it consumes processor time for all communications. An increased amount of processor load can be generated by the use of Virtual Ethernet and SEA.

There are several different ways to configure physical and virtual Ethernet adapters into SEAs to maximize throughput.

- ▶ Using Link Aggregation (EtherChannel), several physical network adapter can be aggregated.
- ▶ Using several SEAs provides more queues and more performance. An example for this configuration is shown in Figure 4-17 on page 108.

Other aspects to take into consideration are availability and the possibility of connecting to different networks.

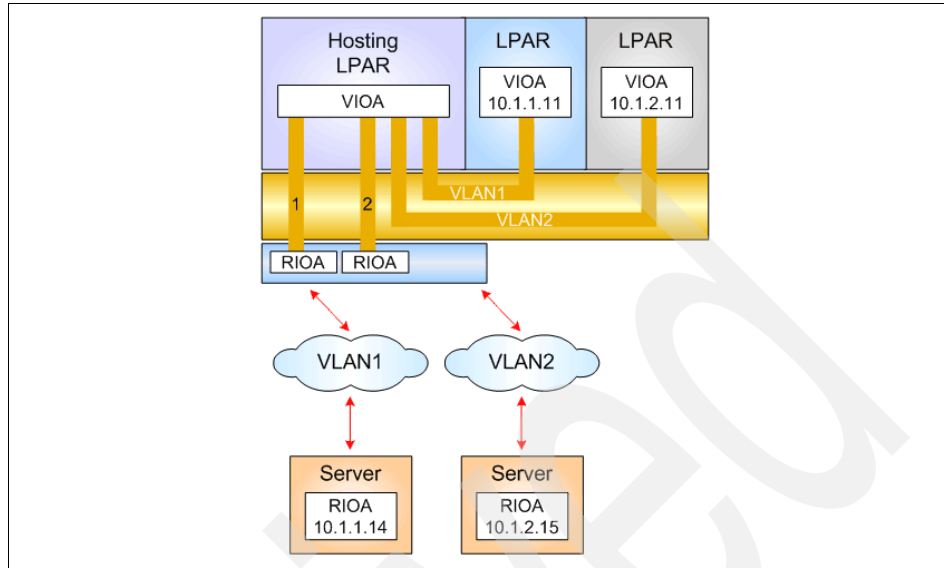


Figure 4-17 Multiple SEA configuration

Using Link Aggregation to external networks

Link aggregation is network port aggregation technology that allows several Ethernet adapters to be aggregated together to form a single pseudo Ethernet device. You can use this technology to overcome the bandwidth limitation of a single network adapter and to avoid bottlenecks when sharing one network adapter among many client partitions.

For example, ent0 and ent1 can be aggregated to ent3. Interface ent3 would then be configured with an IP address. The system considers these aggregated adapters as one adapter. Therefore, IP is configured as on any other Ethernet adapter. In addition, all adapters in the link aggregation are given the same MAC address, so that they are treated by remote systems as though they were one adapter. The main benefit of link aggregation is that they have the network bandwidth of all of their adapters in a single network presence. If an adapter fails, the packets are sent automatically on the next available adapter without disruption to existing user connections. The adapter is returned automatically to service on the link aggregation when it recovers.

You can use EtherChannel (EC) or IEEE 802.3ad Link Aggregation (LA) to aggregate network adapters. While EC is an AIX specific implementation of adapter aggregation, LA follows the IEEE 802.3ad standard. Table 4-4 on page 109 shows the main differences between EC and LA.

Table 4-4 Main differences between EC and LA aggregation

EtherChannel	IEEE 802.3ad link aggregation
Requires switch configuration	Little, if any, configuration of switch required to form aggregation. Some initial setup of the switch may be required.
Supports different packet distribution modes	Supports only standard distribution mode

Using LA, if the switch supports the Link Aggregation Control Protocol, then no special configuration of the switch ports is required. EC supports different packet distribution modes making it possible to influence the load balancing of the aggregated adapters. The remainder of this document uses the term *Link Aggregation* because it is a more universally understood term.

Note: Only outgoing packets are subject to the following discussion. Incoming packets are distributed by the Ethernet switch.

Standard distribution mode selects the adapter for the outgoing packets by algorithm. The adapter selection algorithm uses the last byte of the destination IP address (for TCP/IP traffic) or MAC address (for ARP and other non-IP traffic). Therefore, all packets to a specific IP-address will always go through the same adapter. There are other adapter selection algorithms based on source, destination, or a combination of source and destination ports that are available. EC provides one further distribution mode called *round robin*. This mode rotates through the adapters, giving each adapter one packet before repeating. The packets may be sent out in a slightly different order than they were given to the EC. This method makes the best use of its bandwidth, but consider that it also introduces the potential for out-of-order packets at the receiving system. This risk is particularly high when there are few, long-lived, streaming TCP connections. When there are many such connections between a host pair, packets from different connections could be intermingled, thereby decreasing the chance of packets for the same connection arriving out-of-order.

To avoid the loss of network connectivity by switch failure, EC and LA can provide a backup adapter. The backup adapter should be connected to a different switch than the adapter of the aggregation. So, in case of switch failure the traffic can be moved with no disruption of user connections to the backup adapter.

Figure 4-18 on page 110 shows the aggregation of three plus one adapters to a single pseudo Ethernet device including a backup feature.

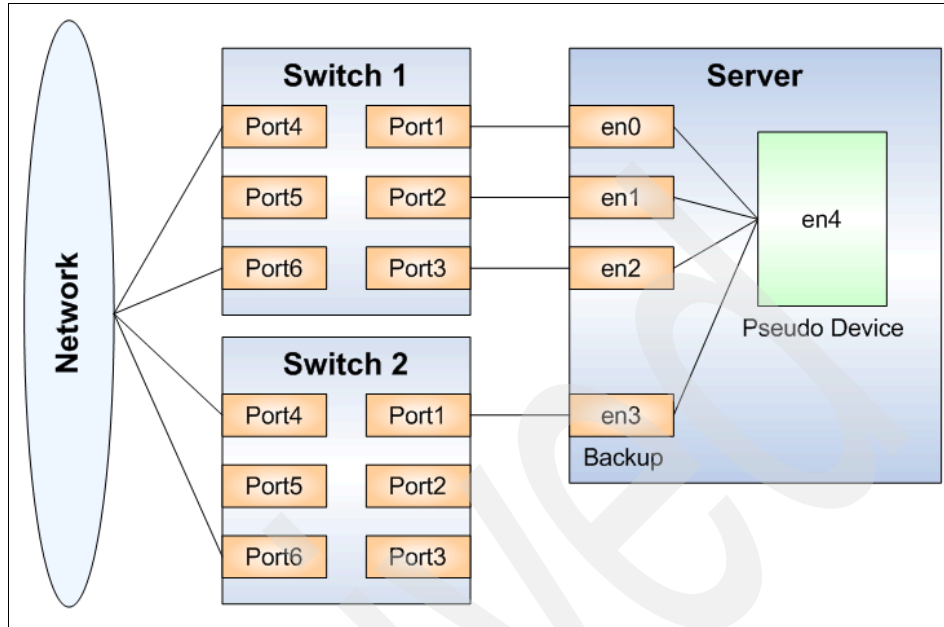


Figure 4-18 Link Aggregation (EtherChannel) pseudo device

Limitations and considerations

Consider the following limitations when implementing SEAs in the Virtual I/O Server:

- ▶ Because SEA depends on virtual Ethernet which uses the system processors for all communications functions, an increased amount of system processor load can be generated by the use of Virtual Ethernet and SEA.
- ▶ One of the virtual adapters in the SEA on the Virtual I/O Server must be defined as default adapter with a default PVID. This virtual adapter is designated as the PVID adapter and Ethernet frames without any VLAN ID tags are assigned the default PVID and directed to this adapter.
- ▶ Up to 16 virtual Ethernet adapters with 21 VLANs (20 VID and 1 PVID) on each can be shared on a single physical network adapter. There is no limit on the number of partitions that can attach to a VLAN. So, the theoretical limit is very high. In practice, the amount of network traffic limits the number of clients that can be served through a single adapter.

For performance information please refer to *Advanced POWER Virtualization on IBM® server p5 Servers Architecture and Performance Considerations*, SG24-5768.

4.3.2 Virtual SCSI

This section discusses Virtual SCSI technology.

Introduction to Virtual SCSI technology

Virtual SCSI requires POWER5 hardware with the Advanced POWER Virtualization feature activated. It provides Virtual SCSI support for AIX 5L Version 5.3 and Linux.

The driving forces behind Virtual SCSI are:

- ▶ The advanced technological capabilities of today's hardware and operating systems such as POWER5 and IBM AIX 5L Version 5.3.
- ▶ The value proposition enabling on demand computing and server consolidation. Virtual SCSI also provides a more economic I/O model by using physical resources more efficiently through sharing.

At the time of the writing of this book, the virtualization features of the POWER5 platform support up to 254 partitions while the server hardware only provides up to 240 I/O slots per machine. Each partition typically requiring one I/O slot for disk attachment and another one for network attachment puts a constraint on the number of partitions. To overcome these physical limitations, I/O resources have to be shared. Virtual SCSI provides the means to do this for SCSI storage devices.

Furthermore Virtual SCSI allows attachment of previously unsupported storage solutions. As long as the Virtual I/O Server supports the attachment of a storage resource, any client partition can access this storage by using Virtual SCSI adapters. For example, you can run Linux in a logical partition of a POWER5 server to provide support for EMC storage devices on Linux.

A Linux client partition can access the EMC storage through a Virtual SCSI adapter. Requests from the virtual adapters are mapped to the physical resources in the Virtual I/O Server. Therefore, driver support for the physical resources is needed only in the Virtual I/O Server.

Note: This publication refers to different terms for the various components that are involved with Virtual SCSI. Depending on the context, these terms can vary. With SCSI, usually the terms *initiator* and *target* are used. So, you might see terms such as *virtual SCSI initiator* and *virtual SCSI target*. On the HMC, the terms *virtual SCSI server adapter* and *virtual SCSI client adapter* are used. Basically, these terms refer to the same thing. When describing the client and server relationship between the partitions involved in Virtual SCSI, the terms *hosting partition* (meaning the Virtual I/O Server) and *hosted partition* (meaning the client partition) are used.

The terms *Virtual I/O Server partition* and *Virtual I/O Server* both refer to the *Virtual I/O Server*. The terms are used interchangeably in this section.

Partition access to virtual SCSI devices

Virtual SCSI is based on a client and server relationship. The Virtual I/O Server owns the physical resources and acts as a server or, in SCSI terms, a target device. The logical partitions access the virtual SCSI resources that are provided by the Virtual I/O Server as clients.

The virtual I/O adapters are configured using an HMC. The provisioning of virtual disk resources is provided by the Virtual I/O Server. Often the Virtual I/O Server is also referred to as *hosting partition* and the client partitions, as *hosted partitions*.

Physical disks owned by the Virtual I/O Server can either be exported and assigned to a client partition as whole or can be partitioned into several logical volumes. The logical volumes can then be assigned to different partitions. Therefore, Virtual SCSI enables sharing of adapters as well as disk devices.

To make a physical or a logical volume available to a client partition it is assigned to a Virtual SCSI server adapter in the Virtual I/O Server.

The client partition accesses its assigned disks through a Virtual SCSI Client Adapter. The Virtual SCSI Client Adapter sees standard SCSI devices and LUNs through this virtual adapter. The commands in the following example show how the disks appear on an AIX client partition:

```
# lsdev -Cc disk -s vscsi
hdisk2 Available Virtual SCSI Disk Drive

# lscfg -vpl hdisk2
hdisk2 111.520.10DDEDC-V3-C5-T1-L810000000000 Virtual SCSI Disk Drive
```

In Figure 4-19, one physical disk is partitioned into two logical volumes inside the Virtual I/O Server. Each of the two client partitions is assigned one logical volume which it accesses through a virtual I/O adapter (Virtual SCSI Client Adapter). Inside the partition the disk is seen as a normal hdisk.

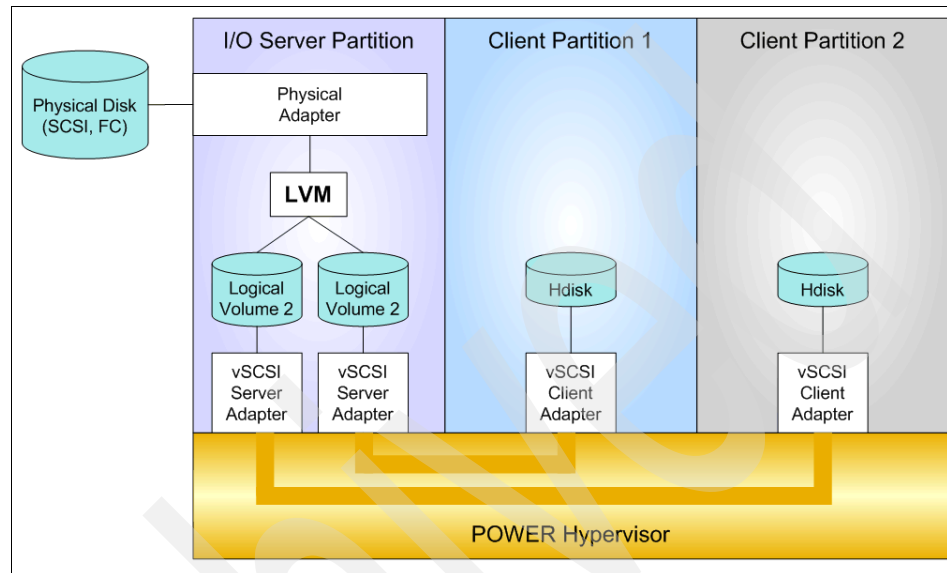


Figure 4-19 Virtual SCSI architecture overview

SCSI Remote Direct Memory Access

The SCSI family of standards provides many different transport protocols that define the rules for exchanging information between SCSI initiators and targets. Virtual SCSI uses the SCSI RDMA Protocol, which defines the rules for exchanging SCSI information in an environment where the SCSI initiators and targets have the ability to directly transfer information between their respective address spaces.

SCSI requests and responses are sent using the Virtual SCSI adapters that communicate through the hypervisor. The actual data transfer, however, is done directly between a data buffer in the client partition and the physical adapter in the Virtual I/O Server by using the Logical Remote Direct Memory Access (LRDMA) protocol.

Figure 4-20 shows how the data transfer using LRDMA appears.

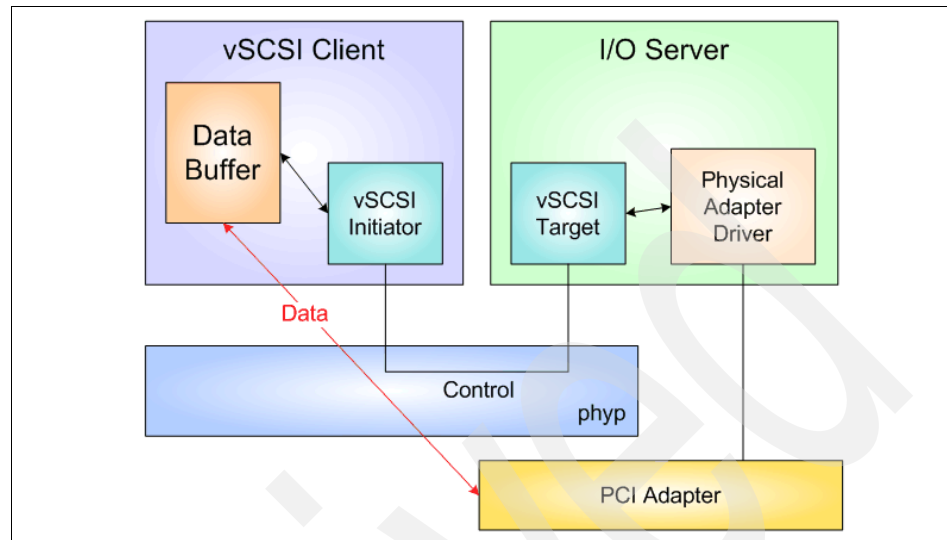


Figure 4-20 Logical Remote Direct Memory Access

AIX device configuration for virtual SCSI

The virtual I/O adapters are connected to a virtual host bridge that AIX treats much like a PCI host bridge. It is represented in the object data model as a bus device whose parent is sysplanar0. The virtual I/O adapters are represented as adapter devices with the virtual host bridge as their parent.

On the Virtual I/O Server, each logical volume or physical volume that is exported to a client partition is represented by a virtual target device that is a child of a Virtual SCSI server adapter. On the client partition, the exported disks are visible as normal hdisks. However, they are defined in subclass vscsi. They have a virtual SCSI client adapter as parent.

Figure 4-21 and Figure 4-22 on page 116 show the relationship of the devices used by AIX for Virtual SCSI and their physical counterparts.

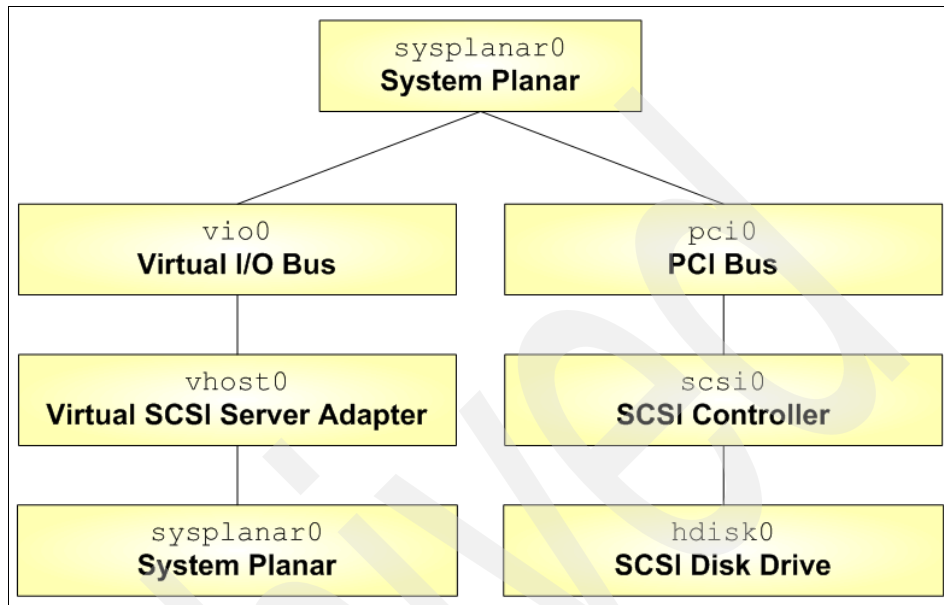


Figure 4-21 Virtual SCSI device relationship on Virtual I/O Server

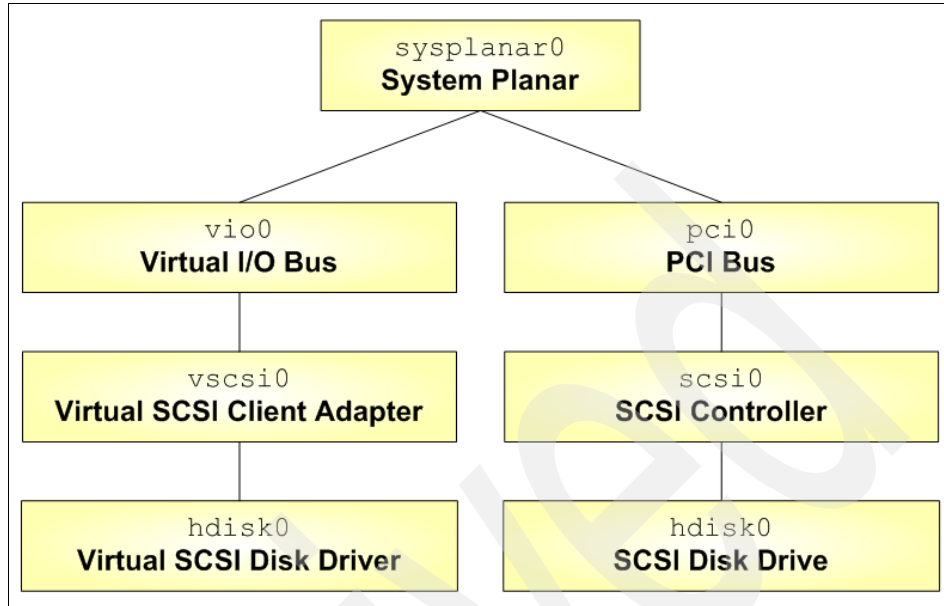


Figure 4-22 Virtual SCSI device relationship on AIX client partition

Dynamic partitioning for virtual SCSI devices

Virtual SCSI resources can be assigned and removed dynamically. On the HMC, Virtual SCSI target and server adapters can be assigned and removed from a partition using dynamic logical partitioning. The mapping between physical and virtual resources on the Virtual I/O Server can also be done dynamically.

4.3.3 Limitations and considerations

You should consider the following areas when implementing Virtual SCSI:

- ▶ At the time of the writing of this book, virtual SCSI supports Fibre Channel, parallel SCSI, and SCSI RAID devices. Other protocols such as SSA or tape and CD-ROM devices are not supported.
- ▶ Virtual SCSI itself does not have any limitations in terms of the number of supported devices or adapters. At the time of writing, the Virtual I/O Server supports a maximum of 1024 virtual I/O slots on an IBM *@server* p5 server. A maximum of 256 virtual I/O slots can be assigned to a single partition.
- ▶ Every I/O slot needs some resources to be instantiated. Therefore, the size of the Virtual I/O Server puts a limit to the number of virtual adapters that can be configured. The SCSI protocol defines mandatory and optional commands. While Virtual SCSI supports all the mandatory commands, not all optional commands are supported.

- ▶ There are performance implications when using Virtual SCSI devices. It is important to understand that associated with hypervisor calls, Virtual SCSI uses additional processor cycles when processing I/O requests. Thus, when putting heavy I/O load on Virtual SCSI devices, you use more processor cycles. Provided that there is sufficient processing capacity available, the performance of Virtual SCSI should be comparable to dedicated I/O devices.
- ▶ Suitable applications for Virtual SCSI include boot disks for the operating system or Web servers which will typically cache a lot of data. When designing a virtual I/O configuration, performance is an important aspect which should be given careful consideration.

For a more in depth discussion of performance issues see *Advanced POWER Virtualization on IBM @server p5 Servers Architecture and Performance Considerations*, SG24-5768.

4.4 Virtual I/O Server and virtualization configuration

This section provides the following information about the configuration and operating environment of the Virtual I/O Server:

- ▶ Command line interface
- ▶ Hardware resources managed
- ▶ Virtual I/O Server software installation
- ▶ Basic configuration
- ▶ Ethernet adapter sharing
- ▶ Virtual SCSI disk
- ▶ Defining the Virtual SCSI Server Adapter on the HMC
- ▶ Defining the Virtual SCSI Client Adapter on the HMC
- ▶ Creating the virtual target device on the Virtual I/O Server
- ▶ Limitations and considerations

4.4.1 Using the command line interface

The Virtual I/O Server provides a restricted scriptable command line interface. All aspects of Virtual I/O server administration are accomplished through the command line interface, including:

- ▶ Device management (physical, virtual, logical volume manager)
- ▶ Network configuration
- ▶ Software installation and update
- ▶ Security
- ▶ User management
- ▶ Installation of OEM software
- ▶ Maintenance tasks

For the initial log on to the Virtual I/O Server, use the user ID padmin, which is the prime administrator. When logging in, you are prompted for a new password, so there is no default password to remember.

Upon logging into the I/O server, you are placed in a restricted Korn shell. The restricted Korn shell works the same way as a regular Korn shell except users cannot:

- ▶ Change the current working directory
- ▶ Set the value of the SHELL, ENV, or PATH variables
- ▶ Specify the path name of the command that contains a redirect output of a command with a '>', '>|', '<>' or '>>'

As a result of these restrictions, you are not able to execute commands that are not accessible to your PATH. In addition, these restrictions prevent you from directly sending the output of the command to a file, requiring you to pipe the output to the tee command instead.

After you have logged on, you can type help to get an overview of the supported commands as shown in Example 4-1.

Example 4-1 \$help command for an overview

\$ help		
Install Commands	Physical Volume Commands	Security Commands
updateios	lspv	lsgcl
lssw	migratepv	cleargcl
ioslevel		lsfailedlogin
remote_management	Logical Volume Command	
oem_setup_env	lslv	UserID Commands
oem_platform_level	mklv	mkuser
license	extendlv	rmuser
	rmlvcopy	lsuser
LAN Commands	rmlv	passwd
mktcpip	mklvcopy	chuser
hostname		
cfglnagg		
netstat	Volume Group Commands	Maintenance Commands
entstat	lsvg	chlang
cfgnamesrv	mkvg	diagmenu
traceroute	chvg	shutdown
ping	extendvg	fsck
optimizenet	reducevg	backupios
lsnetvc	mirrorios	savevgstruct
	unmirrorios	restorevgstruct
Device Commands	activatevg	starttrace
mkvdev	deactivatevg	stoptrace

lsdev	importvg	cattracerpt
lsmapi	exportvg	bootlist
chdev	syncvg	snap
rmdev		startsysdump
cfgdev		topas
mkpath		mount
chpath		unmount
lspath		showmount
rmpath		startnetsvc
		errlog
		stopnetsvc

To receive further help with these commands, you can use the **help** command as shown in Example 4-2.

Example 4-2 \$help command for a specific command

```
$ help errlog
Usage: errlog [-ls | -rm Days]

Displays or clears the error log.

-ls      Displays information about errors in the error log file
         in a detailed format.

-rm      Deletes all entries from the error log older than the
         number of days specified by the Days parameter.
```

The Virtual I/O Server command line interface supports two execution modes:

- ▶ Traditional mode
- ▶ Interactive mode

The traditional mode is for single command execution. In this mode, you execute one command at a time from the shell prompt. For example, to list all virtual devices, enter the following:

```
#ioscli lsdev -virtual
```

To reduce the amount of typing that is required in traditional shell level mode, an alias has been created for each sub-command. With the aliases set, you are not required to type the **ioscli** command. For example, to list all devices of type adapter, you can enter the following:

```
#lsdev -type adapter
```

In interactive mode the user is presented with the **ioscli** command prompt by executing the **ioscli** command without any sub-commands or arguments. From this point on, **ioscli** commands are executed one after the other without having to retype **ioscli**. For example, to enter interactive mode, enter:

```
#ioscli
```

Once in interactive mode, to list all virtual devices, enter:

```
#lsdev -virtual
```

External commands, such as **grep** or **sed**, cannot be executed from the interactive mode command prompt. You must first exit interactive mode by entering **quit** or **exit**.

4.4.2 Managing hardware resources

The optional Advanced POWER Virtualization feature that enables Micro-Partitioning on a @server p5 server provides the Virtual I/O Server installation CD. A logical partition with enough resources to share to other partitions is required. Below is a list of minimum hardware requirements that must be available to create the Virtual I/O Server:

POWER5 server	The Virtual I/O capable machine.
HMC	To create the partition and assign resources.
Storage adapter	The server partition needs at least one storage adapter.
Physical disk	If you want to share your disk with client partitions, you need a disk that is large enough to make sufficient-sized logical volumes.
Ethernet adapter	If you want to route network traffic securely from a virtual Ethernet to a real network adapter.
Memory	At least 128 MB of memory.

Virtual I/O Server Version 1.1 is designed for selected configurations that include specific models from IBM and other storage product vendors. Consult your IBM representative or Business Partner for the latest information and included configurations.

Virtual devices that are exported to client partitions by the Virtual I/O Server must be attached through one of the following physical adapters:

- ▶ PCI 4-Channel Ultra3 SCSI RAID Adapter (FC 2498)
- ▶ PCI-X Dual Channel Ultra320 SCSI RAID Adapter (FC 5703)
- ▶ Dual Channel SCSI RAID Enablement Card (FC 5709)
- ▶ PCI-X Dual Channel Ultra320 SCSI Adapter (FC 5712)
- ▶ 2 Gigabit Fibre Channel PCI-X Adapter (FC 5716)
- ▶ 2 Gigabit Fibre Channel Adapter for 64-bit PCI Bus (FC 6228)
- ▶ 2 Gigabit Fibre Channel PCI-X Adapter (FC 6239)

We recommend that you plan carefully before you begin the configuration and installation of your I/O Server and client partitions. Depending on the type of workload and needs for an application, consider mixing virtual and physical devices. For example, if your application benefits from fast disk access, then plan a physical adapter that is dedicated to this partition.

Installation of the Virtual I/O Server partition is performed from a special mksysb CD that is provided, at an additional charge, when you order the Advanced POWER Virtualization feature. This CD contains dedicated software that is only for the Virtual I/O Server operations. So, the Virtual I/O server software is only supported in Virtual I/O Server partitions.

You can install the Virtual I/O Server from CD or using NIM on Linux from the HMC. For more information about the installation of the Virtual I/O Server, refer to 4.4.3, “Installing Virtual I/O Server” on page 121.

The Virtual I/O Server supports the following operating systems as Virtual I/O client:

- ▶ IBM AIX 5L Version 5.3
- ▶ SUSE LINUX Enterprise Server 9 for POWER
- ▶ Red Hat Enterprise Linux AS for POWER Version 3

4.4.3 Installing Virtual I/O Server

This section describes how to install Virtual I/O Server to the partition called `I/O_Server_1`. These instructions assume that you are familiar with a basic AIX installation.

Figure 4-23 shows an example of an HMC-based Virtual I/O server creation.

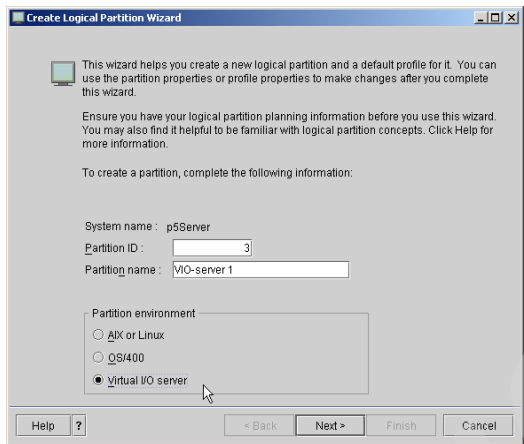


Figure 4-23 Hardware Management Console - create Virtual I/O server

To install Virtual I/O Server, follow these steps:

1. Activate the I/O_Server_1 partition by right-clicking the partition name and selecting **Activate**, as shown in Figure 4-24.

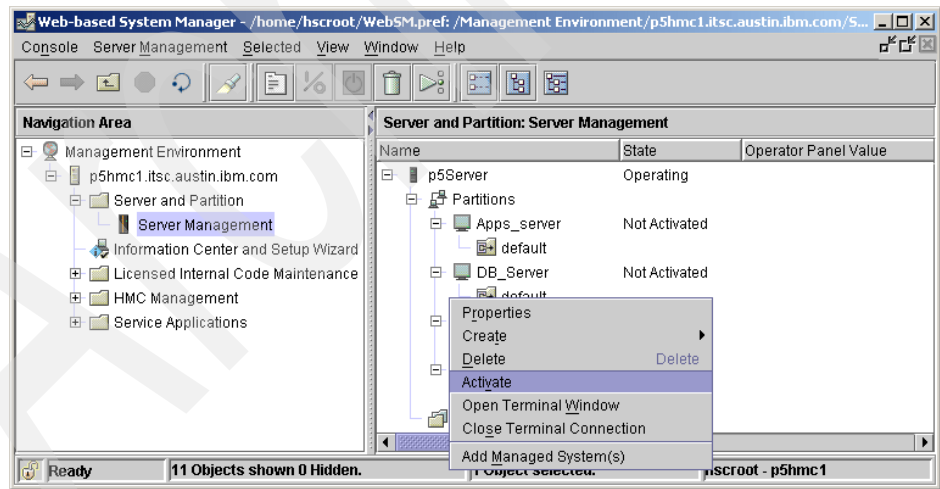


Figure 4-24 Activate I/O_Server_1 partition

2. Select the default profile that you used to create this server. Select **Open a terminal window or console session**, and click **(Advanced...)**, as shown in Figure 4-25.

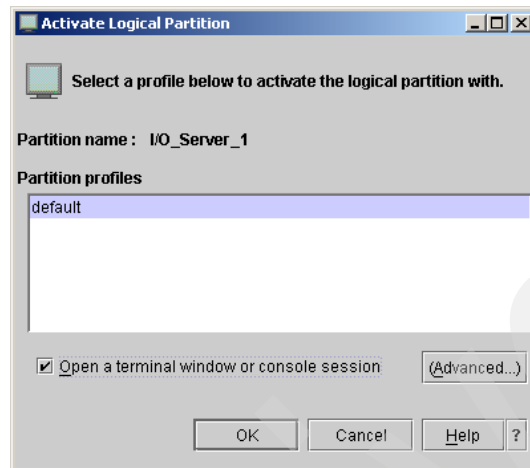


Figure 4-25 Selecting the profile

3. Choose **SMS boot mode** as shown in Figure 4-26. Click **OK** in this window to return to the previous window. When at the previous window, click **OK** to activate the partition and to launch a terminal window.

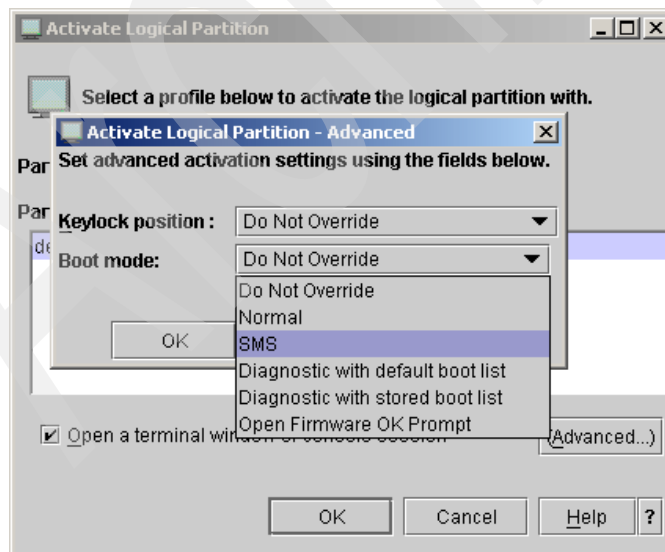


Figure 4-26 Choosing SMS boot mode

4. Figure 4-27 shows a pSeries SMS menu. Proceed with the installation as you would for an AIX installation, and choose **CD** as the installation device.

```
PowerPC Firmware
Version SF220_007
SMS 1.5 (c) Copyright IBM Corp. 2000,2003 All rights reserved.
-----
Main Menu
  1. Select Language
  2. Setup Remote IPL (Initial Program Load)
  3. Change SCSI Settings
  4. Select Console
  5. Select Boot Options

-----

Navigation Keys:

                                     X = eXit System Management Services
-----
Type the number of the menu item and press Enter or select Navigation Key:5_
MA*  a                                                                    p1 25/076
```

Figure 4-27 SMS menu

5. When the installation procedure has finished, use `padmin` for the username at the login prompt. Choose a new password, and accept the license using the `license` command at the prompt (`$`), as shown in Figure 4-28. You can use the `lspv` command to show the available disks.

```
0513-059 The aixmibd Subsystem has been started. Subsystem PID is 655468.
0513-059 The muxatmd Subsystem has been started. Subsystem PID is 659536.
Finished starting tcpip daemons.
Starting NFS services:
0513-059 The biod Subsystem has been started. Subsystem PID is 622644.
0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 663688.
Completed NFS services.
Virtual I/O Server
login: 0513-059 The ctrmc Subsystem has been started. Subsystem PID is 356530.

Virtual I/O Server
login: padmin
[compat]: 3004-610 You are required to change your password.
      Please choose a new one.

padmin's New password:
Enter the new password again:

$ license -accept
$ lspv
hdisk0      00cddedc00efe72a      rootvg      active
hdisk1      00cddedc15d81aaf      None
hdisk2      00cddedc003416f4      None
hdisk3      00cddedc04a9aa04      None
$
MA*  a
```

p1 25/003

Figure 4-28 Finished Virtual I/O Server installation

The `I/O_Server_1` partition is now ready for the further configurations.

4.4.4 Basic configuration

The Virtual I/O Server provides the Virtual SCSI and SEA Virtual I/O to client partitions. You accomplish this configuration by assigning physical devices to the Virtual I/O Server partition, and then by configuring virtual adapters on the clients to allow communication between the client and the Virtual I/O Server.

Using virtual I/O devices:

- ▶ Facilitates the sharing of physical resources between partitions on a POWER5 system
- ▶ Provides Virtual SCSI and SEA function to client partitions
- ▶ Enables the creation of partitions without requiring additional physical I/O resources
- ▶ Allows the creation of more partitions than I/O slots or physical devices with the ability for partitions to have dedicated I/O, virtual I/O, or both
- ▶ Maximizes the utilization of physical resources on a POWER5 system

4.4.5 Ethernet adapter sharing

SEA enables the client partitions to communicate with other systems outside the Central Electronic Complex without requiring physical Ethernet adapters in the partitions. This communication is accomplished by sharing the physical Ethernet adapters in the Virtual I/O Server partition.

VLANs that are bridged outside using a SEA, require a Virtual Ethernet adapter to have the trunk adapter setting on. This Virtual Ethernet adapter is assigned to the Virtual I/O Server partition using the HMC. The SEA setup commands are then run on the Virtual I/O Server to create associations between the physical and virtual adapters.

To configure a trunk Virtual Ethernet adapter on the HMC:

1. Right-click the partition profile of the Virtual I/O Server partition, and open the properties of the profile.
2. Choose the Virtual I/O tab. The HMC panel to create a virtual adapter appears, as shown in Figure 4-29 on page 127.

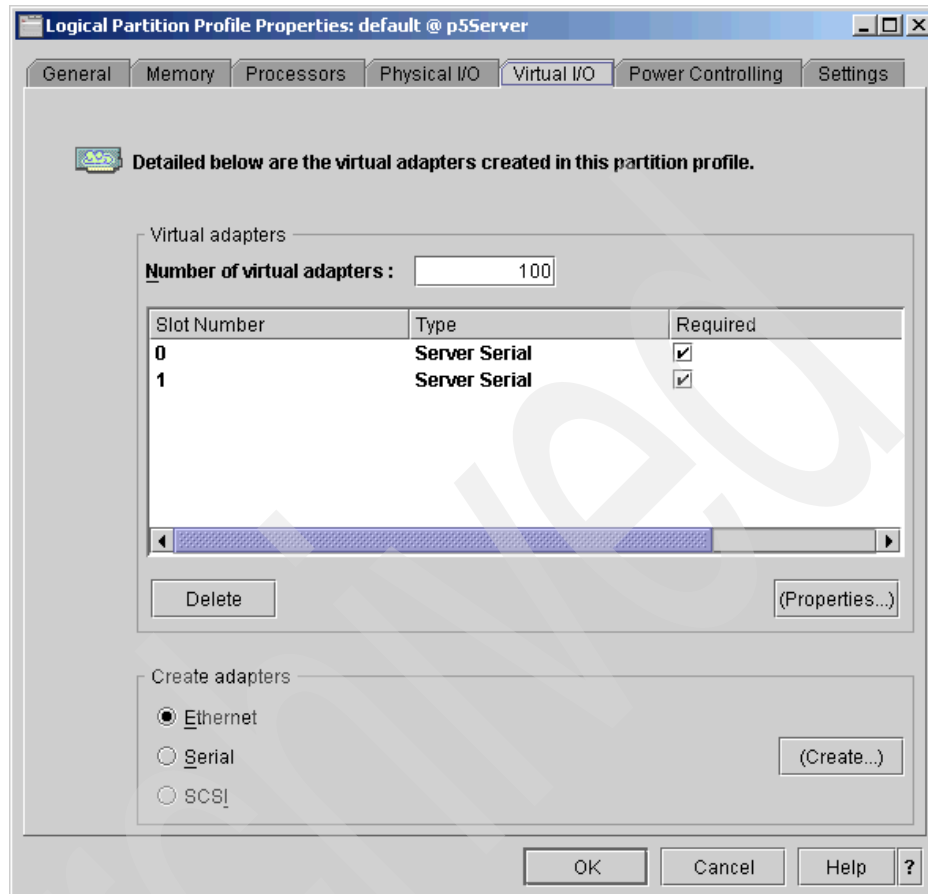


Figure 4-29 Creating the trunk Virtual I/O Server

3. Make sure that the value in the Number of virtual adapters field is higher than the highest used Slot Number. To allow additional dynamically configured adapters, choose a number where you can add more virtual adapters later. This value is similar to the maximum value of the processor and memory definition. To change it, you have to shutdown and reactivate the partition.
4. Select **Ethernet** in the Create Adapters panel, and then click **(Create...)**. The Virtual Ethernet Adapter Properties dialog box appears, as shown in Figure 4-29.

The slot number that is used for this adapter identifies the virtual adapter within the logical partition. The combination of the slot number and the logical partition LAN ID uniquely identifies this slot within the managed system.

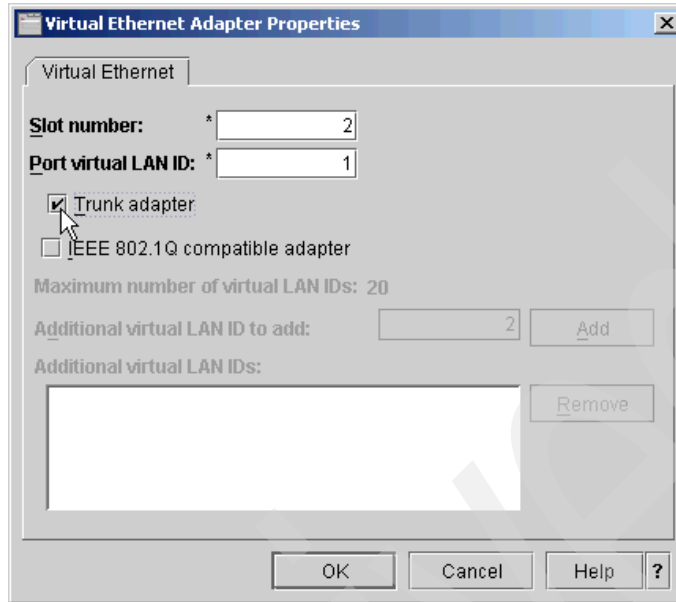


Figure 4-30 Virtual Ethernet Adapter Properties panel

Each Virtual Ethernet adapter has assigned a Port virtual LAN ID (PVID) or a virtual LAN ID (VID) number. Selecting the **IEEE 802.1Q compatible adapter** option allows you to configure additional virtual LAN IDs. This allows the Virtual Ethernet adapter to be part of additional VLANs as specified in the IEEE 802.1Q network standard. Virtual Ethernet adapters can communicate with each other only if they are assigned to the same PVID or VID number.

Important: Trunk adapter must be selected on each Virtual Ethernet adapter that will be mapped to create an SEA.

5. After you define a virtual adapter in a partition profile, you must shutdown and reactivate the partition to make the adapter available. If a network connection is already configured and your RMC connection is working correctly, you can add the Virtual Ethernet adapter dynamically to the partition.

6. Right-click the partition name, and choose **Dynamic Logical Partitioning** → **Virtual Adapter Resources** → **Add / Remove** as shown in Figure 4-31.

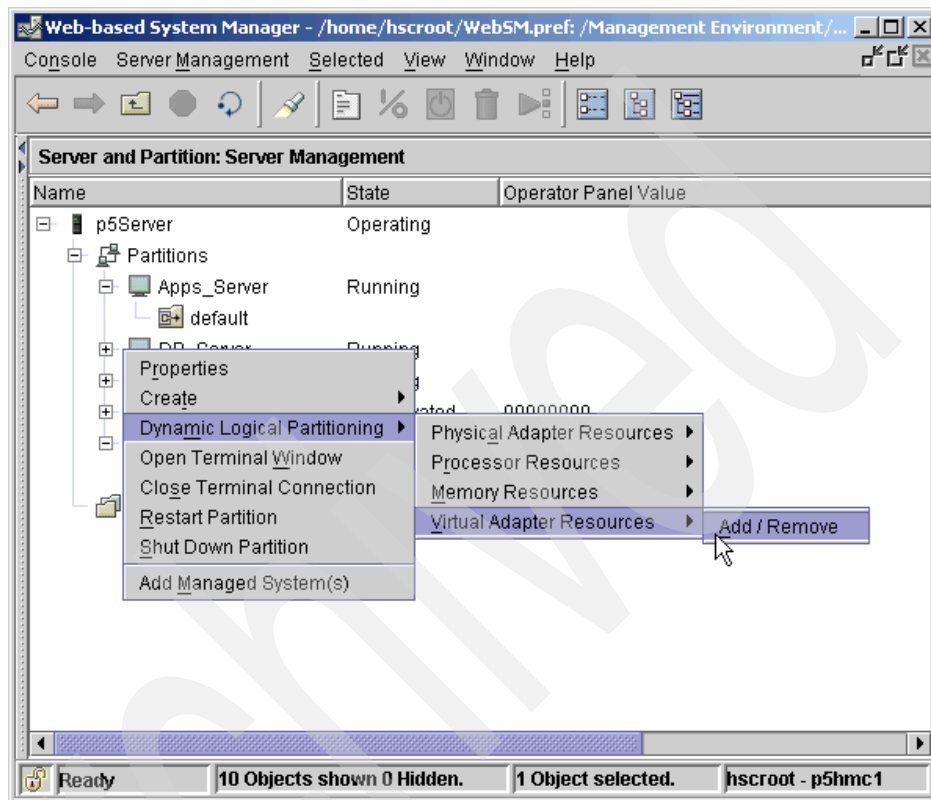


Figure 4-31 Dynamically adding or removing virtual adapters to a partition

Important: If you want to keep your dynamic virtual adapter changes after you reactivate the partition, you also have to add or remove the defined adapters to the partition profile.

7. Run the **cfgdev** command from the command line interface of the Virtual I/O Server to refresh the configuration from the operating system point of view.
8. Define the SEA on the Virtual I/O Server using the **mkvdev** command as follows:

```
mkvdev -sea TargetDevice -vadapter VirtualEthernetAdapter ...
      -default DefaultVirtualEthernetAdapter
      -defaultid SEADefaultPVID [-attr Attributes=Value ...]
```

Using the example in Figure 4-32, the target devices are the physical adapters (for example, ent0 and ent1). The virtual devices are ent2, ent3, and ent4, and the defaultid is the default PVID associated with the default virtual Ethernet adapter.

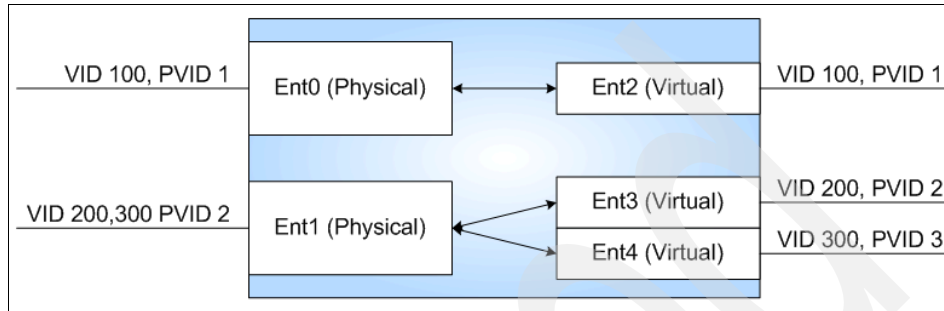


Figure 4-32 Example of an I/O server partition bridge

Important: To set up the SEA, all involved virtual and physical Ethernet interfaces have to be unconfigured (down or detached).

9. Setup SEA using the following commands:

```
$mkvdev -sea ent0 -vadapter ent2 -default ent2 -defaultid 1
$mkvdev -sea ent1 -vadapter ent3 ent4 -default ent3 -defaultid 2
```

After running the **mkvdev** command, the system creates the SEA ent5.

In the second example, the physical Ethernet adapter is ent1. The **mkvdev** command maps the virtual Ethernet adapter ent3 and ent4 to the physical adapter. Additionally, ent3 is defined as a default adapter with the default VLAN ID of 2. Untagged packets that are received by the SEA are tagged with the VLAN 2 ID and are sent to the Virtual Ethernet adapter ent3.

10. Configure the ent5 interface with an IP address using the **mktcpip** command as shown in the following:

```
mktcpip -hostname HostName -inetaddr Address -interface Interface
[-start] [-netmask SubnetMask] [-cabletype CableType]
[-gateway Gateway] [-nsrvaddr NameServerAddress]
[-nsrvdomain Domain]]
```

11. Set up the hostname and IP address for the SEA as shown in the following example.

```
$ mktcpip -hostname p5_2ioserver1 -inetaddr 9.3.5.150 -interface en5
-netmask 255.255.255.0 -gateway 9.3.5.41
```

Restriction: Multiple subnets may connect externally using the same Ethernet adapter; however, each subnet must be tagged with a different VLAN ID.

4.4.6 Virtual SCSI disk

Virtual SCSI facilitates the sharing of physical disk resources (I/O adapters and devices) between logical partitions. Virtual SCSI enables partitions to access SCSI disk devices without requiring physical resources be allocated to the partition. Partitions maintain a client/server relationship in the Virtual SCSI environment. Partitions that contain Virtual SCSI devices are referred to as client partitions while the partition that owns the physical resources (adapters, devices) is the Virtual I/O Server.

The Virtual SCSI disks are defined as logical volumes or as physical volumes in the Virtual I/O Server. All standard conventional rules apply to the logical volumes. The logical volumes appear as real devices (hdisks) in the client partitions and can be used as a boot device and as a NIM target.

After a virtual disk is assigned to a client partition, the Virtual I/O Server must be available before the client partitions are able to boot.

Defining volume groups and logical volumes

If you want to create a logical volume to assign to your client partition, use the **mk1v** command. To create the logical volume on a separate disk, you first have to create a volume group and assign one or more disks using the **mkvg** command.

The basic syntax of the **mkvg** command to create a volume group on the Virtual I/O Server is:

```
mkvg [-f] [-vg VolumeGroup] PhysicalVolume ...
```

The basic syntax of the **mk1v** command to create a logical volume on the Virtual I/O Server is:

```
mk1v [-mirror] [-lv NewLogicalVolume | -prefix Prefix]  
      VolumeGroup Size [PhysicalVolume ...]
```

To create a volume group and define the logical volume:

1. Create a volume group and assign a disk to this volume group using the **mkvg** command as shown. In this example the name of the volume group is **rootvg_clients**.

```
$ mkvg -f -vg rootvg_clients hdisk2  
rootvg_clients
```

2. Define the logical volume which will be visible as a disk to the client partition. The size of this logical volumes acts as the size of disks which will be available to the client partition. Use the **mk1v** command to create 2 GB size logical volume called **rootvg_dbsrv** as follows:

```
$ mk1v -lv rootvg_dbsrv rootvg_clients 2G
rootvg_dbsrv
```

4.4.7 Defining the Virtual SCSI Server adapter on the HMC

To define the Virtual SCSI Server adapter on the HMC:

1. On the Virtual I/O Server partition profile select the Virtual I/O tab to create a Virtual SCSI Server adapter. Choose **SCSI** and click **(Create...)** to proceed. The Virtual SCSI Adapter Properties dialog box opens, as shown in Figure 4-33.

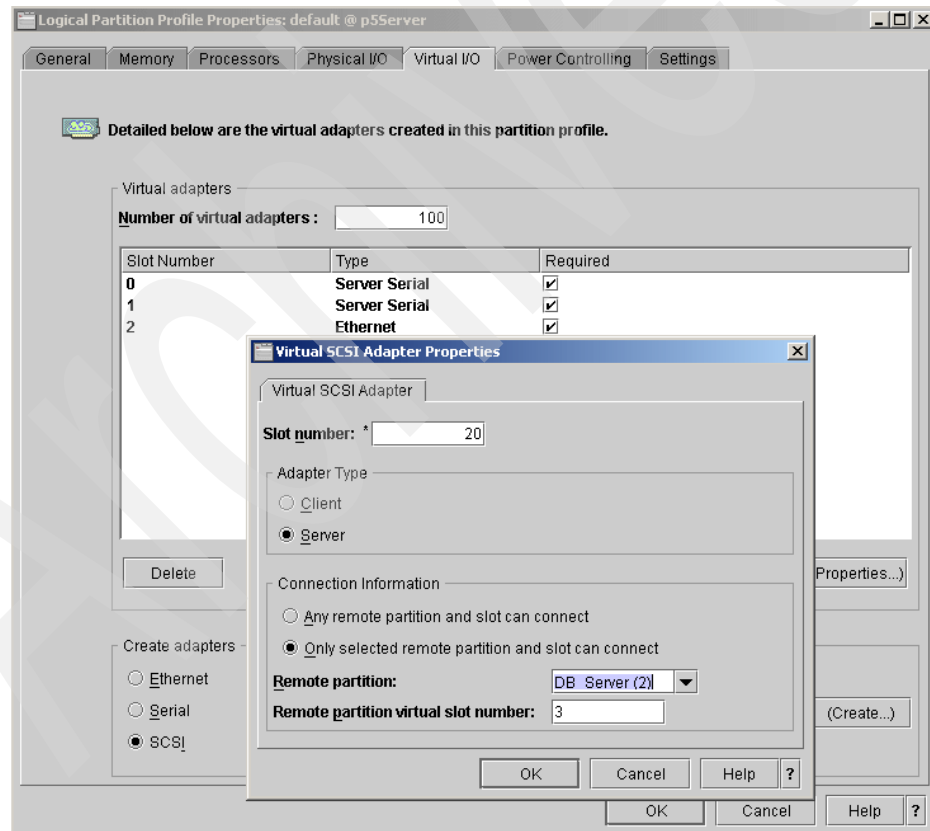


Figure 4-33 Virtual SCSI Adapter Properties panel on the IO Server

The slot number identifies the virtual adapter within the logical partition. The combination of the slot number and the logical partition ID uniquely identify this slot within the managed system.

This slot number does not refer to any physical hardware location on your system. You can, therefore, assign slot numbers to virtual adapters in any way that makes sense to you, provided that you follow these guidelines:

- You can use any slot number from 2 up to (but not including) the maximum number of virtual adapters. Slots 0 and 1 are reserved for system-created virtual adapters. By default, the system displays the lowest unused slot number for this logical partition.
- You cannot use a slot number that was used for any other virtual adapter on the same logical partition.

2. Select **Server** as the Adapter Type.
3. In the Connection Information area, define whether or not you want to allow all partitions or only one dedicated partition to connect to this SCSI drive.
4. If the client partition is not defined yet, you can enter a unique Partition ID into the Remote Partition field. Use this Partition ID when defining your client partition. The HMC will assign this partition automatically as a client for virtual SCSI resources. The Remote partition virtual slot number has to match with the slot number defined for your client SCSI adapter on the client partition.
5. Click **OK** and the Virtual SCSI Server adapter is ready to be configured from the command line interface of the Virtual I/O Server. When you define the adapter in the partition profile you have to shutdown your partition and reactivate it to make the adapter available or to use the dynamically reconfiguration process.

4.4.8 Defining the Virtual SCSI Client adapter on the HMC

The Virtual SCSI Client adapter is defined in the same panel in the client partition profile. Figure 4-34 shows the client partition DB_Server definition for the Virtual SCSI Client adapter.

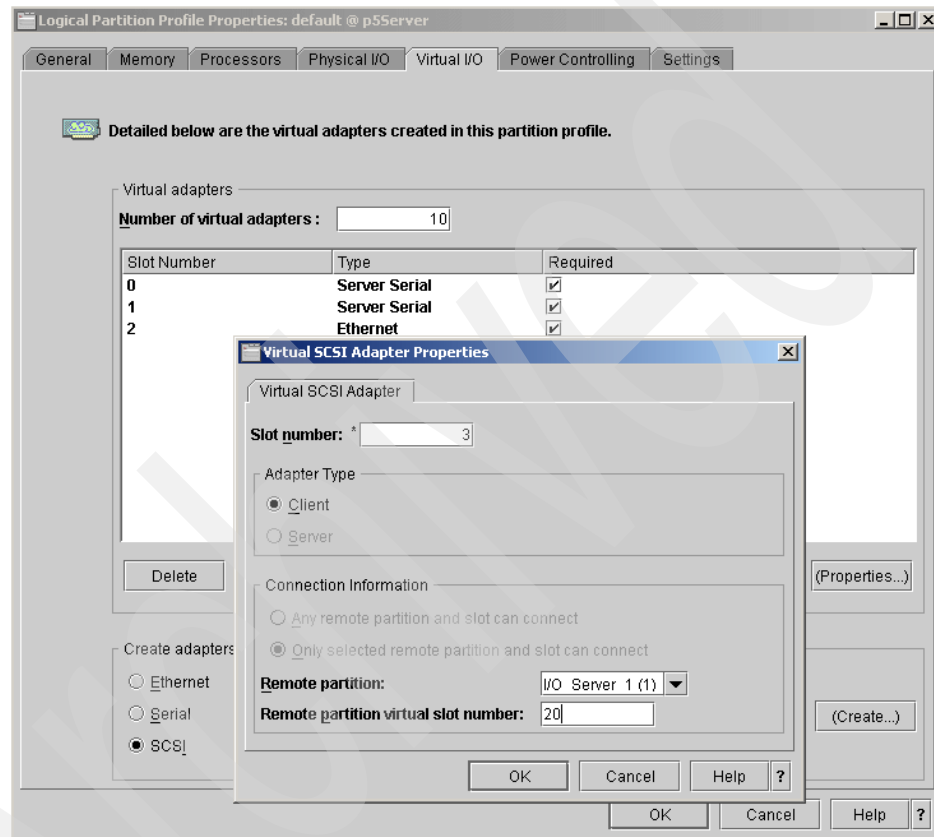


Figure 4-34 Virtual SCSI Adapter Properties panel on the client partition

The Virtual SCSI Client adapter has Slot number 3 defined, which matches to the Remote partition virtual slot number on the Virtual I/O Server partition.

The Adapter Type assigned on the client partition is Client.

In the Connection Information area, select the hosting I/O Server partition and fill in the Remote partition virtual slot number. In this example, this is slot number 20.

4.4.9 Creating the virtual target device on the Virtual I/O Server

The basic command to map the Virtual SCSI with the logical volume or hdisk is as follows:

```
mkvdev -vdev TargetDevice -vadapter VirtualSCSIServerAdapter  
[-dev DeviceName]
```

To create the virtual target device:

1. Run the **lsdev -virtual** command to make sure that the new virtual SCSI adapter is available:

```
$ lsdev -virtual  
name          status      description  
  
ent2          Available  Virtual I/O Ethernet Adapter (1-lan)  
vhost0       Available Virtual SCSI Server Adapter  
vhost1        Available  Virtual SCSI Server Adapter  
vhost2        Available  Virtual SCSI Server Adapter  
vsa0          Available  LPAR Virtual Serial Adapter
```

2. Create a virtual target device, which maps the Virtual SCSI Server adapter vhost0 to the logical volume rootvg_dbsrv that was created previously. When you do not use the -dev flag, the default name of the Virtual Target Device adapter is vtscsix. Run the **mkvdev** command as shown:

```
$ mkvdev -vdev rootvg_dbsrv -vadapter vhost0 -dev vdbsrv  
vdbsrv Available
```

To map a physical volume to the Virtual SCSI Server Adapter use hdiskx instead of the logical volume devices for the -vdev flag.

The **lsdev** command shows the newly created Virtual Target Device adapter.

```
$ lsdev -virtual  
name          status      description  
  
vhost0        Available  Virtual SCSI Server Adapter  
vsa0          Available  LPAR Virtual Serial Adapter  
vdbsrv       Available Virtual Target Device - Logical Volume
```

The **lsmap** command shows us the logical connections between newly created devices, as follows:

```
$ lsmap -vadapter vhost0
SVSA          Physloc          Client
PartitionID
-----
vhost0        U9111.520.10DDEEC-V1-C20  0x00000000

VTD          vdbsrv
LUN          0x8100000000000000
Backing device rootvg_dbsrv
Physloc
```

The physical location is a combination of the slot number. In this example, 20 and the logical partition ID.

3. At this point the virtual device can be attached from the client partition. You can activate the partition with the SMS menu and install the AIX operating system on the virtual disk, or you can add an additional virtual disk using the **cfgmgr** command.

The Client PartitionID is visible as soon the client partition is active.

4.4.10 Limitations and considerations

The Virtual I/O Server software is a dedicated software only for the Virtual I/O Server operations, and there is no possibility to run other applications in the Virtual I/O Server partition.

There is no option to get the Virtual I/O Server partition pre-installed on new systems. At the time of the writing of this book, the preinstall manufacturing process does not allow the Virtual I/O Server partition to be pre-installed.

The Virtual I/O Server should be properly configured with enough resources. The most important are the processor resources. If a Virtual I/O Server has to host a lot of resources to other partitions, you must ensure that enough processor power is available.

Logical volume limitation

The Virtual I/O Server operating system allows you to define up to 1024 logical volumes per volume group, but the actual number that you can define depends on the total amount of physical storage that are defined for that volume group and the size of the logical volumes you configure.

Table 4-5 shows the limitations for logical storage management.

Table 4-5 Limitations for logical storage management

Category	Limit
Volume group	4096 per system
Physical volume	1024 per volume group
Physical partition	2097152 per volume group
Logical volume	4096 per volume group
Logical partition	Based on physical partitions

Dynamic logical partitioning

A dynamic logical partition (DLPAR) allows you to add and remove the resources that are associated with a partition dynamically without rebooting the partition. This functionality was first supported by AIX 5L Version 5.2.

Advanced Virtualization and Micro-Partitioning technology provided a change from the DLPAR perspective with finer granularity of system resources available for DLPAR operations. This chapter provides an update of the DLPAR functions with the introduction of the POWER5 systems and includes the following sections:

- ▶ 5.1, “Dynamic logical partitioning overview” on page 140
- ▶ 5.2, “The process flow of a DLPAR operation” on page 148
- ▶ Table 5.3 on page 152
- ▶ 5.4, “DLPAR-safe and DLPAR-aware applications” on page 155
- ▶ 5.5, “Integrating a DLPAR operation into the application” on page 157
- ▶ 5.6, “Script-based DLPAR event handling” on page 160
- ▶ 5.7, “DLPAR script subcommands” on page 167
- ▶ 5.8, “How to manage DLPAR scripts” on page 186
- ▶ 5.9, “API-based DLPAR event handling” on page 194
- ▶ 5.10, “Error handling of DLPAR operations” on page 205

5.1 Dynamic logical partitioning overview

DLPAR was introduced with AIX 5L Version 5.2. A dynamic partition based on AIX 5L Version 5.2 can consist of the following resource elements:

- ▶ A dedicated processor
- ▶ 256 MB memory region
- ▶ I/O adapter slot

Multiple resources can be placed under the exclusive control of a given logical partition. DLPAR extends these capabilities by allowing this fine-grained resource allocation to occur not only when activating a logical partition, but also while the partitions are running. Individual processors, memory regions, and I/O adapter slots can be released into a free pool, acquired from that free pool, or moved directly from one partition to another.

On POWER5 with AIX 5L Version 5.3, however, a partition can consist of dedicated processors, or virtual processors with a specific capacity entitlement running in capped or uncapped mode, dedicated memory region, and virtual or physical I/O adapter slots.

For dedicated and shared processor partitions, it is possible to:

- ▶ Add, move, or remove memory in a granularity of 16 MB regions dynamically
- ▶ Add, move, or remove physical I/O adapter slots dynamically
- ▶ Add, or remove virtual I/O adapter slots dynamically

For a dedicated processor partition, it is only possible to add, move, or remove whole processors dynamically. When you remove a processor dynamically from a dedicated partition on a system that uses shared processor partitions, it is then assigned to the shared processor pool.

For shared processor partitions, it is also possible to:

- ▶ Remove, move, or add entitled shared processor capacity dynamically
- ▶ Change between capped and uncapped processing dynamically
- ▶ Change the weight of an uncapped partition dynamically
- ▶ Add, and remove virtual processors dynamically

The DLPAR operation for a shared processor refers to the additional processor capacity which is expressed as a percentage, so 100 represents one physical processor and 180 represents the 1.8 processors.

Note: A single DLPAR operation can perform only one type of resource change. You cannot add and remove memory to and from the same partition in a single DLPAR operation. Also, you cannot move processor and memory from a partition to another partition in a single DLPAR operation.

5.1.1 Processor resources

Figure 5-1 shows the panel for dynamic reconfiguration of processor resources on the HMC. From this panel, you can choose to add, remove, or move your resources. Select the partition that you want to change dynamically, and press the right mouse button. Then choose **Dynamic Logical Partitioning** → **Processor Resources** and choose the action that you want to perform.

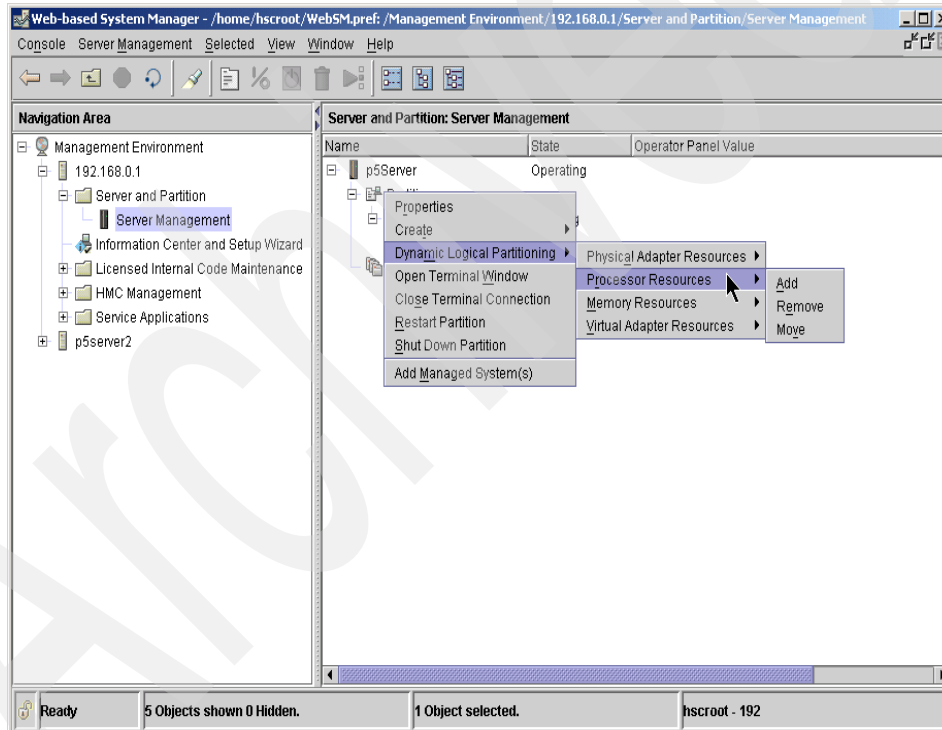


Figure 5-1 Dynamic Logical Partitioning Processor Resources

From the Add Processor Resources panel shown in Figure 5-2 on page 142, you can specify the processing units and the number of virtual processors that you want to add to the selected partition. The limits for adding processing units and virtual processors are the maximum values that are defined in the partition

profile. This panel also allows you to add variable weight when the partition runs in uncapped mode.

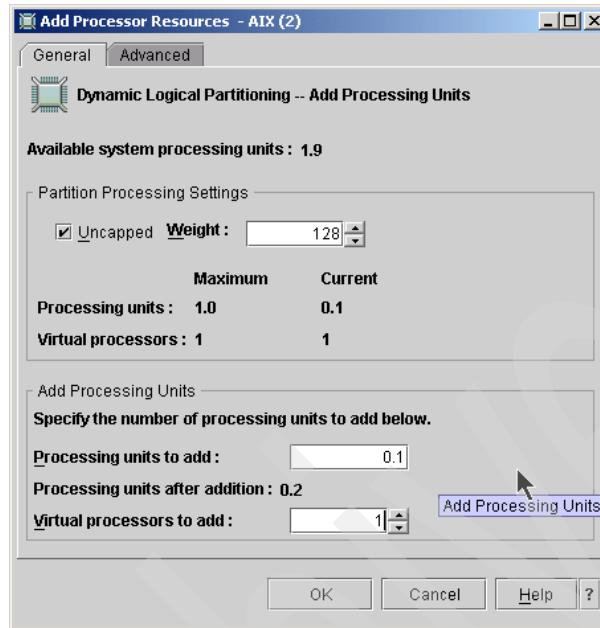


Figure 5-2 Add Processor Resources

Additionally, the Add Processor Resource panel allows you to change the partition mode dynamically from uncapped to capped or vice versa. To show the actual status of the partition, use the **lparstat -i** command from the AIX command line interface of the partition, as shown in the following example:

```
# lparstat -i
Node Name                : applsrv
Partition Name           : Apps_Server
Partition Number         : 4
Type                     : Shared-SMT
Mode                     : Uncapped
Entitled Capacity        : 0.30
Partition Group-ID       : 32772
Shared Pool ID           : 0
Online Virtual CPUs      : 2
Maximum Virtual CPUs     : 10
Minimum Virtual CPUs     : 1
Online Memory            : 512 MB
Maximum Memory           : 1024 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : 128
```


Minimum Capacity	: 0.20
Maximum Capacity	: 1.00
Capacity Increment	: 0.01
Maximum Dispatch Latency	: 16999999
Maximum Physical CPUs in system	: 2
Active Physical CPUs in system	: 2
Active CPUs in Pool	: -
Unallocated Capacity	: 0.00
Physical CPU Percentage	: 15.00%
Unallocated Weight	: 0

Figure 5-3 shows how to change the mode of the partition from uncapped to capped mode. Deselect **Uncapped** and click **OK**.

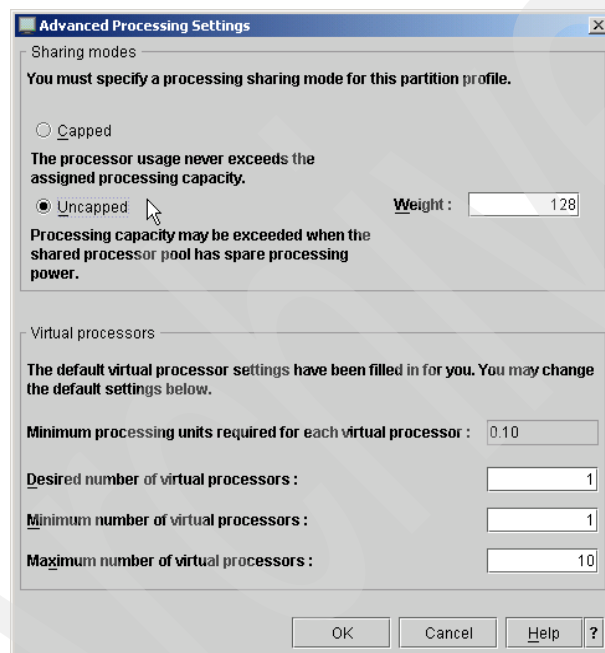


Figure 5-3 Advanced Processor Settings - Uncapped Mode

To verify this dynamic action, use the **lparstat -i** command on the selected partition again. The partition mode changed from uncapped to capped.

```
# lparstat -i
Node Name                : applsrv
Partition Name           : Apps_Server
Partition Number         : 4
Type                     : Shared-SMT
Mode                   : Capped
Entitled Capacity        : 0.30
Partition Group-ID       : 32772
Shared Pool ID           : 0
Online Virtual CPUs      : 2
Maximum Virtual CPUs     : 10
Minimum Virtual CPUs     : 1
Online Memory            : 512 MB
Maximum Memory           : 1024 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : 128
Minimum Capacity         : 0.20
Maximum Capacity         : 1.00
Capacity Increment       : 0.01
Maximum Dispatch Latency : 16999999
Maximum Physical CPUs in system : 2
Active Physical CPUs in system : 2
Active CPUs in Pool      : -
Unallocated Capacity     : 0.00
Physical CPU Percentage   : 15.00%
Unallocated Weight       : 0
```

Figure 5-4 on page 145 shows the Remove Processing Units panel that allows you to remove processing units and virtual processors dynamically. The limit for the removal of processing units and virtual processors is the minimum value defined in the partition profile.

This panel also allows you to remove variable weight when the partition runs in uncapped mode.

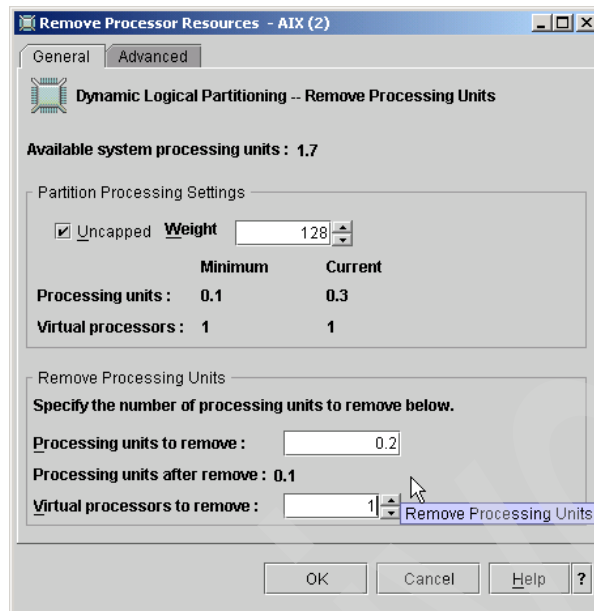


Figure 5-4 Remove Processing Units

When moving processing units, you have to select the partition from which you want the processing units removed and choose the Move Processing Units panel as shown in Figure 5-5 on page 146.

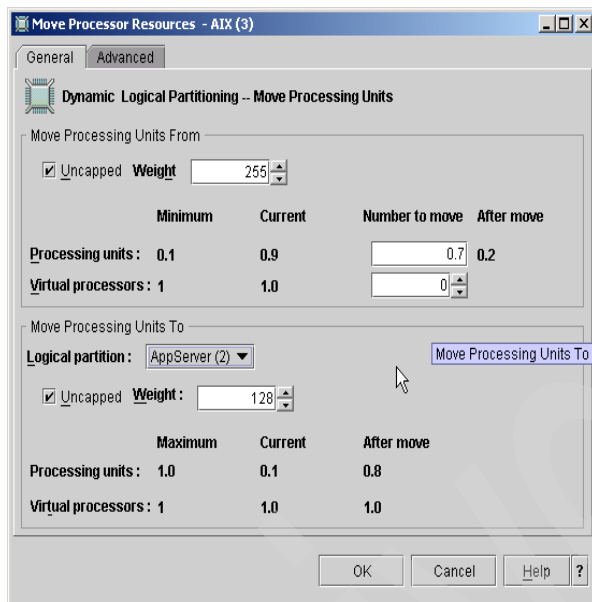


Figure 5-5 Move Processing Units

In the Processing units field, select the amount of processor capacity you want to remove from the selected partition and move to the partition that you select from the menu under Logical Partition. In this example, 0.7 processing units are required to be moved to the Apps_Server partition.

You can also choose to move virtual processors to adjust the number of virtual processors of your partition. This action does not actually move the virtual processor but removes and adds the defined number of Virtual processors to the chosen partitions.

5.1.2 Dynamic partitioning for Virtual Ethernet devices

You can assign and remove Virtual Ethernet resources dynamically. On the HMC, you can assign and remove Virtual Ethernet target and server adapters from a partition using DLPAR. You can also map between physical and virtual resources on the Virtual I/O Server dynamically.

5.1.3 Dynamic partitioning for Virtual SCSI devices

You can assign and remove Virtual SCSI resources dynamically. On the HMC, you can assign and remove Virtual SCSI target and server adapters from a partition using dynamic logical partitioning. You can also map between physical and virtual resources on the Virtual I/O Server dynamically.

5.1.4 Capacity on Demand

Capacity on Demand (CoD) adds operational and configuration flexibility for IBM @server p5 and pSeries systems. CoD is available in a variety of offerings that allow you to pay when purchased, pay after activation, pay before activation, or pay with a one-time cost.

When activating a processor featured for CoD on a system with defined shared processor partitions, the activated processor is assigned automatically to the shared processor pool. You can then decide to add the processor dynamically to a dedicated processor partition or to add capacity entitlement dynamically to the shared processor partitions.

When the system operates as a full system partition, the processor is added automatically to the systems processor capacity.

To remove a CoD processor (for example, when using On/Off CoD, which enables users to temporarily activate processors), you have to make sure that there are enough processing units to deactivate the processor. You can remove the needed capacity entitlement from the partitions dynamically.

A type of CoD is named Reserve CoD. It represents an autonomic way to activate temporary capacity. Reserve CoD enables the user to place a quantity of inactive processors into the server's shared processor pool, which then become available to the pool's resource manager. When the server recognizes the number of base (purchased and active) processors that are assigned across uncapped partitions have been 100% utilized, and at least 10% of an additional processor is needed, then a Processor Day (good for a 24 hour period) is charged against the Reserve CoD account balance. Another Processor Day is charged for each additional processor that is put into use based on the 10% utilization rule. After the 24-hour period elapses and there is no longer a need for the additional performance, no Processor Days are charged until the next performance spike.

DLPAR supports the following dynamic resource changes in a partition without requiring a partition reboot:

- ▶ Resource addition
- ▶ Resource removal

By achieving the resource changes in the following sequence on two partitions in a system, the specified resource can be moved from a partition to another partition:

1. Resource removal from a partition
2. Resource addition to another partition

This resource movement is implemented as single task on the HMC, although it is actually composed of two separate tasks on two partitions internally.

Note: A DLPAR operation can perform only one type of resource change. You cannot add and remove memory to and from the same partition in a single DLPAR operation. Also, you cannot move processor and memory from a partition to another partition in a single DLPAR operation.

Resources that are removed from a partition are marked free (free resources) and are owned by the global firmware of system. These resources are kept in a free resource pool. You can add free resources to any partition in a system as long as the system has enough free resources.

5.2 The process flow of a DLPAR operation

A DLPAR operation initiated on the HMC is transferred to the target partition through Resource Monitoring and Controlling (RMC). The request produces a DLPAR event on the partition. After the event has completed, regardless of the result from the event, a notification is returned to the HMC to mark the completion of the DLPAR operation. Thus, a DLPAR operation is considered a single transactional unit, and only one DLPAR operation is performed at a time.

A DLPAR operation is executed in the process flow is illustrated in Figure 5-6 on page 149.

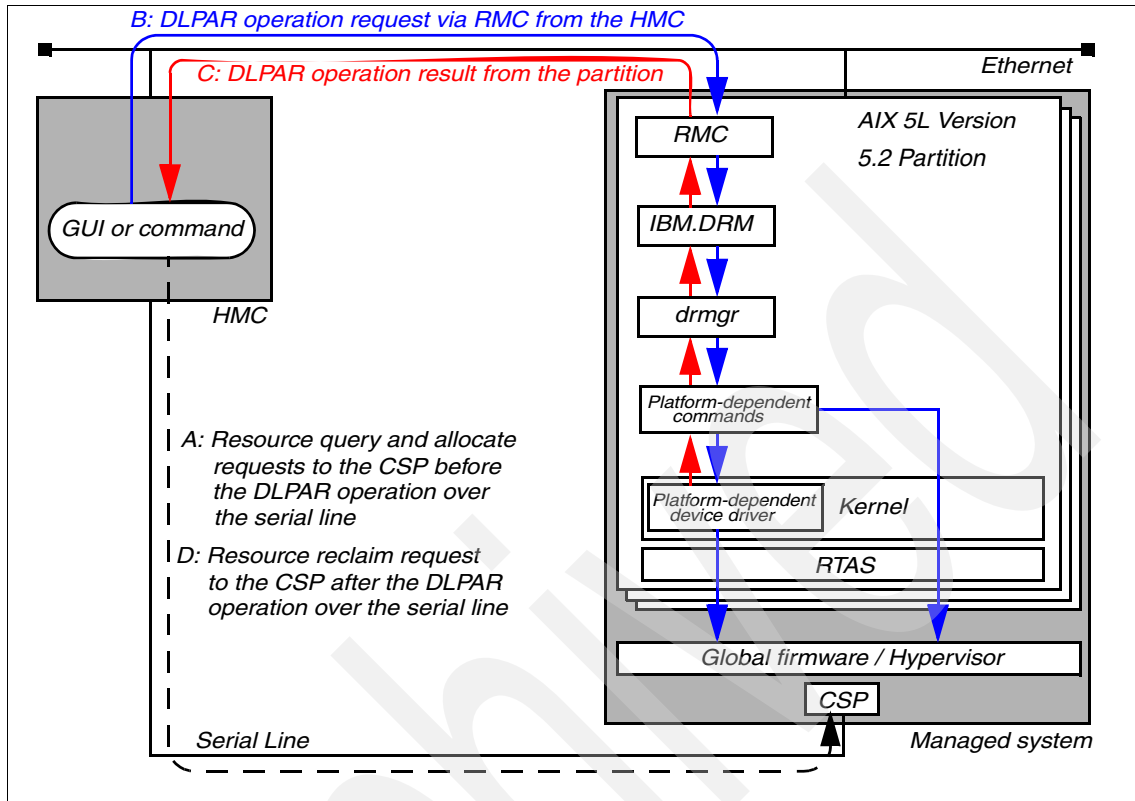


Figure 5-6 Process flow of a DLPAR operation

The following steps explain the process flow of a DLPAR operation:

1. The system administrator initiates a DLPAR operation request on the HMC using either the graphical user interface or command line interface.
2. The requested DLPAR operation is verified on the HMC with the current resource assignment to the partition and free resources on the managed system before being transferred to the target partition. In other words, the HMC provides the policy that determines whether a DLPAR operation request is actually performed on the managed system. The policy is determined by the partition profile. If the request is a resource addition, the HMC communicates with the global firmware to allocate free resources to the target partition through the service processor.
3. If enough free resources exist on the system, the HMC assigns the requested resource to the specified partition, updates the partition's object to reflect this addition, and then creates associations between the partition and the resource to be added.

4. After the requested DLPAR operation has been verified on the HMC, it will be transferred to the target partition using RMC, which is an infrastructure implemented on both the HMC and AIX partitions, as indicated as arrow B in Figure 5-6 on page 149. The RMC is used to provide a secure and reliable connection channel between the HMC and the partitions.

Note: The connection channel established by RMC only exists between the HMC and the partition to where the DLPAR operation is targeted. There are no connection paths required between partitions for DLPAR operation purposes.

5. The request is delivered to the IBM.DRM resource manager running on the partition, which is in charge of the DLPAR function in the RMC infrastructure in AIX. As shown in the following example, the IBM.DRM resource manager is running as the IBM.DRMd daemon process and included in the devices.chrp.base.rte fileset on AIX 5L Version 5.2 or later:

```
# lssrc -ls IBM.DRM
Subsystem      : IBM.DRM
PID            : 18758
Cluster Name   : IW
Node Number    : 1
Daemon start time : Wed Aug 21 16:44:12 CDT 2002

Information from malloc about memory use:
Total Space    : 0x003502c0 (3474112)
Allocated Space: 0x0030b168 (3191144)
Unused Space   : 0x00043e40 (278080)
Freeable Space : 0x00000000 (0)

Class Name(Id) : IBM.DRM(0x2b) Bound
# ps -ef | head -1 ; ps -ef | grep DRMd | grep -v grep
  UID  PID  PPID  C   STIME   TTY   TIME CMD
   root 18758 10444  0   Aug 21    -   0:22 /usr/sbin/rsct/bin/IBM.DRMd
# lslpp -w /usr/sbin/rsct/bin/IBM.DRMd
File                                         Fileset                                Type
-----
/usr/sbin/rsct/bin/IBM.DRMd                 devices.chrp.base.rte                  File
```

Note: The absence of the IBM.DRM resource manager in the `lssrc -a` output does not always mean that the partition has not been configured appropriately for the DLPAR. The resource manager is configured automatically and started by RMC after the first partition reboot, if the network configuration is correctly set up on the partition and the HMC.

6. The IBM.DRM resource manager invokes the **drmgr** command, which is an platform-independent command designed as the focal point of the dynamic logical partitioning support on AIX.

As shown in the following example, the **drmgr** command is installed in the /usr/sbin directory provided by the bos.rte.methods fileset:

```
# whence drmgr
/usr/sbin/drmgr
# ls -lpp -w /usr/sbin/drmgr
```

File	Fileset	Type
/usr/sbin/drmgr	bos.rte.methods	File

Note: The **drmgr** command should not be invoked by the system administrator in order to directly perform resource changes in a partition. It must be invoked in the context explained here to do so. In “How to manage DLPAR scripts” on page 186, another usage of the **drmgr** command is provided.

7. The **drmgr** command invokes several platform-dependent commands depending on the resource type (processor, memory, or I/O resource) and request (resource addition or removal) in order to instruct the kernel to process the actual resource change with necessary information.
8. The kernel does many tasks, as described in Table 5.3 on page 152.
9. After the DLPAR event has completed, regardless of the result, a notification is returned to the HMC to mark the completion of the DLPAR operation, indicated as arrow C in Figure 5-6 on page 149. The notification also includes the exit code, standard out, and standard error from the **drmgr** command. The system administrator who has initiated the DLPAR operation sees the exit code and outputs on the HMC.

If the request is a resource removal, the HMC communicates with the global firmware in order to reclaim resource(s) to the shared or dedicated free resource pool from the source partition through the service processor indicated as arrow D in Figure 5-6 on page 149. The HMC unassigns the resource from the partition and updates the partition’s object to reflect this removal, and then removes associations between the partition and the resource that was just removed.

A DLPAR operation can take noticeable time depending on the availability and the capability to configure or deconfigure a specific resource.

5.3 Internal activity in a DLPAR event

The AIX kernel communicates with the partition firmware through Run-Time Abstraction Services (RTAS). The partition firmware manages resources in the partition). The resources are represented in the Open Firmware device tree that serves as a common reference point for the operating system and firmware. The RTAS operate on objects represented in this database.

Each AIX partition has a private copy of the Open Firmware device tree that reflects the resources that are actually assigned to the partition and those that might be in the future. Structurally, it is organized like a file system with directories and files, where the files represent configured instances of resources, and the directories provide the list of potential assignments. Each installed resource is represented in this list and are individually called dynamic reconfiguration connectors.

5.3.1 Internal activity for processors and memory in a DLPAR event

As described previously, the `drmgr` command handles all DLPAR operations by calling the appropriate commands and controls the process of the reconfiguration of resources.

The following briefly describes the kernel internal activity for processors and memory in a DLPAR event.

1. The Object Data Manager (ODM) lock is taken to guarantee that the ODM, Open Firmware device tree, and the kernel are automatically updated. This step can fail if the ODM lock is held for a long time and the user indicates that the DLPAR operation should have a time limit.
2. The platform-dependent command reads the Open Firmware device tree.
3. The platform-dependent command invokes the kernel to start the DLPAR event. The following steps are taken:
 - a. Requesting validation.
 - b. Locking DLPAR event. Only one event can proceed at a time.
 - c. Saving request in global kernel DR structure that is used to pass information to signal handlers, which runs asynchronously to the platform-dependent command.
 - d. Starting check phase.
4. The check phase scripts are invoked.
5. The check phase signals are sent, conditional wait if signals were posted.

6. The check phase kernel extension callout. Callback routines of registered kernel extensions are called.

The event might fail in steps 4, 5, or 6 if any check phase handler signals an error. After the check phase has passed without an error, and the DLPAR event is in the pre phase, all pre phase application handlers will be called, even if they fail, and the actual resource change is attempted.

7. The kernel marks the start of the pre phase.
8. Pre-phase scripts are invoked.
9. Pre-phase signals are sent—conditional wait, if signals were posted.
10. The kernel marks the doit phase start. This is an internal phase where the resource is either added to or removed from the kernel.

Steps 11-13 can be repeated depending on the request. Processor-based requests never loop; only one shared or dedicated processor can be added or removed at a time in one DLPAR operation. If more than one shared or dedicated processor needs to be added or removed, the HMC invokes AIX once for each processor.

Memory-based requests loop at the LMB level, which represent contiguous from 16 MB segments of logical memory, until the entire user request has been satisfied. The HMC remotely invokes AIX once for the complete memory request.

11. This step is only taken if adding a resource. The Open Firmware device tree is updated. The resource allocated, un-isolated, and the connector configured. When un-isolating the resource, it is assigned to the partition, and ownership is transferred from Open Firmware to AIX:

- For processors, the identity of the global and local interrupt service is discovered.
- For memory, the logical address and size is discovered.

12. Invoke kernel to add or remove resource:

- a. The callback functions of registered kernel extensions are called. Kernel extensions are told the specific resource that is being removed or added.
- b. The resources in the kernel are removed or added.
- c. The kernel extension in post or posterror phase are invoked.

If steps a or b fail, the operation fails.

13. This step is only taken if removing a resource.

The Open Firmware device tree is updated. Resources are isolated and unallocated for removal. The Open Firmware device tree must be kept updated so that the configuration methods can determine the set of resources that are actually configured and owned by the operating system.

14. Kernel marks post (or posterror) phase start depending on the success of the previous steps.
15. Invoke configuration methods so that DLPAR-aware applications and registered DLPAR scripts will see state change in the ODM.
16. The post scripts are invoked.
17. The post signals are sent to registered processes, conditional wait if signals were posted.
18. The kernel clears the DLPAR event.
19. ODM locks are released.

5.3.2 Internal activity for I/O slots in a DLPAR event

Dynamic removal and addition of I/O adapters has been provided by AIX prior to DLPAR support, utilizing the PCI adapter Hot Plug capability on the IBM RS/6000 and IBM pSeries server models. To allow for the dynamic addition and removal of PCI I/O slots, AIX 5L Version 5.2 provided enhancements to the **lsslot** command have been made.

PCI slots and integrated I/O devices can be listed using the new connector type slot in the **lsslot** command, as shown in the following example:

```
# lsslot -c slot
The output of this command looks similar to the following:
#Slot Description Device(s)
U1.5-P1-I1 DLPAR slot pci13 ent0
U1.5-P1-I2 DLPAR slot pci14 ent1
U1.5-P1-I3 DLPAR slot pci15
U1.5-P1-I4 DLPAR slot pci16
U1.5-P1-I5 DLPAR slot pci17 ent2
U1.5-P1/Z1 DLPAR slot pci18 scsi0
```

Before the I/O slot removal, you must delete the PCI adapter device and all its child devices from AIX. Given that ent2 in the slot U1.5-P1-I5 in the previous example is not used, the devices could be removed using the following command as the root user on the partition.

```
# rmdev -l pci17 -d -R
```

After the devices have been removed from AIX, the I/O slot can be removed from the partition using the graphical user interface or command line interface on the HMC.

Note: Any PCI slots defined as *required* are not eligible for the DLPAR operation.

To let AIX recognize the dynamically added I/O slot and its children devices to a partition, you must invoke the **cfgmgr** command as the root user on the partition. To add the previously removed I/O slot from a partition, it first needs to be reassigned to the partition using the HMC.

5.4 DLPAR-safe and DLPAR-aware applications

The dynamic logical partitioning function was first introduced on AIX 5L Version 5.2 and was designed and implemented to not impact the existing applications. In fact, most applications are not affected by any DLPAR operations results. Therefore, those applications are called *DLPAR-safe* applications.

There are two types of application classifications regarding DLPAR operations:

DLPAR-safe	Applications that do not fail as a result of DLPAR operations. The application's performance can suffer when resources are removed, or it cannot scale as resources are added.
DLPAR-aware	Applications that incorporate DLPAR operations that allow the application to adjust its use of the system resources equal to the actual capacity of the system. DLPAR-aware applications are always DLPAR-safe.

5.4.1 DLPAR-safe

Although, most applications are DLPAR-safe without requiring any modification, there are certain instances where programs might not be inherently DLPAR-safe. There are two cases where DLPAR operations can introduce undesirable effects in the application:

- ▶ Programs that are optimized for uni-processors can have problems when a processor is added to the system resources.
- ▶ On programs that are indexed by processor numbers, the increased processor number can cause the code to go down an unexpected code path during its run-time checks.

In addition, applications that use uni-processor serialization techniques can experience unexpected problems. In order to resolve these concerns, system administrators and application developers need to be aware of how their applications get the number of processors.

5.4.2 DLPAR-aware

DLPAR-aware applications adapt to system resource changes that are caused by DLPAR operations. When these operations occur, the application recognizes the resource change and accommodate accordingly.

You can use the following techniques to make applications DLPAR-aware:

- ▶ Consistently poll for system resource changes. Polling is not the recommended way to accommodate for DLPAR operations, but it is valid for systems that do not need to be tightly integrated with DLPAR. Because the resource changes might not be discovered immediately, an application that uses polling can have limited performance. Polling is not suitable for applications that deploy processor bindings, because they represent hard dependencies.
- ▶ Applications have other methods to react to the resource change caused by DLPAR operations. See “Integrating a DLPAR operation into the application” on page 157.
- ▶ Several applications should be made DLPAR-aware, because, they need to scale with the system resources. These types of applications can increase their performance by becoming DLPAR-aware. Table 5-1 lists some examples of applications that should be made DLPAR-aware.

Note: These are only a few types of common applications affected by DLPAR operations. The system administrator and application developer should be sensitive to other types of programs that might need to scale with resource changes.

Table 5-1 Applications that should be DLPAR-aware

Application type	Reason
Database applications	The application needs to scale with the system. For example, the number of threads might need to scale with the number of available processors, or the number of large pinned buffers might need to scale with the available system memory.
Licence Managers	Licenses are distributed based on the number of available processors or the memory capacity.
Workload Managers	Jobs are scheduled based on system resources, such as available processors and memory.
Tools	Certain tools might report processor and memory statistics or rely on available resources.

5.5 Integrating a DLPAR operation into the application

The DLPAR operation can be integrated into the application using the following methods:

- ▶ Script-based DLPAR event handling

If the application is controlled externally to use a specific number of threads or to size its buffers, use this method. In order to facilitate this method, a new command, **drmgr**, is provided. The **drmgr** command is the central focal point of the DLPAR function of AIX. The following several sections discuss the **drmgr** command and typical usage examples are provided in “How to manage DLPAR scripts” on page 186.

See “Script-based DLPAR event handling” on page 160 for more information.

- ▶ API-based DLPAR event handling

If the application is directly aware of the system configuration, and the application source code is available, use this method.

See “API-based DLPAR event handling” on page 194 for more information.

Applications can monitor and respond to various DLPAR events, such as a memory addition or processor removal, by using these two methods. Although, at the high-level, both methods share the same DLPAR events flow, several key differences exist between these two methods.

One difference is that the script-based method externally reconfigures the application once a DLPAR event takes place, while the API-based method can be directly integrated into the application by registering a signal handler so that the process can be notified with the SIGRECONFIG signal when the DLPAR event occurs.

Note: The DLPAR events of I/O resources do not notify applications.

5.5.1 Three phases in a DLPAR event

A DLPAR event executes in three phases: check, pre, and post. Each phase is an automatic execution unit and is executed in its entirety before the next phase is started, preventing partial updates to the system. In the pre and post phases, the state of the application is permitted to change. The operating system only acts upon DLPAR requests between the pre and post phases to perform the actual resource change.

Note: If a dynamic processor deallocation occurs in a partition that is running AIX 5L Version 5.2 or later, it is also treated as a processor removal DLPAR event, and thus invokes these three phases.

Figure 5-7 illustrates the three phases and the order in which they occur for a DLPAR event.

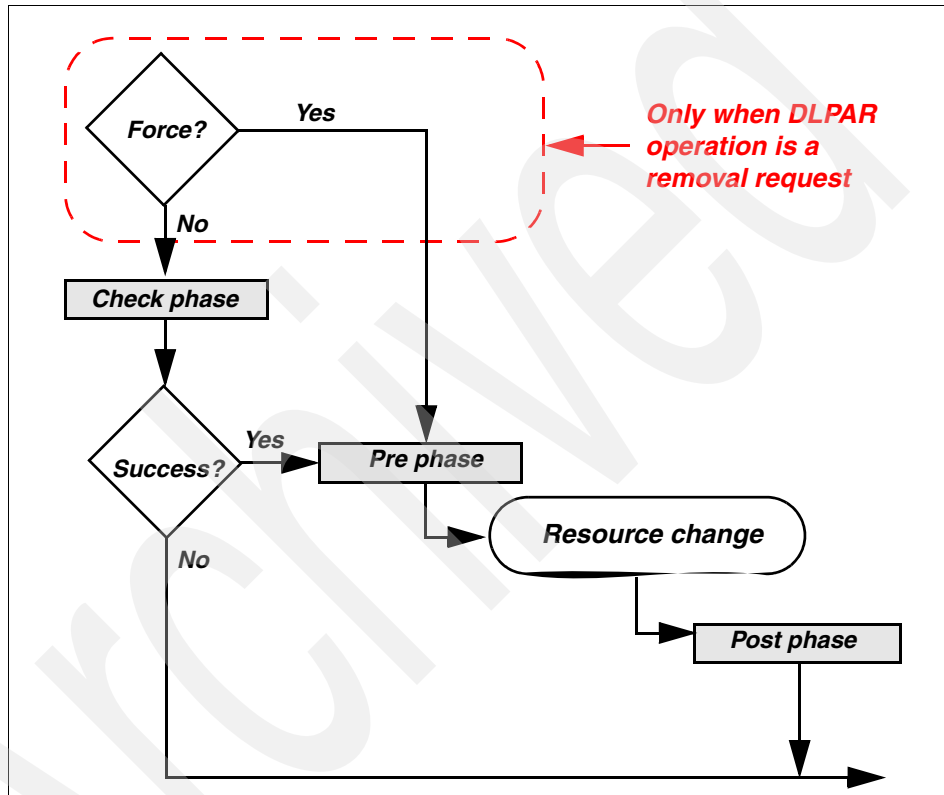


Figure 5-7 Three DLPAR phases of a DLPAR event

Check phase

The check phase usually occurs first. It is used to examine the resource's state and to determine if the application can tolerate a DLPAR event. It gives the script or API a chance to fail the current DLPAR operation request without changing any system state.

Note: In a resource removal DLPAR event, the check phase is skipped if the force option is specified. In a resource addition DLPAR event, the check phase is not skipped regardless of the force option value.

The check phase can be used in several situations, including the following:

- ▶ To determine if a processor cannot be removed because it still has threads bound to it.
- ▶ By a licence manager to fail the integration of a new processor to the partition because it does not have a valid licence to support the addition of a processor.
- ▶ To maintain an application's DLPAR safeness by restricting the effects of DLPAR operations. For instance, if the application is optimized for a uniprocessor environment, the check phase could prevent the application from recognizing the addition of a processor, which could prevent the application from executing an unexpected code path with the presence of additional processors.

Note: If a DLPAR script exits with failure from the check phase, the DLPAR event will not continue. Therefore, the resource change is not performed, and the DLPAR script is not invoked in the pre and post phases.

Pre phase

Before the actual resource change is made, the application is notified that a resource change (addition or removal) is about to occur. The application is given a chance to prepare for the DLPAR request in the pre phase.

When the application is expecting a resource removal, the DLPAR script or API needs to carefully utilize this phase. This phase handles such things as unbinding processors, detaching pinned shared memory segments, removing plocks, and terminating the application if it does not support DLPAR or will be broken by DLPAR requests.

Note: The actual resource change takes place between the pre and post phases.

Post phase

After a resource change has occurred, the application will have a chance to respond to the DLPAR operation. The application can reconfigure itself in the post phase in order to take advantage of the resource addition or to compensate for the resource removal.

If resources are added, the DLPAR script or API could create new threads or attach to pinned shared memory segments. On the other hand, if resources are removed, the DLPAR scripts or API calls might delete threads for scalability.

5.5.2 Event phase summary

When a DLPAR request is made to change resource configurations in a partition, the **drmgr** command notifies applications of the pending resource change. Table 5-2 summarizes the phases of a DLPAR event and some important considerations of what needs to be accomplished in each phase.

Table 5-2 Considerations during each event phase

Phase	Considerations
Check	<ul style="list-style-type: none"> ▶ Can the application support the request? ▶ Are there licence restrictions? ▶ Can the system withstand this application failing?
Pre	<ul style="list-style-type: none"> ▶ Is it best to stop the application and then restart it after the DLPAR operation? ▶ How can the application help facilitate a DLPAR removal or addition? ▶ What can the application eliminate or reduce when a resource is removed? (that is, kill threads)
Post	<ul style="list-style-type: none"> ▶ Does the application need to be restarted after the DLPAR operation? ▶ How can the application take advantage of added resource? (that is, start new threads) ▶ Did the operation complete? Was there a partial success?

5.6 Script-based DLPAR event handling

The script-based DLPAR event handling method is performed by several components, as explained in the following (see Figure 5-8 on page 162):

1. A DLPAR operation request is initiated using either the graphical user interface or command line interface on the HMC.
2. The request is transferred to the target partition through RMC. The IBM.DRM resource manager on the partition receives this request.
3. The IBM.DRM resource manager invokes the **drmgr** command with the necessary information that represents a DLPAR event.

4. The **drmgr** command invokes registered DLPAR scripts depending on the resource type, processor, or memory that is specified by the DLPAR event. The information about the registered DLPAR scripts is kept in the DLPAR script database and fetched by the **drmgr** command.
5. The invoked DLPAR scripts perform necessary tasks that integrate the DLPAR operation into the application.

The DLPAR scripts should satisfy the application's demands when a DLPAR event takes place so that the application can take the appropriate actions. Therefore, DLPAR scripts are carefully developed and tested in order for the applications' DLPAR-awareness.

The DLPAR script can use the following commands in order to resolve the application processes' resource dependency:

ps	To display bindprocessor attachments and plock system call status at the process level.
bindprocessor	To display online processors and make new attachments.
kill	To send signals to processes.
ipcs	To display pinned shared memory segments at the process level.
lsrset	To display processor sets.
lsclass	To display Workload Manager (WLM) classes, which might include processor sets.
chclass	To change WLM class definitions.

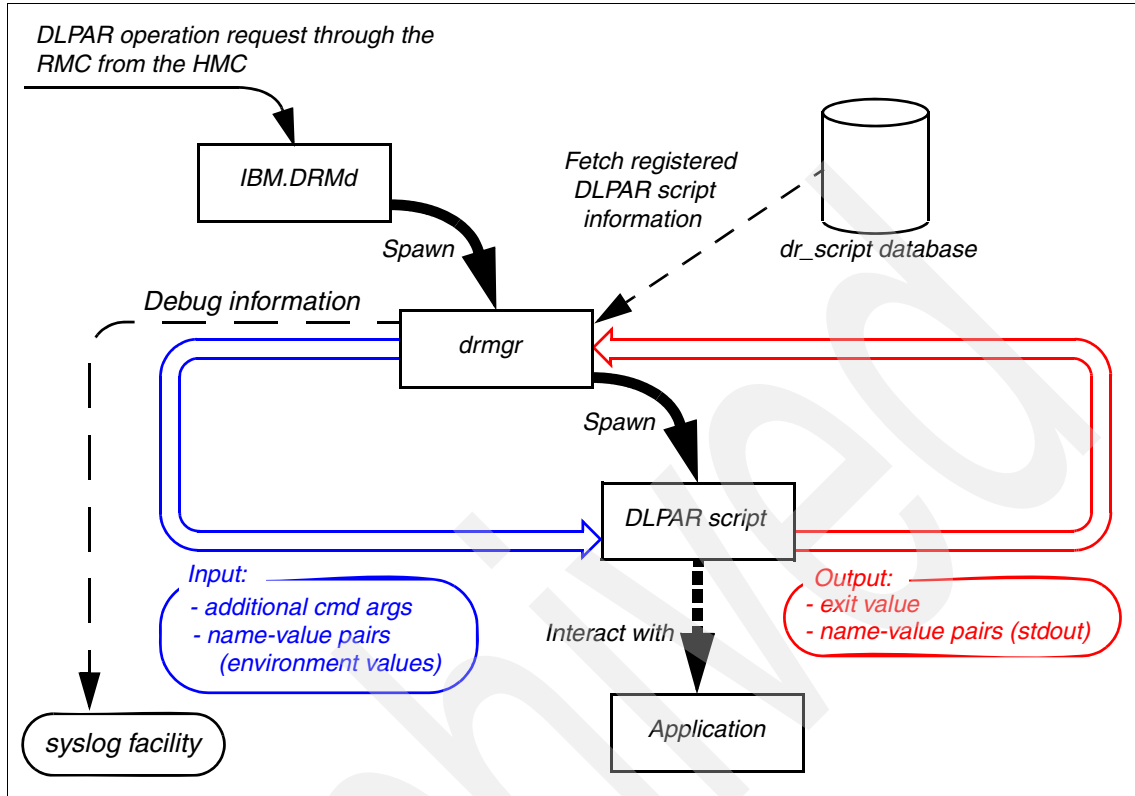


Figure 5-8 A DLPAR script invoked by the `drmgr` command

5.6.1 Script execution environment

When DLPAR scripts are invoked by the `drmgr` command, it sets up the following script execution environment. The required information to set up this environment is taken from the DLPAR script database and the DLPAR event.

- ▶ The UID and GID of the execution process are set to the ones of the DLPAR script.
- ▶ The current working directory is changed to `/tmp`.
- ▶ The PATH environment variable is set to `/usr/bin:/etc:/usr/sbin`.
- ▶ Two pipes are established between `drmgr` and the executing process so that the process reads using the standard in from the `drmgr` command and writes using the standard out to the `drmgr` command.

As illustrated in Figure 5-8, the execution environment defines the input and output for the DLPAR script process.

When the DLPAR script is invoked, the DLPAR script process receives its input using the following two ways:

- ▶ Additional command line arguments

When a DLPAR script is called, the **drmgr** command will invoke it as follows:

```
dr_application_script <sub-command> <additional_cmd_arg>
```

In addition to the subcommands, which are explained in “DLPAR script subcommands” on page 167, additional command arguments can be passed to the script.

- ▶ Environment variables with specified format

When a DLPAR script is called, the **drmgr** command passes several environmental variables using a name-value pair format.

Environmental variables that start with *DR_* are primarily used to send input data to DLPAR scripts; therefore, they should be exclusively set aside for the **drmgr** command.

There are three types of environment values:

- General environment values (see Table 5-3 on page 164)
- processor-specific environment values (Table 5-4 on page 165)
- Memory-specific environment values (Table 5-5 on page 166)

Note: These environment variables only exist during DLPAR events. If you want to view these variable values, the script needs to be coded to write these variables to the standard out using *DR_LOG_** variables so that the **drmgr** command can forward these output to the syslog facility (see Table 5-4 on page 165).

The DLPAR script process produces its output using the following two ways:

- ▶ Exit values

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, then the *DR_ERROR* name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

- ▶ Standard out with specified format

The DLPAR scripts can write strings using a name-value pair format to the standard out. The **drmgr** command will read them from the scripts. Strings that start with *DR_* are primarily used to send output data to the **drmgr** command from the DLPAR scripts.

Note: The script should not print the following to the standard out:

- ▶ A string whose length is larger than 1024 characters.
- ▶ A string that contains new line characters.
- ▶ Any strings that are undefined in Table 5-6 on page 166, Table 5-9 on page 171, Table 5-10 on page 173, and Table 5-11 on page 174.

Input (environment variables)

Table 5-3 shows general environment variables.

Table 5-3 General DLPAR environment variables

Environment variable	Description
DR_DETAIL_LEVEL=N	This name-value pair instructs the script to produce the specified level of detailed debug information sent to the standard out. The value of N must be one of the following: <ul style="list-style-type: none">▶ 0 - None▶ 1 - Min▶ 2 - Medium/more▶ 3 - Max▶ 4 - Debug
DR_FORCE=emergency	This name-value pair gives the emergency processing request to the script. The value of emergency must be one of the following: <ul style="list-style-type: none">▶ FALSE - Emergency processing is required.▶ TRUE - Emergency processing is not required (default).

Note: The DR_DETAIL_LEVEL=N environment value is set on the HMC. If you use the graphical user interface, select **Detail level** in the DLPAR operation panel. If you use the command line interface, use the **-d** option of the **chhwres** command to set the value.

Table 5-4 shows processor-specific environment variables.

Table 5-4 Processor-specific DLPAR environment variables

Processor environment variables	Description
DR_LCPUID=N	The logical CPU ID of the processor that is being added or removed. N is a decimal number.
DR_BCPUID=N	The bind CPU ID of the processor that is being added or removed. N is a decimal number.
DR_CPU_CAPACITY=N	Capacity is not expressed as a fraction in the above parameters. Capacity is expressed as a percentage, where 100 represents one physical processor, and 180 represents the power of 1.8 processors. The environment variables DR_CPU_CAPACITY and DR_VAR_WEIGHT represent the value of the partition attribute before the request was made, so the script will have to internally add or subtract the delta to determine the result of the request.
DR_CPU_CAPACITY_DELTA=N	Capacity is not expressed as a fraction in the above parameters. Capacity is expressed as a percentage, where 100 represents one physical processor, and 180 represents the power of 1.8 processors. The environment variables DR_CPU_CAPACITY and DR_VAR_WEIGHT represent the value of the partition attribute before the request was made, so the script will have to internally add or subtract the delta to determine the result of the request.
DR_VAR_WEIGHT=N	The environment variables DR_CPU_CAPACITY and DR_VAR_WEIGHT represent the value of the partition attribute before the request was made, so the script will have to internally add or subtract the delta to determine the result of the request.
DR_VAR_WEIGHT_DELTA=N	The environment variables DR_CPU_CAPACITY and DR_VAR_WEIGHT represent the value of the partition attribute before the request was made, so the script will have to internally add or subtract the delta to determine the result of the request.

Table 5-5 shows the memory-specific environment variables.

Table 5-5 Memory-specific DLPAR environment variables

Memory environment variables	Description
DR_MEM_SIZE_REQUEST=N	Size of memory requested in megabytes. N is a decimal value.
DR_MEM_SIZE_COMPLETED=N	Number of megabytes that were successfully added or removed. N is a decimal value.
DR_FREE_FRAMES=N	Number of free frames currently in the system. Each frame is a 4 KB page. N is a 32-bit hexadecimal value.
DR_PINNABLE_FRAMES=N	Total number of pinnable frames currently in the system. Each frame is a 4 KB page. N is a 32-bit hexadecimal value.
DR_TOTAL_FRAMES=N	Total number of frames in the system. Each frame is a 4 KB page. N is a 32-bit hexadecimal value.

Output (standard out)

Table 5-6 shows general output variables. The DR_ERROR=failure_cause name=variable pair is a mandatory output when the script exits with 1 (failure).

Table 5-6 General DLPAR output variables

Variable	Description
DR_ERROR=failure_cause (only if the script exits with 1)	This name-value pair describes the reason for failure.
DR_LOG_ERR=message	This name-value pair describes the information message to be sent to the syslog facility with the err (LOG_ERR) priority.
DR_LOG_WARNING=message	This name-value pair describes the information message to be sent to the syslog facility with the warning (LOG_WARNING) priority.
DR_LOG_INFO=message	This name-value pair describes the information message to be sent to the syslog facility with the info (LOG_INFO) priority.
DR_LOG_EMERG=message	This name-value pair describes the information message to be sent to the syslog facility with the emerg (LOG_EMERG) priority.

Variable	Description
DR_LOG_DEBUG=message	This name-value pair describes the information message to be sent to the syslog facility with the debug (LOG_DEBUG) priority.

Note: Except for the DR_ERROR variable, the other variables are used to send messages to the syslog facility.

5.6.2 DLPAR script naming convention

When developing a DLPAR script, you should follow a few simple naming conventions. It is preferable to name the script using prefixes that describe the vendor name and the subsystem that it controls.

For example, dr_ibm_wlm.pl would be a good name for a DLPAR Perl script that was written by IBM to control the WLM assignments. WLM is a standard function of AIX to prioritize multiple processes depending on the predefined attributes.

Another example is dr_sysadmin_wlm.pl. This name could be a DLPAR Perl script provided by system administrator to control the WLM assignments.

5.7 DLPAR script subcommands

Every DLPAR script is required to accept all the subcommands found in Table 5-7 on page 168. This section provides detailed information for each subcommand.

Note: The prefix names for these subcommands (check, pre, and post) coincide with the DLPAR phases that are explained in “Script-based DLPAR event handling” on page 160.

Table 5-7 DLPAR script subcommands

Subcommand name	Description
scriptinfo	Identifies script-specific information. It must provide the version, date, and vendor information. This command is called when the script is installed.
register	Identifies the resources managed by the script, such as <i>cpu</i> or <i>mem</i> .
usage resource_name	Returns a description of how the script plans to use the named resources. It contains pertinent information so that the user can determine whether or not to install the script. Further, the command describes the software capabilities of the applications that are impacted.
checkrelease resource_name	This subcommand is invoked when the drmgr command initiates the release of the specified resource. The script checks the resource dependencies of the application and evaluate the effects of resource removal on the application the script is monitoring. The script can indicate that the resource should not be removed if the application is not DLPAR-aware or if the resource is critical for the subsystem.
prerelease resource_name	Before the removal of the specified resource, this subcommand is invoked. The script uses this time to remove any dependencies the application can have on the resource. This command can reconfigure, suspend, or terminate the application such that the named resource can be released.
postrelease resource_name	After the resource is removed successfully, this subcommand is invoked. The script can perform any necessary cleaning up, or it can restart the application if it stopped the application in the prerelease phase.
undoprerelease resource_name	This subcommand is invoked if an error occurs while the resource is being released. The script takes the necessary steps to undo its prerelease operations on the resource and the application. In the case of a partial resource release, this command reads the environment variables to determine the level of success before the fail.

Subcommand name	Description
checkacquire resource_name	This subcommand is invoked to determine if the drmgr command can proceed with a resource addition to the application.
preacquire resource_name	This subcommand tells the application that a resource will be available for use.
postacquire resource_name	This subcommand informs the drmgr command that the resource addition completed, and the script allows the application to use the new resources. If the application was stopped in the preacquire phase, the application is restarted in this command.
undopreacquire resource_name	This subcommand notifies the drmgr command that the resource addition aborted or partially completed. The script then makes the necessary changes to undo anything it did in the preacquire phase, or the script determines the level of success of the DLPAR addition request by reading the environment variables.

5.7.1 The scriptinfo subcommand

When a script is first installed, the script is invoked with the scriptinfo subcommand by the **drmgr** command. The scriptinfo subcommand displays useful information to identify the script, such as the developed date and the vendor name for it, in order to let the **drmgr** command record appropriate information in the DLPAR script database about the script. The scriptinfo subcommand is also called by the **drmgr** command in the very early stage of a DLPAR operation request.

When the script is invoked with the scriptinfo subcommand, it takes the following syntax:

```
dr_application_script scriptinfo
```

Input to the scriptinfo subcommand

The **scriptinfo** subcommand takes the following input data:

- ▶ Additional command line arguments
None.
- ▶ Name-value pairs from environment variables
See Table 5-8 on page 170.

Output from the scriptinfo subcommand

The **scriptinfo** subcommand produces the following output data.

► Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the **DR_ERROR** name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

► Name-value pairs to the standard output stream

Table 5-8 lists the name-value pairs that must be returned by the script when it is invoked with the **scriptinfo** subcommand.

Table 5-8 Required output name-value pairs for the scriptinfo subcommand

Required output pair	Description
DR_VERSION=1	This name-value pair indicates the version level of the script that specifies the compatibility level of the DLPAR script with respect to the DLPAR implementation version of AIX. On AIX 5L Version 5.2, the version must be set to 1, which indicates that the script is compatible with DLPAR implementation Version 1.
DR_DATE=DDMMYYYY	This name-value pair is the publication date of the script. The format should be DDMMYYYY, where DD=days, MM=months, and YYYY=year. For example, a valid date would be 08102002, which is October 8, 2002.
DR_SCRIPTINFO=description	This name-value pair contains a description of the script's functionality. This string should be a brief human-readable message.
DR_VENDOR=vendor_information	This name-value pair indicates the vendor name and related information. This string can also be used to highlight the application represented by the script.

In addition to Table 5-8, Table 5-9 on page 171 lists the optional name-value pair that can be returned by the script when it is invoked with the **scriptinfo** subcommand. If the script needs to have more processing time for its execution, it prints the timeout value to the standard out explained in Table 5-10 on page 173, so that the **drmgr** command can read the appropriate timeout value for this script.

Table 5-9 Optional output name-value pairs for the scriptinfo subcommand

Optional output pair	Description
DR_TIMEOUT=timeout_in_seconds	This name-value pair indicates the timeout value in seconds of all DLPAR operations done in this script. The default timeout is 10 seconds. This timeout can be overridden by the -w flag of the drmgr command. The drmgr command waits for the timeout before it sends a SIGABRT to the script. After waiting 1 more second for the script to gracefully end, it will send a SIGKILL. A value of zero (0) disables the timer.

Example

In Example 5-1, two sample DLPAR scripts are registered, `dr_test.sh` and `dr_test.pl`. The emphasized lines in this example show the information recorded in the DLPAR script database. The information was derived from the script output with the `scriptinfo` subcommand upon the script registration (see Table 5-8 on page 170).

Also, the two fields, Script Timeout and Admin Override Timeout, correspond to the values specified by the `DR_TIMEOUT` value and the `-w` option, respectively (see Table 5-9).

Example 5-1 Registered sample DLPAR scripts

```
# drmgr -l
DR Install Root Directory: /usr/lib/dr/scripts/all
Syslog ID: DRMGR
-----
/usr/lib/dr/scripts/all/dr_test.sh           DLPAR ksh example script
Vendor:IBM, Version:1, Date:10182002
Script Timeout:10, Admin Override Timeout:0
Resources Supported:
Resource Name: cpu           Resource Usage: cpu binding for performance
Resource Name: mem           Resource Usage: Shared(Pinned) memory for
app XYZ
-----
/usr/lib/dr/scripts/all/dr_test.pl           DLPAR Perl example script
Vendor:IBM Corp., Version:1, Date:04192002
Script Timeout:5, Admin Override Timeout:0
Resources Supported:
Resource Name: cpu           Resource Usage: Testing DLPAR on CPU removal
Resource Name: mem           Resource Usage: Testing DLPAR on MEM removal
-----
```

5.7.2 The register subcommand

When the script is invoked with the **register** subcommand by the **drmgr** command, the script is registered into the DLPAR script database. The **register** subcommand also informs the **drmgr** command about the resource type (processor or memory) that the script is designed to handle.

When the script is invoked with the **register** subcommand, it takes the following syntax:

```
dr_application_script register
```

Input to the register subcommand

The **register** subcommand takes the following input data:

- ▶ Additional command line arguments
None.
- ▶ Name-value pairs from environment variables
See Table 5-10 on page 173.

Output from the register subcommand

The **register** subcommand produces the following output data:

- ▶ Exit value
The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the **DR_ERROR** name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.
- ▶ Name-value pairs to the standard output stream
Table 5-10 on page 173 lists the name-value pair that must be returned by the script when it is invoked with the register subcommand.

Table 5-10 Required output name-value pair for the register subcommand

Required output pair	Description
DR_RESOURCE=resource_name	<p>This string identifies the resource type that the DLPAR script is designed to handle. The valid resource type names are:</p> <ul style="list-style-type: none">▶ <code>cpu</code>▶ <code>capacity</code> = capacity changes to entitled processor capacity▶ <code>var_weight</code> = changes to the variable capacity weight▶ <code>mem</code> <p>If a script needs to handle both processor and memory resource types, the script prints the following two lines: DR_RESOURCE=cpu DR_RESOURCE=mem</p>

Optionally, the script can return the name-value pairs listed in Table 5-9 on page 171.

Note: The resource types, `capacity` and `var_weight`, have been added to support shared partitions and virtual processors in @server p5 servers.

Example

The emphasized fields in the following example are extracted from Example 5-1 on page 171. The fields show the information that is recorded in the DLPAR script database. The information was derived from the script output with the **register** subcommand upon the script registration (see Table 5-10).

Resources Supported:
Resource Name: **cpu** Resource Usage: Testing DLPAR on CPU removal
Resource Name: **mem** Resource Usage: Testing DLPAR on MEM removal

5.7.3 The usage subcommand

The main purpose of the **usage** subcommand is to tell you which resource type (processor or memory) the script is designed to handle. The **usage** subcommand is also called by the **drmgr** command in the very early stage of a DLPAR operation request for information purposes only.

When the script is invoked with the **usage** subcommand, it takes the following syntax:

`dr_application_script usage <resource_type>`

Input to the usage subcommand

The **usage** subcommand takes the following input data:

- ▶ Additional command line arguments
The **usage** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.
- ▶ Name-value pairs from environment variables
See Table 5-3 on page 164.

Output from the usage subcommand

The **usage** subcommand produces the following output data:

- ▶ Exit value
The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.
- ▶ Name-value pairs to the standard output stream
Table 5-11 lists the name-value pair that must be returned by the script when it is invoked with the **usage** subcommand.

Table 5-11 Required output name-value pair for the usage subcommand

Required output pair	Description
DR_USAGE=usage_description	This name-value pair contains a human-readable string describing how the resource is used by the associated application. This description should indicate the impact on the application if that resource is removed or added.

Optionally, the script can return the name-value pairs listed in Table 5-9 on page 171.

Example

The emphasized fields in the following example are extracted from Example 5-1 on page 171. The fields show the information recorded in the DLPAR script database. The information was derived from the script output with the **usage** subcommand upon the script registration (see Table 5-10 on page 173).

Resources Supported:

Resource Name: cpu

Resource Name: mem

Resource Usage: Testing DLPAR on CPU removal

Resource Usage: Testing DLPAR on MEM removal

5.7.4 The checkrelease subcommand

Before the specified resource type is removed, the script is invoked with the **checkrelease** subcommand by the **drmgr** command. The resource is not actually changed with this subcommand.

When the **drmgr** command invokes the script with the **checkrelease** subcommand, the script determines the resource dependencies of the application, evaluate the effects of resource removal on the application, and indicate whether the resource can be successfully removed. If the resource removal request affects the application, the script returns with an exit status of 1 to let the **drmgr** command know to not release the resource.

When the script is invoked with the **checkrelease** subcommand, it takes the following syntax:

```
dr_application_script checkrelease <resource_type>
```

Input for the checkrelease subcommand

The **checkrelease** subcommand takes the following input data:

- ▶ Additional command line arguments

The **checkrelease** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.

- ▶ Name-value pairs from environment variables

The **checkrelease** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **checkrelease** subcommand can also take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: If the `DR_FORCE=TRUE` environment value is passed to a script with `prerelease`, the script interprets the force option as an order, so it returns as soon as possible.

Output for the `checkrelease` subcommand

The `checkrelease` subcommand produces the following output data:

- Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

Note: When invoking scripts in the `prerelease` phase, the failure of a script does not prevent the `drmgr` command from attempting to remove the resource. The theory is that resource removal is safe. It can fail, but the kernel is coded to cleanly remove resources, so there is no harm in trying. The return code from each script is stored so that the `drmgr` command can determine whether it needs to call it back. If a script fails in the `prerelease` phase, it will not be called in the `postrelease` or `undorelease` phases.

- Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-9 on page 171.

5.7.5 The `prerelease` subcommand

Before the specified resource type is actually released, the script is invoked with the `prerelease` subcommand by the `drmgr` command. This is called after the `checkrelease` subcommand.

When the `drmgr` command invokes the script with the `prerelease` subcommand, the script interacts with the application, as briefly summarized in the following:

1. Informs the application about the resource removal event and lets the application release the specified resource, for example, reconfigure, suspend, or terminate the application process that uses the specified resource.
2. If the application has successfully released the specified resource, the script exits with an exit status of 0 (success).

3. Otherwise, there are two options:
 - The script exits with an exit status of 0 (success) regardless of the response from the application.
 - The script exits with an exit status of 1 (failure).

When the script is invoked with the **prerelease** subcommand, it takes the following syntax:

```
dr_application_script prerelease <resource_type>
```

Input for the prerelease subcommand

The **prerelease** subcommand takes the following input data:

- ▶ Additional command line arguments

The **prerelease** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.

- ▶ Name-value pairs from environment variables

The **prerelease** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **prerelease** subcommand can also take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: If the `DR_FORCE=TRUE` environment value is passed to a script with the **prerelease** subcommand, the script returns as soon as possible.

Output for the prerelease subcommand

The **prerelease** subcommand produces the following output data:

- ▶ Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

Note: If the script exits with 1 (failure), the **drmgr** command will not perform actual resource removal; however, it will invoke subsequent events (**postrelease** and **undoprerelease**) against the specified resource.

- Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.7.6 The **postrelease** subcommand

After the specified resource type has been released from the partition, the script is invoked with the **postrelease** subcommand by the **drmgr** command. This is called after the **prerelease** subcommand.

When the **drmgr** command invokes the script with the **postrelease** subcommand, the script interacts with the application, including any necessary cleanup, for example, restarting or resuming the application if it was quiesced in the **prerelease** subcommand.

The script also takes appropriate actions if a partial success occurs. A partial success occurs when a subset of the requested number of resources was successfully removed. For example, the memory-related environment variables are checked to determine if all the requested memory frames were removed.

When the script is invoked with the **postrelease** subcommand, it takes the following syntax:

```
dr_application_script postrelease <resource_type>
```

Input for the **postrelease** subcommand

The **postrelease** subcommand takes the following input data:

- Additional command line arguments

The **postrelease** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are **cpu** or **mem**.

- Name-value pairs from environment variables

The **postrelease** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **postrelease** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: The **force** option should be ignored.

Output for the `postrelease` subcommand

The `postrelease` subcommand produces the following output data:

- ▶ Exit value
The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.
- ▶ Name-value pairs to the standard output stream
Optionally, the script can return the name-value pairs listed in Table 5-9 on page 171.

5.7.7 The `undoprerelease` subcommand

If the `drmgr` command fails to release the specified resource, it invokes the script with the `undoprerelease` subcommand to recover any necessary clean-up tasks that were done by the `prerelease` subcommand. The script undoes any actions that were taken by the script in the `prerelease` subcommand.

Note: If the specified resource has been removed successfully, the `drmgr` command will not invoke the script with the `undoprerelease` subcommand.

When the script is invoked with the `undoprerelease` subcommand, it takes the following syntax:

```
dr_application_script undoprerelease <resource_type>
```

Input for the `undoprerelease` subcommand

The `undoprerelease` subcommand takes the following input data:

- ▶ Additional command line arguments
The `undoprerelease` subcommand requires one additional command line argument that tells the `drmgr` command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.
- ▶ Name-value pairs from environment variables
The `undoprerelease` subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:
 - If the script is registered to handle processor, see Table 5-4 on page 165.
 - If the script is registered to handle memory, see Table 5-5 on page 166.

The **undoprerelease** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: The force option should be ignored.

Output for the undoprerelease subcommand

The **undoprerelease** subcommand produces the following output data:

- ▶ Exit value
The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the DR_ERROR name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.
- ▶ Name-value pairs to the standard output stream
Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.7.8 The checkacquire subcommand

Before the specified resource type is added, the script is invoked with the **checkacquire** subcommand by the **drmgr** command. The resource is not actually changed with this subcommand.

When the **drmgr** command invokes the script with the **checkacquire** subcommand, the script determines the resource dependencies of the application, evaluates the effects of resource addition on the application, and indicates whether the resource can be successfully added. For example, there are some MP-unsafe applications. MP-unsafe applications are not tolerant with multiple processors.

Note: Whether or not an application is MP-unsafe is an application design issue, and independent of DLPAR functionality.

If the resource addition request affects the application, the script returns with an exit status of 1 to let the **drmgr** command know to not add the resource.

When the script is invoked with the **checkacquire** subcommand, it takes the following syntax:

```
dr_application_script checkacquire <resource_type>
```

Input for the checkacquire subcommand

The **checkacquire** subcommand takes the following input data:

- ▶ Additional command line arguments

The **checkacquire** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.

- ▶ Name-value pairs from environment variables

The **checkacquire** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **checkacquire** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: The force option should be ignored.

Output for the checkacquire subcommand

The **checkacquire** subcommand produces the following output data:

- ▶ Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

Note: If the script exits with 1 (failure), the **drmgr** command will not add the specified resource and will not invoke subsequent events (**preacquire**, **postacquire**, and **undopreacquire**) against the specified resource (see Table 5-6 on page 166).

- ▶ Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.7.9 The preacquire subcommand

Before the specified resource type is actually acquired, the script is invoked with the **preacquire** subcommand by the **drmgr** command. This is called after the **checkacquire** subcommand.

When the **drmgr** command invokes the script with the **preacquire** subcommand, the script interacts with the application, for example, it informs the application about the resource addition and lets the application acquire the specified resource if it is DLPAR-aware.

Note: Most applications are DLPAR-safe. If your application is DLPAR-safe, but not DLPAR-aware, the script with the **preacquire** subcommand does not have to do any processing.

When the script is invoked with the **preacquire** subcommand, it takes the following syntax:

```
dr_application_script preacquire <resource_type>
```

Input for the preacquire subcommand

The **preacquire** subcommand takes the following input data:

- ▶ Additional command line arguments

The **preacquire** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.

- ▶ Name-value pairs from environment variables

The **preacquire** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.
- The **preacquire** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: When invoking scripts in the **preacquire** phase, the failure of a script does not prevent the **drmgr** command from attempting to add the resource. The theory is that resource addition is safe. It can fail, but the kernel is coded to cleanly add resources, so there is no harm in trying. The return code from each script is remembered so that the **drmgr** command can determine whether it needs to call it back. If a script fails in the **preacquire** phase, it will not be called in the **postacquire** or **undoacquire** phases.

Output for the preacquire subcommand

The **preacquire** subcommand produces the following output data:

- Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the **DR_ERROR** name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

Note: If the script exits with 1 (failure), the **drmgr** command will not perform actual resource removal; however, it will invoke subsequent events (**postacquire** and **undopreacquire**) against the specified resource.

- Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.7.10 The postacquire subcommand

After the specified resource type has been added to the partition, the script is invoked with the **postacquire** subcommand by the **drmgr** command. The script is called after the **preacquire** subcommand.

When the **drmgr** command invokes the script with the **postacquire** subcommand, the script interacts with the application, including any necessary cleanup, for example, restarting or resuming the application if it was quiesced in the **preacquire** subcommand.

The script also takes the appropriate actions if a partial success occurs. A partial success occurs when a subset of the requested number of resources was successfully added. For example, the memory-related environment variables should be checked to determine if all of the requested memory frames were added.

When the script is invoked with the **postacquire** subcommand, it takes the following syntax:

```
dr_application_script postacquire <resource_type>
```

Input for the postacquire subcommand

The **postacquire** subcommand takes the following input data:

- ▶ Additional command line arguments

The **postacquire** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu`, `capacity`, `var_weight`, or `mem`.

- ▶ Name-value pairs from environment variables

The **postacquire** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **postacquire** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-6 on page 166.

Note: The force option should be ignored.

Output for the postacquire subcommand

The **postacquire** subcommand produces the following output data:

- ▶ Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

- Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.7.11 The undopreacquire subcommand

If the **drmgr** command fails to add the specified resource to the partition, it invokes the script with the **undopreacquire** subcommand to recover any necessary cleanup tasks that were done by the **preacquire** subcommand. The script undoes any actions that were taken by the script in the **preacquire** subcommand.

Note: If the specified resource has been added successfully, the **drmgr** command will not invoke the script with the **undopreacquire** subcommand.

When the script is invoked with the **undopreacquire** subcommand, it takes the following syntax:

```
dr_application_script undopreacquire <resource_type>
```

Input for the undopreacquire subcommand

The **undopreacquire** subcommand takes the following input data:

- ▶ Additional command line arguments

The **undopreacquire** subcommand requires one additional command line argument that tells the **drmgr** command which resource type (processor or memory) the script is designed to handle. The valid values are `cpu` or `mem`.

- ▶ Name-value pairs from environment variables

The **undopreacquire** subcommand takes several required input name-value pairs from environment values depending on the resource type that the script is designed to handle:

- If the script is registered to handle processor, see Table 5-4 on page 165.
- If the script is registered to handle memory, see Table 5-5 on page 166.

The **undopreacquire** subcommand also can take an optional input name-value pair from the environment value shown in Table 5-3 on page 164.

Note: The force option should be ignored.

Output for the undopreacquire subcommand

The **undopreacquire** subcommand produces the following output data:

- ▶ Exit value

The script must return an exit value, either 0 (success) or 1 (failure). If the exit value is 1, the `DR_ERROR` name-value pair must be set to describe the reason for failure, and the script must print it to the standard out.

- ▶ Name-value pairs to the standard output stream

Optionally, the script can return the name-value pairs listed in Table 5-6 on page 166.

5.8 How to manage DLPAR scripts

The **drmgr** command must be used to manage DLPAR scripts. The function provided by the **drmgr** command does the following:

- ▶ Lists the registered DLPAR scripts and shows their information.
- ▶ Registers or uninstalls DLPAR scripts in the DLPAR script database.
- ▶ Changes the script install directory path. The default directory is `/usr/lib/dr/scripts/all`.

Note: The **drmgr** command is the only interface to manipulate the DLPAR script database. To use the **drmgr** command, you need the root authority.

The following sections provide typical **drmgr** command usage examples.

5.8.1 List registered DLPAR scripts

To list registered DLPAR scripts and their information, type **drmgr -l**. If no scripts are registered, it returns the following output:

```
# drmgr -l
DR Install Root Directory: /usr/lib/dr/scripts/
Syslog ID: DRMGR
```

Example 5-1 on page 171 shows the example output of **drmgr -l** when DLPAR scripts are already registered.

5.8.2 Register a DLPAR script

To register a DLPAR script, type **drmgr -i script_file_name**. The script is copied into the script install path (the default value is `/usr/lib/dr/scripts/all`) and registered in the DLPAR script database, as shown in Example 5-2.

Note: The fileset `bos.adt.samples` must be installed to enable these functions

Example 5-2 Register a DLPAR script

```
# drmgr -i
DR Install Root Directory: /usr/lib/dr/scripts
Syslog ID: DRMGR
# ls /usr/samples/dr/scripts/IBM_template.sh
/usr/samples/dr/scripts/IBM_template.sh
```

```
# drmgr -i /usr/samples/dr/scripts/IBM_template.sh

DR script file /usr/samples/dr/scripts/IBM_template.sh installed successfully

# drmgr -l
DR Install Root Directory: /usr/lib/dr/scripts
Syslog ID: DRMGR
-----
/usr/lib/dr/scripts/all/IBM_template.sh          AIX DR ksh example script
Vendor:IBM,      Version:1,      Date:10182002
Script Timeout:10,      Admin Override Timeout:0
Resources Supported:
      Resource Name: cpu      Resource Usage: cpu binding for
performance
      Resource Name: mem      Resource Usage: Shared(Pinned) memory
for app XYZ
-----

# ls /usr/lib/dr/scripts/all
IBM_template.sh
```

If the permission mode of the registered script is not appropriate, for example, no executable bits are set, then **drmgr -l** will not list the registered script name, even if the registration has been successfully completed. In this case, set the appropriate permission mode on the script and register it with the overwrite option **-f**, as shown in the following example:

```
# drmgr -f /usr/samples/dr/scripts/IBM_template.sh
```

5.8.3 Uninstall a registered DLPAR script

To uninstall a registered DLPAR script, type **drmgr -u script_file_name**. The script is unregistered from the DLPAR script database, as shown in the following example:

```
# drmgr -u IBM_template.sh

DR script file IBM_template.sh uninstalled successfully

# drmgr -l
DR Install Root Directory: /usr/lib/dr/scripts
Syslog ID: DRMGR
```

The uninstalled script file name is renamed in the script install path, as shown in the following example:

```
# ls -l /usr/lib/dr/scripts/all
total 32
-rw-r--r--  1 bin      bin          13598 Jul 12 14:08 .IBM_template.sh
```

Note: A dot character is added in front of the original file name.

5.8.4 Change the script install path

To change the script install path, type `drmgr -R new_dir`. In the following example, the script install path is changed to the newly created directory `/local/lpar2`¹ from the default path `/usr/lib/dr/scripts`:

```
# drmgr -l
DR Install Root Directory: /usr/lib/dr/scripts
Syslog ID: DRMGR

# mkdir -p /local/`hostname`
# drmgr -R /local/`hostname`
0930-022 DR script ROOT directory set to:/local/lpar2 successfully

# drmgr -l
DR Install Root Directory: /local/lpar2
Syslog ID: DRMGR
```

Note: If you have changed the script install path, scripts that are already registered will not be referenced by the `drmgr` command.

5.8.5 The `drmgr` command line options

Table 5-12 on page 189 lists the `drmgr` command line options and their purpose. For further information about the `drmgr` command, type `man drmgr` on the command line prompt or refer to AIX 5L product documentation, which is available at:

<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/infocenter/base/aix53.htm>

¹ In this example, the `hostname` command returns the host name, `lpar2` on one of our test partitions.

Table 5-12 The `drmgr` command line options

Command option	Brief description	Detailed description
-i script_name Other associated options: [-D install_directory] [-w timeout] [-f]	Installs a DLPAR script to the default or specified directory.	<p>The system administrator should use the -i flag to install a DLPAR script. The script's file name is used as input.</p> <p>Unless the -D flag is used, the scripts are installed into the <code>/usr/lib/dr/scripts/all/</code> directory under the root install directory (see -R flag).</p> <p>Permissions for the DLPAR script are the same as the <code>script_name</code> file.</p> <p>If a script with the same name is already registered, the install will fail with a warning unless the force option is used.</p>
-w timeout	Timeout value in minutes.	This option is used in conjunction with the -i option. The <code>drmgr</code> command will override the timeout value specified by the LPAR script with the new-user defined timeout value.
-f	Forces an override.	During the installation of a script, the -f option can be set to force an override of a duplicate DLPAR script name.
-u script_name Other associated options: [-D host_name]	Uninstalls a DLPAR script.	<p>The system administrator invokes this command to uninstall a DLPAR script. The script file name is provided as an input.</p> <p>The user can specify the directory from where the script should be removed by using the -D option. If no directory is specified, the command will try to remove the script from the all directory under the root directory (see the -R option).</p> <p>If the script is registered using the -D option, it will only be invoked on a system with that host name.</p> <p>If no file is found, the command will return with an error.</p>
-R base_directory_path	Sets the root directory where the DLPAR scripts are installed.	<p>The default value is <code>/usr/lib/dr/scripts/</code>.</p> <p>The installer looks at the all or hosts directory under this root directory. (<code>/usr/lib/dr/scripts/all/</code>).</p>
-d debug_level	Sets the debug level.	This option sets the <code>DR_DEBUG</code> environment variable, which controls the level of debug messages from the DLPAR scripts.

Command option	Brief description	Detailed description
-l	Lists DLPAR scripts.	This option lists the details of all DLPARR scripts currently active on the system.
-b	Rebuilds DLPAR script database.	This option rebuilds the DLPAR script database by parsing through the entire list of DLPAR script install directories.
-S syslog_chan_id_str	Specifies a syslog channel.	This option enables the user to specify a particular channel to which the syslog messages have to be logged from the DLPAR script by the drmgr command.

5.8.6 Sample output examples from a DLPAR script

Although, the syslog facility can be used to record debug information, the debug information for the example DLPAR script was sent to /tmp/IBM_template.sh.dbg for readability reasons.

After registering the DLPAR script written in Korn shell (see Example A-3 on page 245), the following DLPAR operations on the HMC was initiated:

- ▶ 2 GB memory addition
- ▶ 1 GB memory removal
- ▶ 1 CPU addition
- ▶ 2 CPU removal

To perform a DLPAR operation using the graphical user interface on the HMC, refer to 3.1, “Hardware Management Console” on page 58.

Sample output: 2 GB memory addition

Example 5-3 shows the sample output of a 2 GB memory addition DLPAR event. Notice that there are three line blocks for the check, pre, and post phases.

Example 5-3 Sample output: 2 GB memory addition

```
-- start checkacquire phase --
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_FREE_FRAMES=0x30a
DR_MEM_SIZE_COMPLETED=0x0
DR_MEM_SIZE_REQUEST=0x80000000
DR_PINNABLE_FRAMES=0x54a55
DR_TOTAL_FRAMES=0x80000
mem resources: 0x80000000
-- end checkacquire phase --
-- start preacquire phase --
```



```

DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_FREE_FRAMES=0x30a
DR_MEM_SIZE_COMPLETED=0x0
DR_MEM_SIZE_REQUEST=0x80000000
DR_PINNABLE_FRAMES=0x54a55
DR_TOTAL_FRAMES=0x80000
-- end preacquire phase --
-- start undopreacquire phase --
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_FREE_FRAMES=0x30a
DR_MEM_SIZE_COMPLETED=0x800
DR_MEM_SIZE_REQUEST=0x80000000
DR_PINNABLE_FRAMES=0x54a55
DR_TOTAL_FRAMES=0x80000
-- end undopreacquire phase --

```

Before the DLPAR operation, the partition had 2 GB memory assigned, as shown in the following example:

```

# lsattr -El mem0
size      2048 Total amount of physical memory in Mbytes False
goodsize  2048 Amount of usable physical memory in Mbytes False

```

After the completion of the DLPAR operation, the memory size has been increased to 4 GB, as shown in the following example:

```

# lsattr -El mem0
size      4096 Total amount of physical memory in Mbytes False
goodsize  4096 Amount of usable physical memory in Mbytes False

```

Sample output: 1 GB memory removal

Example 5-4 shows the sample output of a 1 GB memory removal DLPAR event. There are three line blocks for the check, pre, and post phases.

Example 5-4 Sample output: 1 GB memory removal

```

-- start checkrelease phase --
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_FREE_FRAMES=0x7e2f5
DR_MEM_SIZE_COMPLETED=0x0
DR_MEM_SIZE_REQUEST=0x40000000
DR_PINNABLE_FRAMES=0xac3f3
DR_TOTAL_FRAMES=0x100000
-- end checkrelease phase --
-- start prerelease phase --
DR_DRMGR_INFO=DRAF architecture Version 1

```

```

DR_FORCE=FALSE
DR_FREE_FRAMES=0x7e2f5
DR_MEM_SIZE_COMPLETED=0x0
DR_MEM_SIZE_REQUEST=0x40000000
DR_PINNABLE_FRAMES=0xac3f3
DR_TOTAL_FRAMES=0x100000
-- end prerelease phase --
-- start postrelease phase --
DR_DRMGR_INFO=DRAE architecture Version 1
DR_FORCE=FALSE
DR_FREE_FRAMES=0x7e2f5
DR_MEM_SIZE_COMPLETED=0x400
DR_MEM_SIZE_REQUEST=0x40000000
DR_PINNABLE_FRAMES=0xac3f3
DR_TOTAL_FRAMES=0x100000
-- end postrelease phase --

```

Before the DLPAR operation, the partition had 4 GB memory assigned, as shown in the following example:

```

# lsattr -El mem0
size      4096 Total amount of physical memory in Mbytes  False
goodsize  4096 Amount of usable physical memory in Mbytes False

```

After the completion of the DLPAR operation, the memory size has been decreased to 3 GB, as shown in the following example:

```

# lsattr -El mem0
size      3072 Total amount of physical memory in Mbytes  False
goodsize  3072 Amount of usable physical memory in Mbytes False

```

Sample output: 1 CPU addition

Example 5-5 shows the sample output of a 1 CPU addition DLPAR event. You will see there are three line blocks for the check, pre, and post phases for the CPU ID 2.

Example 5-5 Sample output: 1 CPU addition

```

-- start checkacquire phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAE architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=2
cpu resources: logical 2, bind 2
-- end checkacquire phase --
-- start preacquire phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAE architecture Version 1
DR_FORCE=FALSE

```

```
DR_LCPUID=2
-- end preacquire phase --
-- start undopreacquire phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=2
-- end undopreacquire phase --
```

Before the DLPAR operation, the partition had two processors assigned, as shown in the following example:

```
# lsdev -Cc processor -S Available
proc6 Available 00-06 Processor
proc7 Available 00-07 Processor
```

After the completion of the DLPAR operation, the number of active processors has been increased to three, as shown in the following example:

```
# lsdev -Cc processor -S Available
proc6 Available 00-06 Processor
proc20 Available 00-20 Processor
proc7 Available 00-07 Processor
```

Sample output: 2 CPU removal

Example 5-6 shows the sample output of a 2 CPU removal DLPAR event. There are three line blocks for the check, pre, and post phases for each CPU (ID 2 and ID3).

Example 5-6 Sample output: 2 CPU removal

```
-- start checkrelease phase --
DR_BCPUID=3
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=3
-- end checkrelease phase --
-- start prerelease phase --
DR_BCPUID=3
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=3
-- end prerelease phase --
-- start postrelease phase --
DR_BCPUID=3
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=3
-- end postrelease phase --
```

```

-- start checkrelease phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=2
-- end checkrelease phase --
-- start prerelease phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=2
-- end prerelease phase --
-- start postrelease phase --
DR_BCPUID=2
DR_DRMGR_INFO=DRAF architecture Version 1
DR_FORCE=FALSE
DR_LCPUID=2
-- end postrelease phase --

```

Before the DLPAR operation, the partition had four processors assigned, as shown in the following example:

```

# lsdev -Cc processor -S Available
proc6 Available 00-06 Processor
proc20 Available 00-20 Processor
proc7 Available 00-07 Processor
proc21 Available 00-21 Processor

```

After the completion of the DLPAR operation, the number of active processes has been decreased to two, as shown in the following example:

```

# lsdev -Cc processor -S Available
proc6 Available 00-06 Processor
proc7 Available 00-07 Processor

```

5.9 API-based DLPAR event handling

The AIX 5L Version 5.3 operating system includes further enhancements to the `dr_reconfig()` system call. The improvements are intended to exploit the POWER5 processor and virtualization features when building or customizing applications that respond to DLPAR events. Applications wanting to utilize all of the new function will need to be modified to remain DLPAR-aware. Because DLPAR-safe applications should not fail when a DLPAR event occurs, they require no changes.

To properly write an application to recognize a DLPAR event, the application will register a signal handler that calls `dr_reconfig()`. When a DLPAR event occurs,

the application receives SIGRECONFIG signals from the kernel in order to notify the DLPAR event. The signal is sent twice (check and pre phases) before the actual resource change (addition or removal), and sent once (post phase) after the resource change.

Note: The SIGRECONFIG signal is also sent (along with the SIGCPUFAIL signal for backward compatibility) in the case of a CPU Guard event. Therefore, this API-based method can also be utilized by CPU Guard-aware applications.

In the latest release of DLPAR support, DLPAR events for I/O slots do not notify applications using the `dr_config()` system call.

5.9.1 The `dr_reconfig` system call

The `dr_reconfig()` system call is provided to query the information of the current DLPAR event. The system call must be called from a registered signal handler in order for the application be notified from the kernel when a DLPAR event occurs. The `sigaction()` system call is used to register a signal handler.

To use `dr_reconfig()` in your C language application, you need to add the following compiler directive line that instructs the preprocessor to include the `/usr/include/sys/dr.h` file:

```
#include <sys/dr.h>
```

Example 5-7 The `dr_reconfig` system call usage

```
int dr_reconfig(int flags, dr_info_t *info);
```

0 is returned for success; otherwise,
-1 is returned, and the `errno` is set to the appropriate value.

The `dr_reconfig()` system call takes two parameters. The flags determine what the system call does. The `info` parameter is a structure that contains DLPAR-specific data that the signal handler uses to process DLPAR events accordingly. Table 5-13 on page 196 provides the supported flags.

Table 5-13 The dr_reconfig flag parameters²

Flags	Description
DR_QUERY	This flag identifies the current DLPAR event. It also identifies any actions, if any, that the application should take to comply with the current DLPAR event. Any pertinent information is returned in the second parameter.
DR_EVENT_FAIL	This flag fails the current DLPAR event. It requires root authority.
DR_RECONFIG_DONE	This flag is used in conjunction with the DR_QUERY flag. The application notifies the kernel that the actions it took to comply with the current DLPAR request are now complete. The dr_info structure identifying the DLPAR request that was returned earlier is passed as an input parameter.

The other parameter is a pointer to a structure that hold DLPAR-specific information. The signal handler must allocate space for the dr_info_t data structure. The AIX kernel populates this data structure and return it to the signal handler. In AIX 5L Version 5.3, additional handlers have been included. They are outlined in Example 5-8, which shows the definition of the dr_info structure.

Example 5-8 The dr_reconfig info parameter

```
typedef struct dr_info {
/* The following fields are filled out for cpu based requests */
unsigned int    ent_cap : 1; // entitled capacity change request
unsigned int    var_wgt : 1; // variable weight change request
unsigned int    splpar_capable : 1; // partition is Shared Processor Partition
capable
unsigned int    splpar_shared : 1; // shared partition (1), dedicated (0)
unsigned int    splpar_capped : 1; // shared partition is capped
unsigned int    cap_constrained : 1; // capacity is constrained by PHYP
uint64_t        capacity; // the current entitled capacity or
                        // variable capacity weight value
                        // depending on the bit fields
                        // ent_cap and var_wgt.
int delta_cap; // delta entitled capacity or variable
                // weight capacity that is to be added
                // or removed.
} dr_info_t;
```

² The DR_RECONFIG_DONE flag is available on AIX 5L Version 5.2 plus 5200-01 Recommended Maintenance Level and later.

The bindproc and bindpset bits are only set if the request is to remove a processor. If the bindproc is set, then the process has a bindprocessor() attachment that must be resolved before the operation is allowed. If the bindpset bit is set, the application has processor set attachment, which can be lifted by calling the appropriate processor set interface.

The plock and pshm bits are only set if the DLPAR request is to remove memory and the process has plock() memory or is attached to a pinned shared memory segment. If the plock bit is set, the application calls plock() to unpin itself. If the pshm bit is set, the application detaches its pinned memory segments. The memory remove request might succeed, even if the pshm bit is set, as long as there is enough pinnable memory in the partition. Therefore, an action might not be required for the pshm bit to be set, but it is strongly recommended. The sys_pinnable_frames field provides the necessary information if the system has enough excess pinnable memory.

Programming implications of CPU DLPAR events

At boot time, processors are configured in the kernel. In AIX 5L, a processor is identified by three different identifications, namely:

- ▶ The physical CPU ID, which is derived from the Open Firmware device tree and used to communicate with RTAS.
- ▶ The logical CPU ID, which is a ppda-based³ index of online and offline processors.
- ▶ The bind CPU ID, which is the index of online processors.

The logical and bind CPU IDs are consecutive and have no holes in the numbering. No guarantee is given across boots that the processors will be configured in the same order or even that the same processors will be used in a partitioned environment at all.

At system startup, the logical and bind CPU IDs are both consecutive and have no holes in the numbering; however, DLPAR operations can remove a processor from the middle of the logical CPU list. The bind CPU IDs remain consecutive because they refer only to online processors, so the kernel has to explicitly map these IDs to logical CPU IDs (containing online and offline CPU IDs).

The range of logical CPU IDs is defined to be 0 to $M-1$, where M is the maximum number of processors that can be activated within the partition. M is derived from the Open Firmware device tree. The logical CPU IDs name both online and offline processors. The rset⁴ APIs are predicated on the use of logical CPU IDs.

³ Per processor description area.

⁴ Resource set.

The range of bind CPU IDs is defined to be 0 to $N-1$; however, N is the current number of online processors. The value of N changes as processors are added and removed from the system by either DLPAR or CPU Guard. In general, new processors are always added to the N th position. Bind CPU IDs are used by the system call `bindprocessor` and by the kernel service `switch_cpu`.

The number of potential processors can be determined by:

- ▶ `_system_configuration.max_ncpus`
- ▶ `_system_configuration.original_ncpus`
- ▶ `var.v_ncpus_cfg`
- ▶ `sysconf(_SC_NPROCESSORS_CONF)`

The number of online processors can be determined by:

- ▶ `_system_configuration.ncpus`
- ▶ `var.v_ncpus`
- ▶ `sysconf(_SC_NPROCESSORS_ONLN)`

The `_system_configuration` structure is defined in the `/usr/include/sys/systemcfg.h` header file, and those members can be accessed from your application, as shown in the following code fraction example:

```
#include <sys/systemcfg.h>
printf("_system_configuration.original_ncpus=%d\n"
      , _system_configuration.original_ncpus);
```

The `var` structure is defined in the `/usr/include/sys/var.h` header file and populated by the `sysconfig` system call. The following code fraction example demonstrates how to retrieve `var.v_ncpus`:

```
#include <sys/types.h>
#include <sys/sysconfig.h>
#include <sys/var.h>
struct var myvar;
rc = sysconfig(SYS_GETPARMS, &myvar, sizeof(struct var));
if (rc == 0)
    printf("var.v_ncpus = %d\n", myvar.v_ncpus);
```

The number of online processors can also be determined from the command line. AIX provides the following commands:

- ▶ **`bindprocessor -q`**
- ▶ **`lsrset -a`**

As previously mentioned, AIX supports two programming models for processors: the `bindprocessor` model that is based on bind CPU IDs and the `rset` API model that is based on logical CPU IDs. Whenever a program implements any of these programming models, it should be DLPAR-aware.

The following new interfaces (system calls and kernel services) are provided to query bind and logical CPU IDs and the mapping between them:

- ▶ mycpu(): Returns bind CPU ID of the process
- ▶ my_lcpu(): Returns bind CPU ID of the process
- ▶ b2lcpu(): Returns the bind to logical CPU ID mapping
- ▶ l2bcpu(): Returns the logical to bind CPU ID mapping

5.9.2 A sample code using the dr_reconfig system call

A sample application was written in the C language using the dr_reconfig() system call (see Example A-4 on page 257). Because the source code is long, an excerpt of the most important part is provided with annotations from the example.

Basically, this application does nothing voluntary, except for the signal handler registration. It just does the busy loop in the while loop in main and waits until the SIGRECONFIG signal is delivered. You must implement your application logic in the while loop, specified by the comment `Your application logic goes here.`

The behavior of the application is briefly explained in the following:

1. Register a signal handler, dr_func(), in main (indicated as #A in the comment). The signal handler is registered in order to react to the SIGRECONFIG signal when it is delivered to the application process.
2. Wait in the busy loop in main (#B) until the SIGRECONFIG signal is sent:
3. After the SIGRECONFIG signal is delivered by the kernel, the signal handler, dr_func(), is invoked.
4. The handler calls dr_reconfig() in order to query the dr_info structure data. The dr_info structure is used to determine what DLPAR operation triggers this signal (#C).

```
if ((rc = sigaction(SIGRECONFIG, &sigact, &sigact_save)) != 0) { /* #A */
```

```
while (1) { /* #B */
    ;
    /* your application logic goes here. */
}
```

```
l_rc = dr_reconfig(DR_QUERY, &dr_info); /* #C */
```

Note: You must include the following preprocessor directive line to use the dr_reconfig() system call:

```
#include <sys/dr.h>
```

5. The handler parses the `dr_info` structure to determine the DLPAR operation type:

- If the `dr_info.add` member is set, this signal is triggered by a DLPAR resource addition request (#D):

```
if (dr_info.add) { /* #D */
```

- If the `dr_info.rem` member is set, this signal is triggered by a DLPAR resource removal request (#E):

```
if (dr_info.rem) { /* #E */
```

6. The handler again parses the `dr_info` structure to determine the DLPAR resource type:

- If the `dr_info.cpu` member is set, this signal is triggered by a DLPAR CPU resource addition or removal request (#F):

```
if (dr_info.cpu) { /* #F */
```

- If the `dr_info.mem` member is set, this signal is triggered by a DLPAR memory resource addition or removal request (#G):

```
} else if (dr_info.mem) { /* #G */
```

7. Invoke the corresponding function based on the information determined:

- If the requested DLPAR resource type is CPU, call the function pointer stored in the `l_currentPhase->cpu_ptr` array (#H):

```
l_rc = l_currentPhase->cpu_ptr(); /* #H */
```

- If the requested DLPAR resource type is memory, call the function pointer stored in the `l_currentPhase->mem_ptr` array (#I):

```
l_rc = l_currentPhase->mem_ptr(); /* #I */
```

Note: You must modify the functions included in the `definedPhase` array (#J) by adding your own logic in order to react against DLPAR operation phases. The comment `Perform actions here` specifies the location where you modify the functions.

5.9.3 Sample output examples from a DLPAR-aware application

Although, the `syslog` facility can be used to record debug information, the example application debug information was sent to `/tmp/dr_api_template.C.dbg` for readability reasons.

After compiling the C source code (see Example A-4 on page 257), the application was run and initiated several DLPAR operations on the HMC.

The following several examples exhibit the internal behaviors of the following DLPAR operations:

- ▶ 1 GB memory addition
- ▶ 1 GB memory removal
- ▶ 2 CPU addition
- ▶ 1 CPU removal

To perform a DLPAR operation using the graphical user interface on the HMC, refer to 3.1, “Hardware Management Console” on page 58.

Sample output: 1 GB memory addition

Example 5-9 shows the sample output of a 1 GB memory addition DLPAR event. There are three line blocks for the check, pre, and post phases.

Example 5-9 Sample output: 1 GB memory addition

```
---Start of Signal Handler---
An add request for
  ** check phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 2147483648
  number of free frames in system = 29434
  number of pinnable frams in system = 339916
  total number of frames in system = 524288
*****Entered CheckedAcquire_mem*****
---end of signal handler---

---Start of Signal Handler---
An add request for
  ** pre phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 2147483648
  number of free frames in system = 29434
  number of pinnable frams in system = 339916
  total number of frames in system = 524288
*****Entered PreeAcquire_mem*****
---end of signal handler---

---Start of Signal Handler---
An add request for
  ** post phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 2147483648
  number of free frames in system = 284761
```

```
number of pinnable frams in system = 516763
total number of frames in system = 786432
*****Entered PostAcquire_mem*****
---end of signal handler---
```

Sample output: 1 GB memory removal

Example 5-10 shows the sample output of a 1 GB memory removal DLPAR event. There are three line blocks for the check, pre, and post phases.

Example 5-10 Sample output: 1 GB memory removal

```
---Start of Signal Handler---
A remove request for
  ** check phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 3221225472
  number of free frames in system = 284771
  number of pinnable frams in system = 516763
  total number of frames in system = 786432
*****Entered CheckeRelease_mem*****
---end of signal handler---

---Start of Signal Handler---
A remove request for
  ** pre phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 3221225472
  number of free frames in system = 284770
  number of pinnable frams in system = 516763
  total number of frames in system = 786432
*****Entered PreRelease_mem*****
---end of signal handler---

---Start of Signal Handler---
A remove request for
  ** post phase **
Resource is Memory.
  requested memory size (in bytes) = 1073741824
  system memory size = 3221225472
  number of free frames in system = 29043
  number of pinnable frams in system = 339916
  total number of frames in system = 524288
*****Entered PostReleasee_mem*****
---end of signal handler---
```

Sample output: 2 CPU addition

Example 5-11 shows the sample output of a 2 CPU addition DLPAR event. There are three line blocks for the check, pre, and post phases for each processor (CPU ID 2 or 3).

Example 5-11 Sample output: 2 CPU addition

```
---Start of Signal Handler---
An add request for
    ** check phase **
Resource is CPU .
    logical CPU ID = 2
    Bind CPU ID = 2
*****Entered CheckedAcquire_cpu*****
---end of signal handler---

---Start of Signal Handler---
An add request for
    ** pre phase **
Resource is CPU .
    logical CPU ID = 2
    Bind CPU ID = 2
*****Entered PreAcquire_cpu*****
---end of signal handler---

---Start of Signal Handler---
An add request for
    ** post phase **
Resource is CPU .
    logical CPU ID = 2
    Bind CPU ID = 2
*****Entered PostAcquire_cpu*****
---end of signal handler---

---Start of Signal Handler---
An add request for
    ** check phase **
Resource is CPU .
    logical CPU ID = 3
    Bind CPU ID = 3
*****Entered CheckedAcquire_cpu*****
---end of signal handler---

---Start of Signal Handler---
An add request for
    ** pre phase **
Resource is CPU .
    logical CPU ID = 3
    Bind CPU ID = 3
```

```

*****Entered PreAcquire_cpu*****
---end of signal handler---

---Start of Signal Handler---
An add request for
  ** post phase **
Resource is CPU .
  logical CPU ID = 3
  Bind CPU ID = 3
*****Entered PostAcquire_cpu*****
---end of signal handler---

```

Sample output: 1 CPU removal

Example 5-12 shows the sample output of a 1 CPU removal DLPAR event. There are line three blocks for the check, pre, and post phases for the CPU ID 3.

Example 5-12 Sample output: 1 CPU removal

```

---Start of Signal Handler---
A remove request for
  ** check phase **
Resource is CPU .
  logical CPU ID = 3
  Bind CPU ID = 3
*****Entered CheckRelease_cpu*****
---end of signal handler---

---Start of Signal Handler---
A remove request for
  ** pre phase **
Resource is CPU .
  logical CPU ID = 3
  Bind CPU ID = 3
*****Entered PreRelease_cpu*****
---end of signal handler---

---Start of Signal Handler---
A remove request for
  ** post phase **
Resource is CPU .
  logical CPU ID = 3
  Bind CPU ID = 3
*****Entered PostRelease_cpu*****
---end of signal handler---

```

5.9.4 DLPAR-aware kernel extensions

Like applications, most kernel extensions are DLPAR-safe by default. However, some are sensitive to the system configuration and might need to be registered with the kernel in order to be notified of DLPAR events.

To register and unregister from the kernel in order to be notified in the case of DLPAR events, the following kernel services are available:

- ▶ `reconfig_register`
- ▶ `reconfig_unregister`
- ▶ `reconfig_complete`

The `reconfig_register` and `reconfig_unregister` services have had events added to support the following shared processors functions:

- ▶ Capacity addition and removal
- ▶ Virtual processor add and remove is supported by pre-existing CPU add and remove

5.10 Error handling of DLPAR operations

Knowing what errors the `drmgr` command can return is fundamental to creating a comprehensive DLPAR script or DLPAR-aware application. This section covers the methods AIX provides to help perform error analysis on failed DLPAR operations. It also discusses some actions that should be taken when an error occurs.

5.10.1 Possible causes of DLPAR operation failures

A DLPAR operation request can fail for various reasons. The most common of these is that the resource is busy, or that there are not enough system resources currently available to complete the request. In these cases, the resource is left in a normal state as though the DLPAR event never happened.

The following are possible causes of DLPAR operation failures:

- ▶ The primary cause of processor removal failure is processor bindings. The operating system cannot ignore processor bindings and carry on DLPAR operations or applications might not continue to operate properly. To ensure that this does not occur, release the binding, establish a new one, or terminate the application. The processors that are impacted is a function of the type of binding that is used.

- ▶ The primary cause of memory removal failure is that there is not enough pinned memory available in the system to complete the request. This is a system-level issue and is not necessarily the result of a specific application. If a page in the memory region to be removed has a pinned page, its contents must be migrated to another pinned page, while automatically maintaining its virtual to physical mappings. The failure occurs when there is not enough pinnable memory in the system to accommodate the migration of the pinned data in the region that is being removed. To ensure that this does not occur, lower the level of pinned memory in the system. This can be accomplished by destroying pinned shared memory segments, terminating programs that implement the plock system call, or removing the plock on the program.
- ▶ The primary cause of PCI slot removal failure is that the adapters in the slot are busy. Note that device dependencies are not tracked. For example, the device dependency might extend from a slot to one of the following: an adapter, a device, a volume group, a logical volume, a file system, or a file. In this case, resolve the dependencies manually by stopping the relevant applications, unmounting file systems, varying off volume groups, and unconfiguring the device drivers associated with the adapters in the target slot.

If an error occurs in a DLPAR operation, the error message dialog box shown in Figure 5-9 appears on the HMC.

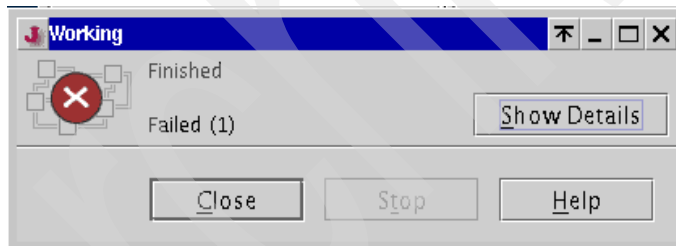


Figure 5-9 DLPAR operation failed message

The HMC also displays the information message dialog box, as shown in Figure 5-10 on page 207.

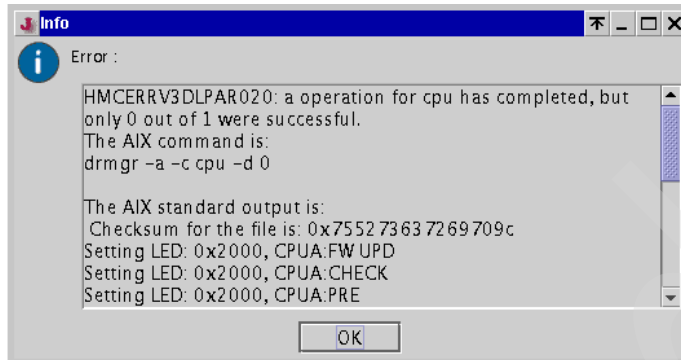


Figure 5-10 DLPAR operation failure detailed information

Note: If the registered DLPAR scripts have bugs, they are also the cause of failure. You must carefully code the scripts and test them on a test partition before the deployment on the production partition.

5.10.2 Error analysis facilities

AIX provides the following facilities to help isolate DLPAR operation failures:

- ▶ The syslog facility
- ▶ AIX system trace facility
- ▶ AIX error log facility
- ▶ Kernel debugger

You can also use these facilities if a script or DLPAR-aware application fails. Moreover, by learning how to use these facilities, you can modify your programs to handle some of the possible errors automatically.

The syslog facility

The syslog facility is another useful tool to help isolate DLPAR-related errors. You can use it to keep a record of the progress of DLPAR events. The syslog entries come with a time stamp to indicate when all the DLPAR events occurred.

On AIX, the syslog facility is not enabled by default. To enable recording DLPAR events using syslog, do the following:

1. Edit the `/etc/syslog.conf` file as the root user.
2. Add the required syslog entries to the end of the file.

For example, add the following:

```
*.debug /var/adm/syslog.log rotate size 10k
```

This directive line instructs the syslog facility to log all messages of priority debug (LOG_DEBUG) and above to the /var/adm/syslog.log file. The /var/adm/syslog.log file is automatically rotated to limit the maximum file size to 10 KB.

3. Create the file explicitly:

```
# touch /var/adm/syslog.log
```

4. Restart the syslogd subsystem:

```
# stopsrc -s syslogd
# startsrc -s syslogd
```

In Appendix B, “Dynamic logical partitioning output samples” on page 273, the following syslog output examples are included:

- ▶ Sample syslog output for a processor addition request
- ▶ Sample syslog output for a memory addition request
- ▶ Sample syslog output for a memory removal request

When you register your DLPAR scripts, if you explicitly specify a channel ID string other than the default value DRMGR by using **drmgr -S**, you can quickly search the corresponding information that is produced by your DLPAR scripts. The default channel ID, DRMGR, is shown in several syslog output examples.

AIX system trace facility

The AIX system trace facility is a tool that can trace many kernel internal activities by specifying trace hook IDs. In case of processor- and memory-related DLPAR events, the trace hook ID is 38F. After capturing the trace of DLPAR events, you can generate a trace report in order to examine the results.

To use AIX system trace facility in order to capture DLPAR events, do the following as the root user:

1. Start the trace:

```
# trace -a -j 38f
```

2. Perform the desired DLPAR operations.

3. Stop the trace:

```
# trcstop
```

4. Analyze the trace:

```
# trcrpt
```

You can also use SMIT to do the same activities (you need the root authority):

1. Invoke **smit** and select the following panels, and then press Enter:

```
Problem Determination
  Trace
    START Trace
```

2. Type 38F in the ADDITIONAL event IDs to trace field, as shown in Example 5-13, and then press Enter.

Example 5-13 START Trace panel

START Trace

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

	[Entry Fields]	
EVENT GROUPS to trace	<input type="checkbox"/>	+
ADDITIONAL event IDs to trace	[38F]	+
Event Groups to EXCLUDE from trace	<input type="checkbox"/>	+
Event IDs to EXCLUDE from trace	<input type="checkbox"/>	+
Trace MODE	[alternate]	+
STOP when log file full?	[no]	+
LOG FILE	[/var/adm/ras/trcfile]	
SAVE PREVIOUS log file?	[no]	+
Omit PS/NM/LOCK HEADER to log file?	[yes]	+
Omit DATE-SYSTEM HEADER to log file?	[no]	+
Run in INTERACTIVE mode?	[no]	+
Trace BUFFER SIZE in bytes	[131072]	#
LOG FILE SIZE in bytes	[1310720]	#
Buffer Allocation	[automatic]	+

3. Perform the desired DLPAR operations.

4. Invoke **smit** and select the following panels, and then press Enter:

```
Problem Determination
  Trace
    STOP Trace
```

5. Invoke **smit**, select the following panels, select **1 filename (defaults stdout)**, and then press Enter:

```
Problem Determination
  Trace
    Generate a Trace Report
```

6. Select the following values in the smit panel shown in Example 5-14, and then press Enter:
 Show PROCESS IDs for each event? yes
 Show THREAD IDs for each event? yes

Example 5-14 Generate a Trace Report panel

Generate a Trace Report

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

	[Entry Fields]	
Show exec PATHNAMES for each event?	[yes]	+
Show PROCESS IDs for each event?	[yes]	+
Show THREAD IDs for each event?	[yes]	+
Show CURRENT SYSTEM CALL for each event?	[yes]	+
Time CALCULATIONS for report	[elapsed only]	+
Event Groups to INCLUDE in report	<input type="checkbox"/>	+
IDs of events to INCLUDE in report	<input type="checkbox"/>	+X
Event Groups to EXCLUDE from report	<input type="checkbox"/>	+
ID's of events to EXCLUDE from report	<input type="checkbox"/>	+X
STARTING time	<input type="checkbox"/>	
ENDING time	<input type="checkbox"/>	
LOG FILE to create report from	[/var/adm/ras/trcfile]	
FILE NAME for trace report (default is stdout)	<input type="checkbox"/>	

For more information, see Appendix B, “Dynamic logical partitioning output samples” on page 273.

AIX error log facility

The **drmgr** command can generate error log messages in the few cases involving kernel, kernel extensions, or platform failures that have been caused by a DLPAR event. Table 5-14 on page 211 shows a list of the possible errors that could be found in the system error log.

Table 5-14 AIX error logs generated by DLPAR operations

Error log entry	Description
DR_SCRIPT_MSG	Application script error or related messages, or both. Entry includes failing script name and DLPAR phase where the error occurred.
DR_RECONFIG_HANDLER_MSG	Kernel extension reconfiguration handler error. Entry includes failing handler's registration name.
DR_MEM_UNSAFE_USE	Non-DLPAR aware kernel extension's use of physical memory is not valid. The result is that the affected memory is not available for DLPAR removal. The entry includes: <ul style="list-style-type: none"> ▶ The affected logical memory address ▶ An address corresponding to the kernel extension's load module ▶ The kernel extension load module's path name
DR_DMA_MEM_MIGRATE_FAIL	Memory removal failure due to DMA activity. The affected LMB had active DMA mappings that could not be migrated by the platform. The entry includes: <ul style="list-style-type: none"> ▶ The logical memory address within the LMB ▶ Hypervisor migration return code ▶ Logical bus number of the slot owning the DMA mapping ▶ The DMA address
DR_DMA_MEM_MAPPER_FAIL	Memory removal failure due to a kernel extension responsible for controlling DMA mappings error. The entry includes: <ul style="list-style-type: none"> ▶ DMA mapper handler return code ▶ An address corresponding to the DMA mapper's kernel extension load module ▶ The DMA mapper's kernel extension load module's path name

Kernel debugger

The AIX kernel debugger (KDB) helps isolate DLPAR operation errors. KDB can be especially useful to diagnose errors found in the kernel extensions. For further information about the use of KDB, refer to *AIX 5L Version 5.3 Technical Reference: Kernel and Subsystems*, available at:

<http://techsupport.services.ibm.com/server/library>

5.10.3 AIX error log messages when DLPAR operations fail

Several different AIX error messages can be generated when a DLPAR event failure occurs. These error messages and the action that should be taken are listed. The following tables show all the AIX error messages in relation to DLPAR operations.

Table 5-15 indicates general error messages that can be displayed when a DLPAR event failure occurs and the recommended actions to take.

Table 5-15 General AIX error messages

Error message	Recommended action
You must have root authority to run this command.	Log in as the root user.
Failed to set the ODM data.	Contact an IBM service representative.
Consult AIX error log for more information.	Open the AIX error log and look for error log entries with the DR_ prefix.
Resource identifier out of range.	Consult AIX syslog and HMC logs.

Table 5-16 describes the possible errors that can be generated by the **drmgr** command.

Table 5-16 *drmgr*-specific AIX error messages

Error message	Recommended action
Error building the DLPAR script information.	Check system resources, such as free space in the /var file system. If the problem persists, contact an IBM service representative.
Aborting DLPAR operation due to Check Phase failure.	Examine the AIX syslog. Contact the script/application owner.

Error message	Recommended action
Error: Specified DLPAR script file already exists in the destination directory.	Examine the AIX syslog. Contact the script/application owner to change the script name. Use the force flag (drmgr -f) to overwrite the pre-existing script.
Error: Specified DLPAR script file does not exist in directory.	File could not be found for uninstallation. Check the file name specified.
The DLPAR operation is not supported.	The machine or configuration does not support that operation. Upgrade the system firmware or operating system software, or both.
Invalid parameter.	Contact an IBM service representative.

While a DLPAR operation is taking place, an error can occur. Table 5-17 indicates some of these error messages caused by AIX during a DLPAR operation.

Table 5-17 DLPAR operation-specific AIX error messages

Error message	Recommended action
DLPAR operation failed because of timeout.	Increase the timeout value, or try again later. Also, try the DLPAR operation without a timeout specified.
DLPAR operation failed. Kernel busy with another DLPAR operation.	Only perform one DLPAR operation at a time. Try again later.
DLPAR operation failed due to kernel error.	Examine the AIX syslog or contact an IBM service representative, or both.
The DLPAR operation could not be supported by one or more kernel extensions.	Find the corresponding AIX error log entry DR_RECONFIG_HANDLER_MSG and contact the kernel extension owner.
DLPAR operation failed since resource is already online.	Examine the AIX syslog and HMC log.
DLPAR operation timed out.	Increase the timeout, or try again later. Also, try initiating the DLPAR operation without a timeout specified.

Finally, there are several DLPAR errors resulting from resource events. Table 5-18 displays these types of errors.

Table 5-18 DLPAR resource-specific AIX error messages

Error message	Recommended action
The specified connector type is invalid, or there is no dynamic reconfiguration support for connectors of this type on this system.	Examine the AIX syslog and HMC log.
Insufficient resource to complete operation.	Try again later. Free up resources and try again.
CPU could not be started.	Examine the AIX syslog and HMC log.
Memory could not be released. DLPAR operation failed since a kernel extension controlling DMA mappings could not support the operation.	Examine the AIX error log and look for the DR_DMA_MAPPER_FAIL entry. The logical memory block or address of the message should correspond to the logical memory address in the error log entry. The LR value in the error log is an address within the failing kernel extension. The Module Name in the error log is the path name of the kernel extension load module. Unconfigure the kernel extension and retry. Contact the kernel extension owner.
Resource could not be found for the DLPAR operation.	Examine the AIX syslog and HMC log.
Resource is busy and cannot be released.	Examine the AIX syslog. Quiesce activity using the resource and try again.
Memory in use by a non DLPAR-safe kernel extension and hence cannot be released.	Examine the AIX error log and look for the DR_MEM_UNSAFE_USE entry. The logical memory block or address in the message should correspond to the logical memory address in the error log. The LR value in the error log is an address within the owning kernel extension. The Module Name in the error log is the path name of the kernel extension load module. Unconfigure the kernel extension and retry. Contact the kernel extension owner.

Error message	Recommended action
Memory could not be released because system does not contain enough resources for optimal pinned memory and large page memory ratios.	Reduce pinned or large page memory requirements, or both, and try again.

The POWER Hypervisor

The technology behind the shared processor on @server p5 systems is provided by a piece of firmware known as the POWER Hypervisor (referred to in this document simply as “hypervisor”). The enhanced layered code structure of the hypervisor resides in flash memory on the Service Processor. This firmware performs the initialization and configuration of the POWER5 processor, as well as the virtualization support required to run up to 140 partitions concurrently on the IBM @server p5 servers.

6.1 Introduction

The hypervisor supports many advanced functions when compared to the previous version of the hypervisor, including sharing of processors, virtual I/O, high-speed communications between partitions using Virtual LAN, concurrent maintenance and allows for multiple operating systems to run on the single system. AIX 5L, Linux, and i5/OS are supported.

With support for dynamic resource movement across multiple environments, customers can move processors, memory and I/O between partitions on the system as they move workloads between the three environments.

The hypervisor is the underlying control mechanism that resides below the operating systems. The hypervisor owns all system resources and creates partitions by allocating these resources and sharing them.

The layers above the hypervisor are different for each supported operating system. The @server p5 systems use the new hypervisor for support of the Micro-Partitioning technology model. The POWER4 Hypervisor processor-based systems worked on a demand basis, as the result of machine interrupts and callbacks to the operating system. The new hypervisor operates continuously in the background.

For the AIX 5L and Linux operating systems, the layer above the hypervisor are similar but the contents are characterized by each operating system. The layers of code supporting AIX 5L and Linux consist of System Firmware and Run-Time Abstraction Services (RTAS).

System firmware is composed of low level firmware that is code that performs server unique I/O configurations and the Open Firmware which contains the boot time drivers, boot manager, and the device drivers required to initialize the PCI adapters and attached devices. RTAS consist of code that supplies platform dependent accesses and can be called from the operating system. These calls are passed to the hypervisor that handles all I/O interrupts.

The role of RTAS versus Open Firmware is very important to understand. Open Firmware and RTAS are both platform-specific firmware and both are tailored by the platform developer to manipulate the specific platform hardware. However, RTAS is intended to present to access platform hardware features on behalf of the operating system, while Open Firmware need not be present when the operating system, is running. This frees Open Firmware's memory to be used by applications. RTAS is small enough to painlessly coexist with the operating system and applications.

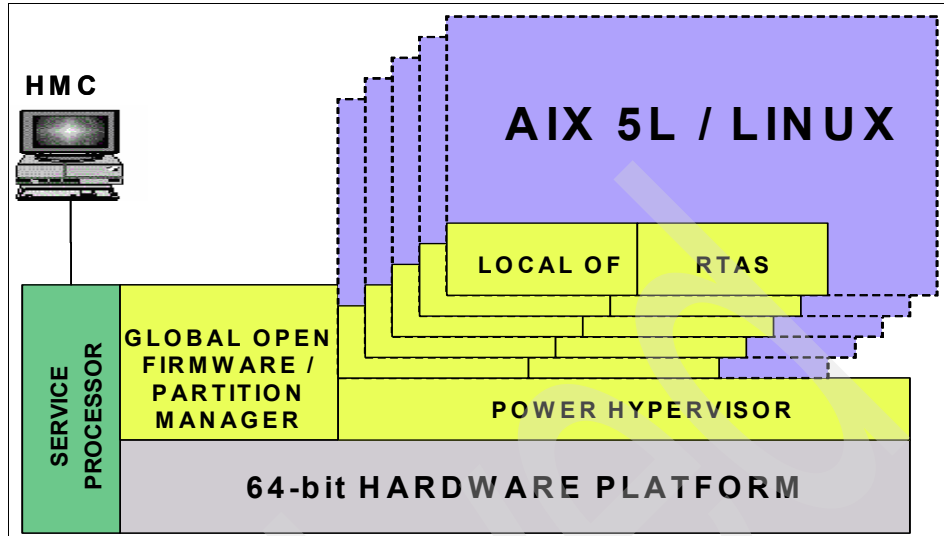


Figure 6-1 POWER Hypervisor on AIX 5L and Linux

For i5/OS, Technology Independent Machine Interface and the layers above the hypervisor are still in place. System Licensed Internal Code, however, is changed and enabled for interfacing with the hypervisor. The hypervisor code is based on the iSeries Partition Licensed Internal Code code that is enhanced for use with the IBM @server® i5 hardware and that is now part of the hypervisor.

Attention: All @server p5 based servers require the use of the hypervisor. A system administrator can configure the system as a single, partition that includes all the resources on the system, but cannot run in SMP mode without the hypervisor as they could with POWER4 systems. A @server p5 based server is always partition capable.

6.2 Hypervisor support

The POWER5 processor supports a special set of instructions which are exclusively used by the hypervisor. If an operating system instance in a partition requires access to hardware, it first invokes the hypervisor by using hypervisor calls. The hypervisor allows privileged access to the operating system for dedicated hardware facilities and includes protection for those facilities in the processor and memory locations.

The introduction of shared processors has not fundamentally changed with the introduction of Virtualization and Micro-Partitioning technology. New virtual processor objects and hypervisor calls have been added to support shared processor partitions. Actually, the existing physical processor objects have just been refined, so as not to include physical characteristics of the processor, since there is not fixed relationship between a virtual processor and the physical processor that actualizes it. These new hypervisor calls are intended to support the scheduling heuristic of minimizing idle time.

The hypervisor is entered by the way of three interrupts:

- ▶ **System Reset Interrupt**

The hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the Processor Identification Register (PIR). The PIR is a read only register. During power-on reset, PIR is set to a unique value for each processor in a multi-processor system.

- ▶ **Machine Check Interrupt**

The hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the PIR.

The hypervisor investigates the cause of the machine check. The cause can either be a recoverable event on the current processor or one of the other processors in the logical partition. Also the hypervisor must determine if the machine check has corrupted its own internal state by looking at the footprints, if any, that were left in the PIR processor data area of the errant processor.

- ▶ **System (hypervisor) Call Interrupt**

The hypervisor call (hcall) interrupt is a special variety of the **sc** (system call) instruction. The parameters to the hcall() are passed in registers using the POWERPC Application Binary Interface (ABI) definitions. This ABI specifies an interface for compiled application programs to system software. In contrast to the PowerPC ABI, pass by reference parameters are avoided to or from hcall(). This minimizes the address translation problem parameters passed by reference would cause because address translation is disabled automatically when interrupts are invoked. Input parameters can be indexes. Output parameters can be passed in the registers and require special in-line assembler code on the part of the caller. The first parameter in the hypervisor call function table to hcall() is the function token. The assignment of function token is designed such that a single mask operation can be used to validate the value to be within the range of a reasonable size branch table. Entries within the branch table can handle unimplemented code points. And some of

the `hcall()` functions indicate if the system is partitioned, and which ones are available. The Open Firmware property is provided in the `/rtas` node of the partition's device tree. The property is present if the system is partitioned while its value specifies which function sets are implemented by a given implementation. If the system implements any `hcall()` of a function set it implements the entire function set. Additionally, certain values of the Open Firmware property indicate that the system supports a given architecture extension to a standard `hcall()`.

The hypervisor routines are optimized for execution speed. In some rare cases, locks will have to be taken, and short wait loops will be required due to specific hardware designs. However, if a needed resource is truly busy, or processing is required by an agent, the hypervisor returns to the caller, either to have the function retried or continued at a later time. The performance class establishes specific performance against specific `hcall()` function.

6.3 Hypervisor call functions

The hypervisor provides the following functions:

- Page Frame Table

Page Frame Table (PFT) access is called using 64-bit linkage conventions. The hypervisor PFT access functions carefully update a Page Table Entry (PTE) with at least 64-bit store operations since an invalid update sequence could result in machine check. The hypervisor protects check-stop conditions by allocating certain PTE bits for PTE locks and reserve for operating system assumes that the PTE is in use.

For logical addressing, an additional level of virtual addresses translation is managed by the hypervisor. The operating system is not allowed to use the physical address for its memory this includes main storage, memory-mapped I/O (MMIO) space, and NVRAM. The operating system sees main storage as regions of contiguous logical memory. Each logical region is mapped by the hypervisor into a corresponding block of contiguous physical memory on a specific node. All regions on a specific system are the same size though different systems with different amount of memory can have different region sizes since they are the quantum of memory allocation to partitions. That is, partitions are granted memory in region size chunks and if a partition's operating system gives up memory, it is in units of a full region.

- ▶ Translation Control Entry

Translation Control Entry (TCE) access `hcall()` and take as a parameters in the Logical I/O Bus Number which is the logical bus number value derived from the property that are associated with the particular I/O adapter. TCE is responsible for the I/O address to memory address translation in order to perform direct memory access (DMA) transfers between memory and PCI adapters. The TCE tables are allocated in the physical memory.

- ▶ Processor Register Hypervisor Resource Access

Processor Register Hypervisor Resource Access provides controlled in the write access services.

- ▶ Debugger Support

Debugger support provides the capability for the real mode debugger to get to its asynchronous port and beyond the real mode limit register without turning on virtual address translation.

- ▶ Virtual Terminal Support

The hypervisor provides console access to every logical partition without a physical device assigned. The console emulates a vt320 terminal that can be used to access partition system using the Hardware Management Console (HMC). Some functions are limited, and the performance cannot be guaranteed because of the limited bandwidth of the connection between the HMC and the managed system. A partition's device tree that contains one or more nodes notifying that this has been assigned to one or more virtual terminal client adapters. The unit address of the node is used by the partition to map the virtual device(s) to the operating system's corresponding logical representations and notify the partition that the virtual adapter is a Vterm client adapter. The node's interrupts property specifies the interrupt source number that has been assigned to the client Vterm I/O adapter for receive data.

- ▶ Dump Support

Dump support allows the operating system to dump hypervisor data areas in support of field problem diagnostics. The `hcall-dump` function set contains the `H_HYPERVISOR_DATA` `hcall()`. This `hcall()` is enabled or disabled (default disabled) with the HMC.

- ▶ Memory Migration Support

The Memory Migration Support `hcall()` was provided to assist the operating system in the memory migration process. It is the responsibility of the operating system not to change the DMA mappings referenced by the translation buffer. Failure of the operating system to serialize relative to the logical bus numbers might result DMA data corruption within the caller's partition.

► Performance Monitor Support

The performance registers are saved when a virtual processor yields or is preempted. They are restored when the state of the virtual processor is restored on the hardware. A bit in one of the performance monitor registers enables the partition to specify whether the performance monitor registers count when a hypervisor call (except yield) is made (MSR[HV]=1). When a virtual processor yields or is preempted, the performance monitor registers do count. This allows a partition to query the hypervisor to appropriate information regarding hypervisor code and data addresses.

Table 6-1 provides a list of hypervisor calls.

Note: This table is not intended to be a programming reference. Therefore, these calls can change in future levels of firmware. However, the definitions can provide a better understanding of the mechanics within the hypervisor.

Table 6-1 Hypervisor calls

Hypervisor call	Definition
H_REGISTER_VPA	This hcall() provides a data area registered with the Hypervisor by the operating system for each virtual processor. The Virtual Processor Area (VPA) is the control area which contains information used by Hypervisor and the operating system in cooperation with each other.
H_CEDE	This hcall() is to have the virtual processor, which has no useful work to do, enter a wait state ceding its processor capacity to other virtual processor until some useful work appears, signaled either through an interrupt or a H_PROD hcall().
H_CONFER	This hcall() allows a virtual processor to give its cycles to one or all other virtual processors in its partition.
H_PROD	This hcall() makes the specific virtual processor runnable.
H_ENTER	This hcall() adds an entry into the page frame table. PTE high and low order bytes of the page table contains the new entry.
H_PUT_TCE	This hcall() provides mapping of a single 4096 byte page into the specified TCE.

Hypervisor call	Definition
H_READ	This hcall() returns the contents of a specific PTE in GPR4 and GPR5.
H_REMOVE	This hcall() is for invalidating an entry in the page table.
H_BULK_REMOVE	This hcall() is for invalidating up to four entries in the page table.
H_GET_PPP	This hcall() returns the partition's performance parameters.
H_SET_PPP	This hcall() allows the partition to modify its entitled processor capacity percentage and variable processor capacity weight within limits.
H_CLEAR_MODE	This hcall() clears the modified bit in the specific PTE. The second double word of the old PTE is returned in GPR4.
H_CLEAR_REF	This hcall() clears the reference bit in the specific PTE from the partition's node PFT.
H_PROTECT	This hcall() sets the page protects bits in the specific PTE.
H_EOI	This hcall() incorporates the interrupt reset function when specifying an interrupt source number associated with an interpartition logical I/O adapter.
H_IPI	This hcall() generates an interprocessor interrupt.
H_CPPR	This hcall() sets the processor's current interrupt priority.
H_MIGRATE_DMA	This hcall() is extended to serialize the sending of a logical LAN message to allow for migration of TCE mapped DMA pages.
H_PUT_RTCE	This hcall() maps the number of contiguous TCEs in an RTCE to the same number of contiguous I/O adapter TCEs.
H_PAGE_INIT	This hcall() initializes pages in real mode either to zero or to the copied contents of another page.
H_GET_TCE	This standard hcall() is used to manage the interpartition logical LAN adapters's I/O translations.

Hypervisor call	Definition
H_COPY_RDMA	This hcall() copies data from an RTCE table mapped buffer in one partition to an RTCE table mapped buffer in another partition, with the length of the transfer being specified by the transfer length parameter in the hcall().
H_SEND_CRQ	This hcall() sends one 16 byte message to the partner partition's registered Command / Response Queue (CRQ). The CRQ facility provides ordered delivery of messages between authorized partitions.
H_SEND_LOGICAL_LAN	This hcall() sends a logical LAN message.
H_ADD_LOGICAL_LAN_BUF	This hcall() adds receive buffers to the logical LAN receive buffer pool.
H_PIC	This hcall() returns the summation of the physical processor pool's idle cycles.
H_XIRR	This hcall() is extended to report the virtual interrupt source number associated with virtual interrupts associated with an interpartition logical LAN I/O adapter.
H_POLL_PENDING	This hcall() provides the operating system with the ability to perform background administrative functions and the implementation with indication of pending work so that it can more intelligently manage the use of hardware resources.
H_PURR	This hcall() is a new resource provided for Micro-Partitioning and SMT. It provides an actual count of ticks that the shared resource has used on a per virtual processor or per SMT thread basis. In the case of Micro-Partitioning, the virtual processor's Processor Utilization Resource Register (PURR) begins incrementing when the virtual processor is dispatched onto a physical processor. Therefore, comparisons of elapsed PURR with elapsed Timebase provides an indication of how much of the physical processor a virtual processor is getting. The PURR will also count Hypervisor calls made by the partition, with the exception of H_CEDD and H_CONFER. For improved accuracy, the existing hcall() time stamping should be converted to use PURR instead of timebase.

The **lparstat** command in AIX 5L Version 5.3 with **-H** flag displays the partition data with detailed breakdown of hypervisor time by call type, as shown in Figure 6-2.

System configuration: type=Shared mode=Capped smt=On lcpu=2 mem=512 psize=- ent=0.30					
Detailed information on Hypervisor Calls					
Hypervisor Call	Number of Calls	%Total Time Spent	%Hypervisor Time Spent	Avg Call Time(ns)	Max Call Time(ns)
remove	11488	0.0	12.0	403	3613
read	81	0.0	0.1	293	632
nclear_mod	0	0.0	0.0	1	0
page_init	228	0.0	0.5	898	2797
clear_ref	0	0.0	0.0	1	0
protect	0	0.0	0.0	1	0
put_tce	7851	0.0	11.8	576	1256
xirr	99	0.0	0.2	616	1859
eoi	95	0.0	0.1	437	589
ipi	0	0.0	0.0	1	0
cppr	94	0.0	0.1	328	478
asr	0	0.0	0.0	1	0
others	0	0.0	0.0	1	0
enter	13251	0.0	12.7	368	3130
cede	1076	0.0	59.4	21248	3507400
migrate_dma	0	0.0	0.0	1	0
put_rtce	0	0.0	0.0	1	0
confer	0	0.0	0.0	1	0
prod	858	0.0	1.0	439	772
get_ppp	9	0.0	0.0	1791	3135
set_ppp	0	0.0	0.0	1	0
purr	0	0.0	0.0	1	0
pic	9	0.0	0.0	391	618
bulk_remove	507	0.0	1.7	1285	2241
send_crq	125	0.0	0.4	1295	1584
copy_rdma	0	0.0	0.0	1	0
get_tce	0	0.0	0.0	1	0
send_logical_lan	0	0.0	0.0	1	0
add_logical_lan_buf	0	0.0	0.0	1	0

Figure 6-2 lparstat -H command output

6.4 Micro-Partitioning technology extensions

A new virtual processor is dispatched on a physical processor when one of the following conditions happens:

- ▶ The physical processor is idle and a virtual processor was made ready to run (interrupt or process).
- ▶ The old virtual processor exhausted its time slice (HDERC interrupt).
- ▶ The old virtual processor ceded or conferred its cycles.

When one of the above conditions occurs, the hypervisor, by default, records all the virtual processor architected state including the Time Base and Decrementer values and sets the hypervisor timer services to wake the virtual processor per the setting of the decrementer. The virtual processor's Processor Utilization Resource Register (PURR) value for this dispatch is computed. The Virtual Processor Area (VPA) dispatch count is incremented (such that the result is odd). Then the hypervisor selects a new virtual processor to dispatch on the physical processor using an implemented dependent algorithm having the following characteristics listed in priority order:

1. The virtual processor is ready to run (has not ceded or conferred its cycles or exhausted its time slice).
2. Ready-to-run virtual processors are dispatched prior to waiting in excess of their maximum specified latency.
3. Of the non-latency critical virtual processors ready to run, select the virtual processor that is most likely to have its working set in the physical processor's cache or for other reasons will run most efficiently on the physical processor.

If no virtual processor is ready to run at this time, start accumulating the Pool Idle Count of the total number of idle processor cycles in the physical processor pool.

6.5 Memory considerations

POWER5 processors use memory to temporarily hold information. Memory requirements for partitions depend on partition configuration, I/O resources assigned, and applications used. Memory can be assigned in increments of 16 MB.

Depending on the overall memory in your system and the maximum memory values you choose for each partition, the server firmware must have enough memory to perform logical partition tasks. Each partition has a Hardware Page Table (HPT). The size of the HPT is based on an HPT ratio and determined by

the maximum memory values you establish for each partition. The HPT ratio is 1/64.

When selecting the maximum memory values for each partition, consider the following:

- ▶ Maximum values affect the HPT size for each partition.
- ▶ The logical memory map size of each partition.

When you create a logical partition on your managed system, the managed system reserves an amount of memory to manage the logical partition. Some of this physical partition is used for hypervisor page table translation support. The current memory available for partition usage in the HMC is the amount of memory that is currently available to the logical partitions on the managed system, see Figure 6-3. This is the amount of active memory on your managed system minus the estimated memory needed by the managed system to manage the logical partitions currently defined on your system. Therefore, the amount in this field decreases for each additional logical partition you create.

When you are assessing changing performance conditions across system reboots, it is important to know that memory allocations might change based on the availability of the underlying resources. Memory is allocated by the system across the system. Applications in partitions cannot determine where memory has been physically allocated.

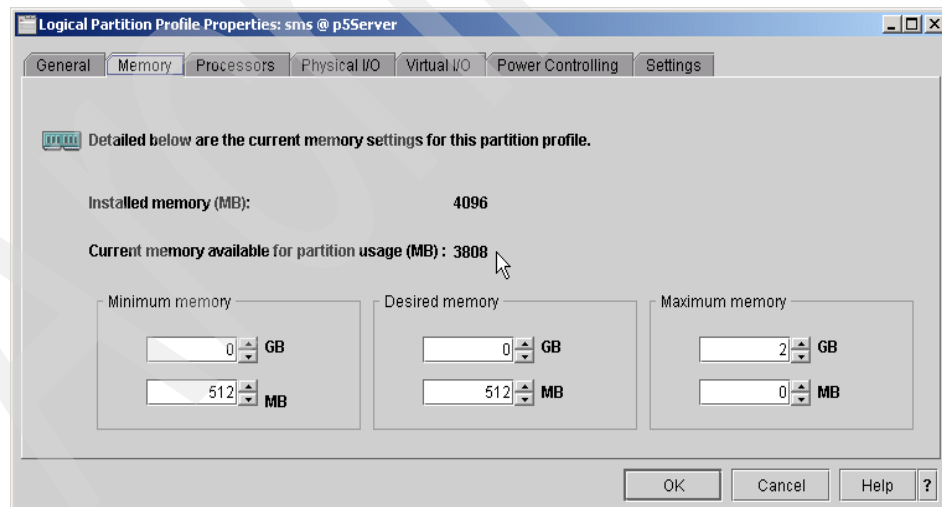


Figure 6-3 Logical Partition Profile Properties - current memory settings

6.6 Performance considerations

The hypervisor does use a small percentage of the system processor and memory resources. This is associated with virtual memory management and is used for the hypervisor dispatcher, virtual processor data structures (including save areas for virtual processor) and for queuing up of interrupts. This is dependant on most workloads, and page-mapping activity. Partitioning can actually help performance in some cases for applications that do not scale well on large SMP systems by enforcing strong separation between workloads running in the separate partitions.

The output of **lparstat** with **-h** flag displays the percentage spent in hypervisor (%hypv) and the number of hcalls. Notice that in the example output shown in Figure 6-4, the %hypv in relation to entitlement capacity is only around 1% of the system resources. This percentage shows that the hypervisor consumes a small amount of the processor during this sample.

```
# lparstat -h 1 16
```

System configuration: type=Shared mode=Capped smt=On lcpu=2 mem=512 psize=- ent=0.30

%user	%sys	%wait	%idle	physc	%ntc	lbusy	app	ucsw	phint	%hypv	hcalls
15.6	76.6	0.0	7.9	0.30	100.3	90.5	-	298	2	1.4	230
15.5	76.8	0.0	7.7	0.30	100.0	90.5	-	321	1	0.9	259
15.6	76.3	0.0	8.1	0.30	100.0	85.0	-	295	1	1.9	224
15.6	76.6	0.0	7.9	0.30	100.0	89.5	-	310	5	1.3	246
15.6	76.7	0.0	7.7	0.30	100.0	91.5	-	299	2	1.0	220
15.6	76.6	0.0	7.8	0.30	100.0	90.0	-	315	1	1.2	249
15.4	75.9	0.0	8.7	0.30	99.1	91.6	-	315	2	1.3	249
10.0	49.6	0.0	40.4	0.19	64.9	58.5	-	442	1	1.0	507
0.0	0.4	0.0	99.6	0.00	1.1	0.0	-	383	1	0.8	456
0.0	0.3	0.0	99.6	0.00	1.0	0.0	-	332	0	0.7	397
0.0	0.4	0.0	99.6	0.00	1.0	0.0	-	334	0	0.8	399
0.0	0.3	0.0	99.7	0.00	0.9	0.0	-	330	0	0.7	391
0.0	0.3	0.0	99.6	0.00	0.9	0.0	-	330	0	0.7	391
0.0	0.3	0.0	99.6	0.00	1.0	0.0	-	335	0	0.7	409
0.0	0.3	0.0	99.6	0.00	1.0	4.9	-	356	0	0.7	425
0.0	0.3	0.0	99.7	0.00	0.9	0.0	-	324	0	0.7	390

```
#
```

Figure 6-4 **lparstat -h 1 16** command output

To provide input to the capacity planning and quality of service tools, the hypervisor reports to an operating system certain statistics, these include the number of virtual processor that are online, minimum processor capacity that the operating system can expect (the operating system can cede any unused capacity back to the system), the maximum processor capacity that the partition will grant to the operating system, the portion of spare capacity (up to the maximum) that the operating system will be granted, variable capacity weight, and the latency to a dispatch by an hcall(). The output of the **lparstat** command

with the `-i` flag, shown in Figure 6-5, reports the logical partition related information.

```
squadron:root[/]lparstat -i
Node Name                : sqltest1
Partition Name           : Test1_AIX_0425A
Partition Number         : 2
Type                     : Shared-SMT
Mode                     : Capped
Entitled Capacity        : 30
Partition Group-ID       : 32770
Shared Pool ID           : 0
Online Virtual CPUs      : 1
Maximum Virtual CPUs     : 5
Minimum Virtual CPUs     : 1
Online Memory            : 512 MB
Maximum Memory           : 1024 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : 0
Minimum Capacity         : 10
Maximum Capacity         : 50
Capacity Increment       : 1
Maximum Dispatch Latency : 13999999
Maximum Physical CPUs in system : 5
Active Physical CPUs in system : 2
Active CPUs in Pool      : -
Unallocated Capacity     : 0
Physical CPU Percentage   : 30.00%
Unallocated Weight       : 0
Minimum Virtual Processor Required Capacity: 10
squadron:root[/]
```

Figure 6-5 `lparstat -i` command output

Dynamic logical partitioning program templates

This appendix provides the following sample dynamic logical partitioning (DLPAR) program templates:

- ▶ “Perl template” on page 233
- ▶ “Korn shell template” on page 245
- ▶ “DLPAR-aware application using a signal handler” on page 257

These templates are provided as samples to integrate DLPAR operations into your applications.

General information

If you can access the source code of your application, see the last template example to make your applications DLPAR-aware.

If not, you can choose your favorite programming language from Perl, Korn shell, and the C language in order to integrate DLPAR operations into your applications. The first two templates are slightly modified from the files originally installed in the /usr/samples/dr/scripts directory on AIX 5L Version 5.2 to add a debug output facility and insert comments for readability. The original template files are included in the bos.adt.samples fileset, as shown in the following example:

```
# ls1pp -w /usr/samples/dr/scripts/IBM_template.*
File                               Fileset                             Type
-----
/usr/samples/dr/scripts/IBM_template.c  bos.adt.samples  File
/usr/samples/dr/scripts/IBM_template.pl  bos.adt.samples  File
/usr/samples/dr/scripts/IBM_template.sh  bos.adt.samples  File
# ls1pp -L bos.adt.samples
Fileset      Level  State  Type  Description (Uninstaller)
-----
bos.adt.samples  5.2.0.0  C      F      Base Operating System
Samples
```

Note: Some scripts in the /usr/samples/dr/scripts directory are provided to demonstrate error situations. For example, IBM_XYZ_fail_dr_2.sh generates the AIX error log shown in Example A-1 in a CPU removal DLPAR event. You should carefully read the readme file in this directory before registering these scripts.

Example: A-1 DR_UNSAFE_PROCESS

LABEL:	DR_UNSAFE_PROCESS
IDENTIFIER:	0E2A04B4
Date/Time:	Fri Nov 15 11:04:17 CST
Sequence Number:	114
Machine Id:	0021768A4C00
Node Id:	lpar01
Class:	S
Type:	INFO
Resource Name:	SYSPROC
Description	DR Unsafe application
Detail Data	
Process ID	30499421762355200

Perl template

Example A-2 provides the Perl version of the DLPAR script template. System administrators can use it as a starting point when integrating DLPAR operations into your applications.

The script is added a file handle, DBG, to be used to print debug information to the debug file, /tmp/IBM_template.pl.dbg. The debug information is very helpful when you have to debug your DLPAR script, because the script should not print any undefined name-value pairs to the standard out.

To display the debug information sent to the file, enter the following command:

```
$ tail -f /tmp/IBM_template.pl.dbg
```

Example: A-2 DLPAR script template: Perl

```
#!/usr/bin/perl

# (C) COPYRIGHT International Business Machines Corp. 2000, 2002
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# FILE NAME: IBM_template.pl
#
# FILE DESCRIPTION:
# This perl script will provide a template for DLPAR script developers
# to develop their custom perl scripts.
#
# It is the basic test script. It implements all the commands of DLPAR
# script and for all them returns success. It identifies itself with
# distinct details.

#-----
# GLOBAL OBJECTS
#-----

# This hash contains the required commands specified in the DRAF.
# The values assigned to each command is irrelevant. The script
# converts the text into a number for quick accessing
%DR_script_commands = (
    scriptinfo => 1,
    register => 2,
    usage => 3,
    checkrelease => 4,
    prerelease => 5,
```

```

    postrelease => 6,
    undoprerelease => 7,
    checkacquire => 8,
    preacquire => 9,
    postacquire => 10,
    undopreacquire => 11
);
# This hash contains data used by the scriptinfo command. It lists
# required information about the script
%SCRIPT_DATA = (
    SCRIPT_INFO => "AIX ",
    SCRIPT_VERSION => "1",
    SCRIPT_VENDOR => "IBM Corp.",
    SCRIPT_TIMEOUT => 5
);
# This hash contains the resources for which the script will register.
# In this case, this script wants to register for DR operations that
# involve memory and cpu events.
%REGISTER_DATA = (
    CPU_RESOURCE => "cpu",
    MEM_RESOURCE => "mem"
);
# This hash contains usage descriptions for each possible resource.
%USAGE_DATA = (
    CPU_USAGE => "Testing DLPAR on CPU resource",
    MEM_USAGE => "Testing DLPAR on MEM resource"
);

#-----
# Helper Functions
#-----

#=====
# Name:  str_to_cmd
#
# Description: converts a string to a command value
#
# Input: command string
#
# Output: logically mapped command value
#
# Return Code:  None
#=====
sub str_to_cmd {

    $s_cmd = $_[0];
    $DR_script_commands{$s_cmd};
}

```

```

#-----
# Required DRAF commands
#-----

#=====
# Name: process_scriptinfo
#
# Description: returns information about the script
#
# Input: none
#
# Output: name-value pairs
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_scriptinfo {

    print "DR_SCRIPTINFO=$SCRIPT_DATA{SCRIPT_INFO}\n";
    print "DR_VERSION=$SCRIPT_DATA{SCRIPT_VERSION}\n";
    print "DR_DATE=19042002\n";
    print "DR_VENDOR=$SCRIPT_DATA{SCRIPT_VENDOR}\n";
    print "DR_TIMEOUT=$SCRIPT_DATA{SCRIPT_TIMEOUT}\n";

    0;
}

#=====
# Name: process_scriptinfo
#
# Description: returns information about the script
#
# Input: none
#
# Output: name-value pairs
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_register {

    foreach $key ( keys %REGISTER_DATA){
        print "DR_RESOURCE=$REGISTER_DATA{$key}\n";
    }
    0;
}

#=====

```

```

# Name:  process_usage
#
# Description:  returns usage information about the script
#
# Input:  resource
#
# Output:  name-value pairs
#
# Return Code:  0 = success
#               1 = failure
#=====
sub process_usage {

    $l_rc = 0;
    $res = $_[0];

    USE_SWITCH: {
        if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
            print "DR_USAGE=$USAGE_DATA{CPU_USAGE}\n";
            last USE_SWITCH;
        }
        if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
            print "DR_USAGE=$USAGE_DATA{MEM_USAGE}\n";
            last USE_SWITCH;
        }
        print "DR_ERROR=script does not use resource $res\n";
        $l_rc = 1;
    }
    return $l_rc;
}

#=====
# Name:  process_checkrelease
#
# Description:  verifies a resource can be removed without compromises
#               the application.
#
# Input:  resource
#
# Output:  Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code:  0 = success
#               1 = failure
#=====
sub process_checkrelease {

    $l_rc = 0;
    $res = $_[0];

```

```

print DBG "-- start checkrelease phase --\n";
foreach $key (sort keys %ENV) {
    if ($key =~ /^DR_/) {
        print DBG $key, '=', $ENV{$key}, "\n";
    }
}

USE_SWITCH: {
    if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
        # perform all cpu related checks here and determine
        # if resource remove can proceed.
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # perform all memu related checks here and determine
        # if resource remove can proceed.
        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end checkrelease phase --\n";

return $l_rc;
}

#####
# Name: process_prerelease
#
# Description: Prepares for the resource to be removed
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#####
sub process_prerelease {

    $l_rc = 0;
    $res = $_[0];

    print DBG "-- start prerelease phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }
}

```

```

}

# before we allow DR manager to proceed, we can do any prerelease
# actions here. For instance, we could send a signal from here
# and wait for the application to take some action.

# resource specific actions
USE_SWITCH: {
    if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
        # release any cpu bindings, etc. here if the resource
        # is being used by the application.
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # release any application hold over memory, etc, that
        # is being removed.
        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end prerelease phase --\n";

return $l_rc;
}

#=====
# Name: process_undoprerelease
#
# Description: Invoked to undo any changes done by the prerelease
#              command.
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_undoprerelease {

    $l_rc = 0;
    $res = $_[0];
    # perform any actions here which were performed in the prerelease
    # command.

    print DBG "-- start undoprerelease phase --\n";
    foreach $key (sort keys %ENV) {

```



```

        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }

    # resource specific actions
    USE_SWITCH: {
        if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
            # undo cpu related changes done by the prerelease cmd
            last USE_SWITCH;
        }
        if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
            # undo mem related changes done by the prerelease cmd
            last USE_SWITCH;
        }
        print "DR_ERROR=script does not use resource $res\n";
        $l_rc = 1;
    }

    print DBG "-- end undoprerelease phase --\n";

    return $l_rc;
}

#=====
# Name: process_postrelease
#
# Description: After the resource is removed, this command makes
#              necessary adjustments
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_postrelease {

    $l_rc = 0;
    $res = $_[0];

    # reacquire any resource released during prerelease
    # activate any applications quieced during prerelease.

    print DBG "-- start postrelease phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }
}

```

```

    }
}

# resource specific actions
USE_SWITCH: {
    if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
        # perform cpu related actions.
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # perform mem related actions.
        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end postrelease phase --\n";
return $l_rc;
}

#####
# Name: process_checkacquire
#
# Description: verifies a resource can be added without
#               compromising the application or system.
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#####
sub process_checkacquire {

    $l_rc = 0;
    $res = $_[0];

    print DBG "-- start checkacquire phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }

    # resource specific actions
    USE_SWITCH: {
        if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {

```

```

        # perform all cpu related checks here and determine
        # if resource addition can proceed.
        print DBG "cpu resources: logical $ENV{DR_LCPUID}, bind $ENV{DR_BCPUID}\n";
        if ($END{DR_LCPUID} eq 2) {
            $l_rc = 1;
        }
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # perform all mem related checks here and determine
        # if resource addition can proceed.
        print DBG "mem resources: $ENV{DR_MEM_SIZE_REQUEST}\n";
        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end checkacquire phase --\n";
return $l_rc;
}

#=====
# Name: process_preacquire
#
# Description: prepares application before the resource is added
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_preacquire {
    $l_rc = 0;
    $res = $_[0];

    print DBG "-- start preacquire phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }
}

# resource specific actions
USE_SWITCH: {
    if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {

```

```

        # Prepare application for cpu additions.
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # Prepare application for memory additions.
        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end preacquire phase --\n";

return $l_rc;
}

#=====
# Name: process_undopreacquire
#
# Description: If a failure occurs, this will undo any changes made by
#              the preacquire command.
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
#=====
sub process_undopreacquire {

    $l_rc = 0;
    $res = $_[0];

    print DBG "-- start undopreacquire phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }

    # resource specific actions
    USE_SWITCH: {
        if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
            # undo cpu actions taken in the preacquire command.
            last USE_SWITCH;
        }
        if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
            # undo mem actions taken in the preacquire command.

```

```

        last USE_SWITCH;
    }
    print "DR_ERROR=script does not use resource $res\n";
    $l_rc = 1;
}

print DBG "-- end undopreacquire phase --\n";

return $l_rc;
}

=====
# Name: process_postacquire
#
# Description: After a resource has been added, this will perform any
#              necessary actions.
#
# Input: resource
#
# Output: Any DR_ERROR=description or DR_LOG_DEBUG=description variables
#
# Return Code: 0 = success
#              1 = failure
=====
sub process_postacquire {

    $l_rc = 0;
    $res = $_[0];

    print DBG "-- start undopreacquire phase --\n";
    foreach $key (sort keys %ENV) {
        if ($key =~ /^DR_/) {
            print DBG $key, '=', $ENV{$key}, "\n";
        }
    }
}

# resource specific actions
USE_SWITCH: {
    if ($res eq $REGISTER_DATA{CPU_RESOURCE}) {
        # Perform actions to allow the application to adjust to a
        # cpu addition such as adding more threads, etc.
        last USE_SWITCH;
    }
    if ($res eq $REGISTER_DATA{MEM_RESOURCE}) {
        # Perform actions to allow the application to adjust to a
        # memory addition such as increasing memory areas reserved
        # for application, etc.
        last USE_SWITCH;
    }
}

```

```

        print "DR_ERROR=script does not use resource $res\n";
        $l_rc = 1;
    }

    print DBG "-- end undopreacquire phase --\n";

    return $l_rc;
}

#-----
# Main Program
#-----

# because we should only write the specified name-value
# pairs in the DRAF, we should print debug information
# to a file.
open (DBG, ">>/tmp/IBM_template.pl.dbg");

# This block processes the command line inputs.
ARG_SWITCH: {
    if ($#ARGV == -1) { $rc = -1; last ARG_SWITCH; }
    if ($#ARGV == 0) { $command_str = $ARGV[0] ; last ARG_SWITCH; }
    if ($#ARGV == 1) { $command_str = $ARGV[0]; $res_name = $ARGV[1]; last ARG_SWITCH; }
    $rc = -2;
}

# Convert the string to a command.
$command = str_to_cmd $command_str;

#This block invokes the proper function to handle the command
CMD_SWITCH: {
    if ($command == '') {$rc = 10; print "DR_ERROR=command not supported\n"; last CMD_SWITCH }
    if ($command == 1) {$rc = process_scriptinfo; last CMD_SWITCH }
    if ($command == 2) {$rc = process_register; last CMD_SWITCH }
    if ($command == 3) {$rc = process_usage $res_name; last CMD_SWITCH }
    if ($command == 4) {$rc = process_checkrelease $res_name; last CMD_SWITCH }
    if ($command == 5) {$rc = process_prerelease $res_name; last CMD_SWITCH }
    if ($command == 6) {$rc = process_postrelease $res_name; last CMD_SWITCH }
    if ($command == 7) {$rc = process_undoprerelease $res_name; last CMD_SWITCH }
    if ($command == 8) {$rc = process_checkacquire $res_name; last CMD_SWITCH }
    if ($command == 9) {$rc = process_preacquire $res_name; last CMD_SWITCH }
    if ($command == 10) {$rc = process_postacquire $res_name; last CMD_SWITCH }
    if ($command == 11) {$rc = process_undopreacquire $res_name; last CMD_SWITCH }
}

# close the debug file handle
close(DBG);
# exit status generated from command processing
$rc;

```

Korn shell template

Example A-3 provides the Korn shell version of the DLPAR script template. System administrators can use it as a starting point when integrating DLPAR operations into your applications.

The script sends debug information to the debug file, /tmp/<script_file_name>.dbg. The debug information is very helpful when you have to debug your DLPAR script, because the script should not print any undefined name-value pairs to the standard out.

To display the debug information sent to the file, enter the following command:

```
$ tail -f /tmp/<script_file_name>.dbg
```

Example: A-3 DLPAR script template: Korn shell

```
#!/usr/bin/ksh
# (C) COPYRIGHT International Business Machines Corp. 2000, 2002
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
# DLPAR aware Application developers will modify this script to
# develop a DLPAR script to suit their application's needs of control
# vis-a-vis Dynamic Reconfiguration(DLPAR) Operations.
#

# FILE NAME: IBM_template.sh
#
# FILE DESCRIPTION:
# This is an example template shell DLPAR script file for
# Dynamic Reconfiguration Application Framework of AIX.
# This template file just prints the various inputs
# from drmgr.
#
# Note that DLPAR script file should adhere to the guidelines
# related to AIX Dynamic Reconfiguration. Some of the
# issues to be considered while changing this script file
# are:
# 1. Output name=value pairs only to stdout
# as per the DRAF guidelines. Refer to
# Manuals related to DLPAR for more details.
# 2. Return 0 upon success, 10 if the command
```

```

#      is not implemented, else return any other
#      return code (1 to 9, 11 to 255)
# 3. Use DRAF defined environment variables and
#      input parameters for processing the
#      command.
# 4. To debug the script file, one can use
#      the method shown in this template file.
#
# RETURN VALUE DESCRIPTION:
# 0          Successful
# 10         Command not implemented
# Else       Error
#
##### dbg #####
#
# NAME: dbg()
#
# DESCRIPTION: Write the debug message to debug file
#
# INPUT:
# Message to write to debug file
#
# OUTPUT:
# Message echoed to the debug file.
#
# RETURN VALUE DESCRIPTION:
# None
#
#####

dbg()
{
    echo $1 >> ${DBG_FILE_NAME}
}

##### process_scriptinfo #####
#
# NAME: process_scriptinfo()
#
# DESCRIPTION: Process 'scriptinfo' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
#
# OUTPUT:
# Output name=value pairs to stdout
# Various pieces of information about the DLPAR script.
#
# RETURN VALUE DESCRIPTION:

```



```

# 0 : success
# Else failure.
#
#####

process_scriptinfo()
{
    echo "DR_SCRIPTINFO=AIX DR ksh example script"
    echo "DR_VERSION=1"
    echo "DR_DATE=18102002"
    echo "DR_VENDOR=IBM"
    echo "DR_TIMEOUT=10"
    return 0
}

##### process_register #####
#
# NAME: process_register()
#
# DESCRIPTION: Process 'register' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
#
# OUTPUT:
# Output name=value pairs to stdout
# List of all the resources supported by this DLPAR script.
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# Else failure.
#
#####

process_register()
{
    echo "DR_RESOURCE=cpu"
    echo "DR_RESOURCE=mem"
    return 0
}

##### process_usage #####
#
# NAME: process_usage()
#
# DESCRIPTION: Process 'usage' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr

```

```

# resource name input variable
#
# OUTPUT:
# Output name=value pairs to stdout
# Writes the how this resource is being used by the application
# associated with this DLPAR script.
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# Else failure.
#
#####

process_usage()
{
    case "$1" in
        "cpu")
            echo "DR_USAGE=cpu binding for performance"
            ;;
        "mem")
            echo "DR_USAGE=Shared(Pinned) memory for app XYZ"
            ;;
        *)
            echo "DR_ERROR=Script does not use Resource $1"
            ;;
    esac
    return 0
}

##### process_checkrelease #####
#
# NAME: process_checkrelease()
#
# DESCRIPTION: Process 'checkrelease' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR operatin is not ok with the DLPAR script/associated app.

```

```

#
#####

process_checkrelease()
{
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            # Do all the cpu related checks here and determine
            # whether DLPAR remove can proceed.
            ;;
        "mem")
            dbg "Resource : mem"
            # Do all the memory related checks here and determine
            # whether DLPAR remove can proceed.
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac

    return 0
}

##### process_prerelease #####
#
# NAME: process_prerelease()
#
# DESCRIPTION: Process 'prerelease' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application could not release the resource
# for DLPAR operation.
#
#####
process_prerelease()
{

```

```

# Do any pre release actions here. One could send a signal
# from here and wait for application do the necessary.
# Return from here only after the desired actions have
# taken place.
case "$1" in
    "cpu")
        bg "Resource : cpu"
        # Release any cpu bindings etc here if the
        # resource being released is used by the app.
        ;;
    "mem")
        dbg "Resource : mem"
        # Release application hold over any memory
        # that is being removed.
        ;;
    *)
        echo "DR_ERROR=Script does not support Resource $1"
        ;;
esac

return 0
}

##### process_postrelease #####
#
# NAME: process_postrelease()
#
# DESCRIPTION: Process 'postrelease' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application could not post DLPAR operations.
#
#####
process_postrelease()
{
    # Reacquire any resource release during prerelease.
    # activate any apps quieced during prerelease.

```

```

    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac

    return 0
}

##### process_undoprerelease #####
#
# NAME: process_undoprerelease()
#
# DESCRIPTION: Process 'process_undoprerelease' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application failed undorelease
#
#####
process_undoprerelease()
{
    # DLPAR operation was aborted/failed. Hence undo any
    # changes done during prerelease for this resource
    # and the application associated with the DLPAR script.
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"

```

```

        ;;
        *)
        echo "DR_ERROR=Script does not support Resource $1"
        ;;
    esac
    return 0
}

##### process_checkacquire #####
#
# NAME: process_checkacquire()
#
# DESCRIPTION: Process 'process_checkacquire' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application does want this resource.
#
#####
process_checkacquire()
{
    # Do any checks prior to resource addition.
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac
    return 0
}

##### process_preacquire #####
#
# NAME: process_preacquire()

```

```

#
# DESCRIPTION: Process 'process_preacquire' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application preacquire failed.
#
#####
process_preacquire()
{
    # Do all the necessary work prior to resource addition.
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac
    return 0
}

##### process_undopreacquire #####
#
# NAME: process_undopreacquire()
#
# DESCRIPTION: Process 'process_undopreacquire' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."

```

```

# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application undopreacquire failed.
#
#####
process_undopreacquire()
{
    # DLPAR operation has failed. So undo any activities done during
    # preacquire
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac
    return 0
}
##### process_postacquire #####
#
# NAME: process_postacquire()
#
# DESCRIPTION: Process 'process_postacquire' command from drmgr
#
# INPUT:
# The various environment variables set by drmgr
# resource name input variable
#
# OUTPUT:
# Any debug information using DR debug name=value pairs
# such as DR_LOG_DEBUG="..."
# Any error message in the form of DR_ERROR name=value pair
#
#
# RETURN VALUE DESCRIPTION:
# 0 : success
# 10 : Command not implemented for this resource.
# Else DLPAR script/associated application postacquire failed.
#
#####
process_postacquire()

```



```

{
    # execute any actions required after the DLPAR add operation.
    # Egs: Increase the number of threads for the application
    #       Increase memory areas reserved for application etc.
    case "$1" in
        "cpu")
            dbg "Resource : cpu"
            ;;
        "mem")
            dbg "Resource : mem"
            ;;
        *)
            echo "DR_ERROR=Script does not support Resource $1"
            ;;
    esac
    return 0
}

```

```

#####
#           MAIN SCRIPT STARTS HERE
#####

```

```

script_file_name=`basename $0`
DBG_FILE_NAME=/tmp/${script_file_name}.dbg

```

```

date_and_time=`date`
dbg "----- DLPAR Script start at $date_and_time -----"

```

```

if [ $# -eq 0 ]; then
    # Atleast the command must be part of the invocation
    dbg "No command passed to the DLPAR script"
    echo "DR_ERROR=Script Usage Error"
    exit 1
fi

```

```

# Note down the command
command=$1
ret_code=0

```

```

dbg "command issued: $1"
case "$1" in
    scriptinfo)
        process_scriptinfo
        ret_code=$?
        ;;
    register)
        process_register
        ret_code=$?
        ;;

```

```

usage)
process_usage $2
ret_code=?
;;
checkrelease)
process_checkrelease $2
ret_code=?
;;
prerelease)
process_prerelease $2
ret_code=?
;;
postrelease)
process_postrelease $2
ret_code=?
;;
undoprerelease)
process_undoprerelease $2
ret_code=?
;;
checkacquire)
process_checkacquire $2
ret_code=?
;;
preacquire)
process_preacquire $2
ret_code=?
;;
undopreacquire)
process_undopreacquire $2
ret_code=?
;;
postacquire)
process_postacquire $2
ret_code=?
;;
*)
dbg "unknown command: $1 issued"
ret_code=10
;;
esac

dbg "SCRIPT exiting with return code : $ret_code"

dbg ".....DLPAR Script end ....."

return $ret_code

```

DLPAR-aware application using a signal handler

If you can access the source code of your application, you can modify your application by adding a signal handler that reacts to the SIGRECONFIG signal so that the application is made DLPAR-aware. The SIGRECONFIG signal is delivered to the application process upon DLPAR events.

How to compile and run the application

Before running this application, you must compile the C source code by executing the following command¹:

```
$ cc -o DLPAR_app1 DLPAR_app1.c
```

In this command:

DLPAR_app1	Is the application program name to be executed
DLPAR_app1.c	Is the file name of C program source code

To run this application, enter DLPAR_app1 at the command line prompt. To stop it, type Control-C at the command line prompt where you have invoked it.

The application process sends debug information to the /tmp/dr_api_template.C.dbg file. To display the debug information sent to the file, enter the following:

```
$ tail -f /tmp/dr_api_template.C.dbg
```

Example: A-4 C language application with a signal handler

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>
#include <sys/dr.h>

/*=====*/
/* Prototypes */
/*=====*/
void perror_msg(char *func_name, int errno_old, const int line_num);
void dr_func(int arg);
int checkAcquire_mem(void);
int checkAcquire_cpu(void);
int preAcquire_mem(void);
int preAcquire_cpu(void);
int postAcquire_mem(void);
int postAcquire_cpu(void);
```

¹ This example requires that the C compiler is installed and available on your AIX system.

```

int postAcquireError_mem(void);
int postAcquireError_cpu(void);
int checkRelease_mem(void);
int checkRelease_cpu(void);
int preRelease_mem(void);
int preRelease_cpu(void);
int postRelease_mem(void);
int postRelease_cpu(void);
int postReleaseError_mem(void);
int postReleaseError_cpu(void);

/*=====*/
/*  Globals                                     */
/*=====*/
extern int errno;
dr_info_t dr_info;
char msg_buf[BUFSIZ];
FILE * l_dbgFd;

typedef struct {
    int (*mem_ptr)(void);
    int (*cpu_ptr)(void);
} phases_t;

phases_t definedPhase[] = { /* #J */
    { &checkAcquire_mem,    &checkAcquire_cpu    },
    { &preAcquire_mem,      &preAcquire_cpu    },
    { &postAcquire_mem,     &postAcquire_cpu    },
    { &postAcquireError_mem,&postAcquireError_cpu },
    { &checkRelease_mem,    &checkRelease_cpu    },
    { &preRelease_mem,      &preRelease_cpu    },
    { &postRelease_mem,     &postRelease_cpu    },
    { &postReleaseError_mem,&postReleaseError_cpu },
};

#define CHECK_ACQUIRE    &definedPhase[0];
#define PRE_ACQUIRE      &definedPhase[1];
#define POST_ACQUIRE     &definedPhase[2];
#define POST_ACQUIRE_ERROR &definedPhase[3];
#define CHECK_RELEASE     &definedPhase[4];
#define PRE_RELEASE       &definedPhase[5];
#define POST_RELEASE      &definedPhase[6];
#define POST_RELEASE_ERROR &definedPhase[7];

#define DR_API_SUCCESS    0
#define DR_API_FAIL       1

/*=====*/
/*  Helper Functions                                     */
/*=====*/

```

```

/*=====*/

/*=====*/
/* Description:  Handles any unexected errors */
/* */
/* Output: none */
/* Inputs: function name */
/*      errno */
/*      lineNumber */
/*=====*/
void
perror_msg(char *func_name, int errno_old, const int line_num)
{
    sprintf(msg_buf
        , "%s failed with errno = %d at line (%d).\n"
        , func_name, errno_old, line_num);
    perror(msg_buf);

    return;
}

/*=====*/
/* Description: Memory Check Acquire phase */
/* */
/*      This function should ensure that a memory addition will not */
/*      disrupt the application, before the actual DR event occurs. */
/* */
/* Output: integer value indicating status of DR operation */
/*      Values: DR_API_SUCCESS */
/* Input: None */
/*=====*/
int checkAcquire_mem(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered CheckedAcquire_mem*****\n");

    /* Check for plock'd memory */
    if (dr_info.plock) {
        fprintf(l_dbgFd, "\t-- process has plock()'ed memory --\n");
        l_rc = DR_API_FAIL;
    }
    /* check for pinned memory */
    if (dr_info.pshm) {
        fprintf(l_dbgFd, "\t-- process has pinned shared memory --\n");
        l_rc = DR_API_FAIL;
    }

    return l_rc;
}

```

```

}

/*=====*/
/* Description: CPU check acquire phase */
/* This function should ensure that a cpu addition will not */
/* disrupt the application, before the actual DR event takes place*/
/* */
/* Output: integer value indicating status of DR operation */
/* Values: DR_API_SUCCESS */
/* DR_API_FAIL */
/* Inputs: None */
/*=====*/
int checkAcquire_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered CheckedAcquire_cpu*****\n");

    /* check for processor dependencies */
    if (dr_info.bindproc) {
        fprintf(l_dbgFd, "\t-- process has bindprocessor() dependency --\n");
        l_rc = DR_API_FAIL;
    }
    if (dr_info.softpset) {
        fprintf(l_dbgFd, "\t-- process has soft pset() dependency --\n");
        l_rc = DR_API_FAIL;
    }
    if (dr_info.hardpset) {
        fprintf(l_dbgFd, "\t-- process has hard pset() dependency --\n");
        l_rc = DR_API_FAIL;
    }

    return l_rc;
}

/*=====*/
/* Mem pre acquire phase */
/* */
/* Detailed Description: */
/* This function should ensure that the necessary steps are taken */
/* to prepare the application for a memory addition. If need be, */
/* the application should be halted. */
/* */
/* Output: integer value indicating status of DR operation */
/* Values: DR_API_SUCCESS */
/* DR_API_FAIL */
/* Inputs: None */
/*=====*/
int preAcquire_mem(void)

```

```

{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PreeAcquire_mem*****\n");

    /* Perform actions here. */

    return l_rc;
}

/*=====*/
/* CPU pre acquire phase */
/*
/* Detailed Description:
/*     This function should ensure that the necessary steps are taken
/*     to prepare the application for a cpu addition. If need be,
/*     the application should be stopped.
/*
/* Output: integer value indicating status of DR operation
/*     Values:  DR_API_SUCCESS
/*              DR_API_FAIL
/* Inputs: None
/*=====*/
int preAcquire_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PreAcquire_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Mem post acquire phase */
/*
/* Detailed Description:
/*     After a memory addition has taken place, this function should
/*     perform any actions to clean up the DR operation and allow the
/*     application to use the new resources. If the application was
/*     stopped, it should be restarted here.
/*
/* Output: integer value indicating status of DR operation
/*     Values:  DR_API_SUCCESS
/*              DR_API_FAIL
/* Inputs: None
/*=====*/
int postAcquire_mem(void)

```

```

{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostAcquire_mem*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* CPU post acquire phase */
/*
/* Detailed Description: */
/*     After a cpu addition, this function allows the application */
/*     access to the new resources. If the application was stopped, */
/*     iy should be restarted here. */
/*
/* Output: integer value indicating status of DR operation */
/*     Values: DR_API_SUCCESS */
/*             DR_API_FAIL */
/* Inputs: None */
/*=====*/
int postAcquire_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostAcquire_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Handles post acquire Error phase for mem */
/*
/* Detailed Description: */
/*     If an error should occur, this phase should handle undoing the */
/*     preacquire actions taken for mem rmovals. */
/*
/* Output: integer value indicating status of DR operation */
/*     Values: DR_API_SUCCESS */
/*             DR_API_FAIL */
/* Inputs: None */
/*=====*/
int postAcquireError_mem(void)
{
    int l_rc = DR_API_SUCCESS;

```



```

    fprintf(l_dbgFd, "*****Entered PostAcquireError_mem*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Handles post acquire Error phase for cpu */
/*
/* Detailed Description:
/*     If an error should occur, this phase should handle undoing the
/*     preacquire actions taken for cpu rmovals.
/*
/* Output: integer value indicating status of DR operation
/*     Values: DR_API_SUCCESS
/*             DR_API_FAIL
/* Inputs: None
/*=====*/
int postAcquireError_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostAcquireError_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Mem check release phase */
/*
/* Detailed Description:
/*     This should check to make sure the application can tolerate a
/*     memory removal. If not, this should terminate the DR operation*/
/*
/* Output: integer value indicating status of DR operation
/*     Values: DR_API_SUCCESS
/*             DR_API_FAIL
/* Inputs: None
/*=====*/
int checkRelease_mem(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered CheckeRelease_mem*****\n");

```

```

/* Check for memory issues */
if (dr_info.plock) {
    fprintf(l_dbgFd, "\t-- process has plock()'ed memory --\n");
    l_rc = DR_API_FAIL;
}
if (dr_info.pshm) {
    fprintf(l_dbgFd, "\t-- process has pinned shared memory --\n");
    l_rc = DR_API_FAIL;
}
return l_rc;
}

/*=====*/
/* Handles post release Error phase for cpu */
/*
/* Detailed Description:
/*     If an error should occur, this phase should handle undoing the
/*     prerelease actions taken for cpu rmovs.
/*
/* Output: integer value indicating status of DR operation
/*     Values: DR_API_SUCCESS
/*             1 DR_API_FAIL
/* Inputs: None
/*=====*/
int checkRelease_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered CheckRelease_cpu*****\n");

    /* Check for processor dependencies */
    if (dr_info.bindproc) {
        fprintf(l_dbgFd, "\t-- process has bindprocessor() dependency --\n");
        l_rc = DR_API_FAIL;
    }
    if (dr_info.softpset) {
        fprintf(l_dbgFd, "\t-- process has soft pset() dependency --\n");
        l_rc = DR_API_FAIL;
    }
    if (dr_info.hardpset) {
        fprintf(l_dbgFd, "\t-- process has hard pset() dependency --\n");
        l_rc = DR_API_FAIL;
    }
    return l_rc;
}

/*=====*/
/* Mem pre release phase */
/*

```

```

/*                                                                    */
/* Detailed Description:                                              */
/*     This function should prepare the application for memory      */
/*     resources being removed.                                     */
/*                                                                    */
/* Output: integer value indicating status of DR operation          */
/*     Values: DR_API_SUCCESS                                       */
/*             DR_API_FAIL                                          */
/* Inputs: None                                                     */
/*=====*/
int preRelease_mem(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PreRelease_mem*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Cpu pre release phase                                           */
/*                                                                    */
/* Detailed Description:                                              */
/*     This should prepare the application for cpu resources being  */
/*     removed.                                                     */
/*                                                                    */
/* Output: integer value indicating status of DR operation          */
/*     Values: DR_API_SUCCESS                                       */
/*             DR_API_FAIL                                          */
/* Inputs: None                                                     */
/*=====*/
int preRelease_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PreRelease_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Mem post release phase                                           */
/*                                                                    */
/* Detailed Description:                                              */
/*     After the memory resources are removed, this function should */

```

```

/*      take care of cleaning up any DR modifications made and allow */
/*      the application to continue running.                        */
/*                                                                  */
/* Output: integer value indicating status of DR operation          */
/*      Values: DR_API_SUCCESS                                     */
/*              DR_API_FAIL                                       */
/* Inputs: None                                                    */
/*=====*/
int postRelease_mem(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostReleasee_mem*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Cpu post release phase                                         */
/*                                                                  */
/* Detailed Description:                                          */
/*      After cpu resources are removed, this function should handle */
/*      the application so that it can continue after the DR operation.*/
/*                                                                  */
/* Output: integer value indicating status of DR operation          */
/*      Values: DR_API_SUCCESS                                     */
/*      Values: DR_API_FAIL                                       */
/* Inputs: None                                                    */
/*=====*/
int postRelease_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostRelease_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Handles post release Error phase for mem                      */
/*                                                                  */
/* Detailed Description:                                          */
/*      If an error should occur, this phase should handle undoing the */
/*      prerelease actions taken for mem rmovals.                */
/*                                                                  */
/*                                                                  */

```

```

/* Output: integer value indicating status of DR operation          */
/*   Values: DR_API_SUCCESS                                         */
/*   DR_API_FAIL                                                    */
/* Inputs: None                                                     */
/*=====*/
int postReleaseError_mem(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostReleaseError_mem*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Handles post release Error phase for cpu                        */
/*                                                                    */
/* Detailed Description:                                           */
/*   If an error should occur, this phase should handle undoing the */
/*   prerelease actions taken for cpu rmovals.                    */
/*                                                                    */
/* Output: integer value indicating status of DR operation          */
/*   Values: DR_API_SUCCESS                                         */
/*   DR_API_FAIL                                                    */
/* Inputs: None                                                     */
/*=====*/
int postReleaseError_cpu(void)
{
    int l_rc = DR_API_SUCCESS;

    fprintf(l_dbgFd, "*****Entered PostReleaseError_cpu*****\n");

    /* Perform actions here */

    return l_rc;
}

/*=====*/
/* Functions                                                         */
/*=====*/

/*=====*/
/* SIGRECONFIG signal handler                                       */
/*                                                                    */
/* Detailed Description:                                           */
/*   This will handle the signal, when a DR event occurs. The main */
/*   information is communicated through a dr_info_t data structure.*/

```

```

/*      This will take care of parsing the data structure and taking      */
/*      appropriate actions.                                              */
/*                                                                           */
/* Output: None                                                            */
/* Input: argument, not used for compatability                           */
/*=====*/
void
dr_func(int arg)
{
    int l_rc= DR_API_SUCCESS;
    phases_t *l_currentPhase;

/* #1 create debug output file stream */
    l_dbgFd = fopen("/tmp/dr_api_template.C.dbg", "a"); /* open for appending */
    if (l_dbgFd == NULL) {
        perror_msg("NULL file descriptor", errno, __LINE__);
        exit(1);
    }

    fprintf(l_dbgFd, "---Start of Signal Handler---\n");

    l_rc = dr_reconfig(DR_QUERY, &dr_info);          /* #C */
    if (l_rc != 0) {
        perror_msg("dr_reconfig()", errno, __LINE__);
        exit(1);
    }

/* #2 determine type of operation and phase. */

    /* addition operations */
    if (dr_info.add) {                                /* #D */
        fprintf(l_dbgFd, "An add request\n");

        /* now determine which acquire phase we are in. */
        if (dr_info.check) {
            fprintf(l_dbgFd, "\t** check phase **\n");
            l_currentPhase = CHECK_ACQUIRE;
        } else if (dr_info.pre) {
            fprintf(l_dbgFd, "\t** pre phase **\n");
            l_currentPhase = PRE_ACQUIRE;
        } else if (dr_info.post) {
            fprintf(l_dbgFd, "\t** post phase **\n");
            l_currentPhase = POST_ACQUIRE;
        } else if (dr_info.posterror) {
            fprintf(l_dbgFd, "\t** error phase **\n");
            l_currentPhase = POST_ACQUIRE_ERROR;
        }
    }

    /* remove operations. */

```

```

if (dr_info.rem) {
    /* #E */
    fprintf(l_dbgFd, "A remove request\n");

    /* now determine which remove phase we are in. */
    if (dr_info.check) {
        fprintf(l_dbgFd, "\t** check phase **\n");
        l_currentPhase = CHECK_RELEASE;
    } else if (dr_info.pre) {
        fprintf(l_dbgFd, "\t** pre phase **\n");
        l_currentPhase = PRE_RELEASE;
    } else if (dr_info.post) {
        fprintf(l_dbgFd, "\t** post phase **\n");
        l_currentPhase = POST_RELEASE;
    } else if (dr_info.posterror) {
        fprintf(l_dbgFd, "\t** error phase **\n");
        l_currentPhase = POST_RELEASE_ERROR;
    }
}

/* #3 invoke the command associated with the resource. */

/* cpu resource. */
if (dr_info.cpu) {
    /* #F */
    fprintf(l_dbgFd, "Resource is CPU .\n");
    fprintf(l_dbgFd, "\tlogical CPU ID = %d\n\tBind CPU ID = %d\n",
        , dr_info.lcpu, dr_info.bcpu);

    /* invoke the command to process a cpu DR event */
    l_rc = l_currentPhase->cpu_ptr();
    /* #H */

/* memory resource. */
} else if (dr_info.mem) {
    /* #G */
    fprintf(l_dbgFd, "Resource is Memory.\n");
    fprintf(l_dbgFd, "\trequested memory size (in bytes) = %lld\n",
        , dr_info.req_memsz_change);
    fprintf(l_dbgFd, "\tsystem memory size = %lld\n", dr_info.sys_memsz);
    fprintf(l_dbgFd, "\tnumber of free frames in system = %lld\n",
        , dr_info.sys_free_frames);
    fprintf(l_dbgFd, "\tnumber of pinnable frams in system = %lld\n",
        , dr_info.sys_pinnable_frames);
    fprintf(l_dbgFd, "\ttotal number of frames in system = %lld\n",
        , dr_info.sys_total_frames);

    /* invoke the command to process a mem DR event */
    l_rc = l_currentPhase->mem_ptr();
    /* #I */

/* unknown resource. */
} else {
    fprintf(l_dbgFd, "Unknown resource type.\n");
}

```

```

    }

    /* Check the return code of the DLPAR operation handler. */
    if (l_rc == DR_API_FAIL) {
        fprintf(l_dbgFd, "DLPAR OPERATION failed!\n");

        /* Let the DR manager know we have to fail the DLPAR operation. */
        l_rc = dr_reconfig(DR_EVENT_FAIL, &dr_info);
        if (l_rc != 0) {
            perror_msg("dr_reconfig()", errno, __LINE__);
            exit(1);
        }
    }

    fprintf(l_dbgFd, "---end of signal handler.---\n\n");
    fclose(l_dbgFd);
}

/*=====*/
/* Main application */
/*=====*/

/*=====*/
/* Some Application that is registered for DR signals */
/* */
/* Detailed Description: */
/* This is a sample program that registers the signal handler */
/* function that will response to the SIGRECONFIG signal. */
/* */
/* Output: None */
/* Inputs: None */
/*=====*/
int
main(int argc, char *argv[], char *envp[])
{
    int rc;
    struct sigaction sigact_save, sigact;

    /* Start: register this application for a DR signal. */
    if ((rc = sigemptyset(&sigact.sa_mask)) != 0) {
        perror_msg("sigemptyset()", errno, __LINE__);
        exit(1);
    }
    if ((rc = sigemptyset(&sigact_save.sa_mask)) != 0) {
        perror_msg("sigemptyset()", errno, __LINE__);
        exit(1);
    }

    /* register the signal handler function dr_func. */

```



```
sigact.sa_handler = dr_func;
sigact.sa_flags |= SA_SIGINFO;

if ((rc = sigaction(SIGRECONFIG, &sigact, &sigact_save)) != 0) { /* #A */
    perror_msg("sigaction()", errno, __LINE__);
    exit(1);
}
/* Finish: registered the signal handler. */
while (1) { /* #B */
    ;
    /* your applicaiton logic goes here. */
}

exit(0);
}
```

Dynamic logical partitioning output samples

This appendix provides sample outputs from several debug facilities to be used with dynamic logical partitioning (DLPAR) events. It describes three debug facilities:

- ▶ CPU addition
- ▶ Memory addition
- ▶ Using the AIX error log facility

You can use these facilities to analyze the problem if a DLPAR operation request fails. When writing a DLPAR script or DLPAR-aware application, if the problem persists, then testing the application in a test partition can also help isolate the problem.

By familiarizing yourself with these facilities, you can quickly determine the root cause of a DLPAR operation failure.

Using the syslog facility

The syslog facility records the activity of DLPAR operations when it is configured, as explained in “The syslog facility” on page 207.

CPU addition

Example B-1 shows a sample syslog output when a CPU addition DLPAR operation is successfully performed.

Example: B-1 Sample syslog output for a CPU addition request

```
Jul 27 16:49:51 thimblelp4 syslogd: restart
Jul 27 16:50:25 thimblelp4 DRMGR: ==== Start: CPU addition operation ====
Jul 27 16:50:25 thimblelp4 DRMGR: Cpu: 0x1002 has been unisolated and allocated
Jul 27 16:50:25 thimblelp4 DRMGR: Starting CHECK phase for cpu Add operation.
Jul 27 16:50:25 thimblelp4 DRMGR: Phase CHECK started for scripts,kernel extensions and
applications.
Jul 27 16:50:25 thimblelp4 DRMGR: Starting CHECK phase for Scripts.
Jul 27 16:50:25 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:50:26 thimblelp4 DRMGR: Starting the phase for kernel extensions.
Jul 27 16:50:26 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:50:26 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:50:26 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:50:26 thimblelp4 DRMGR: Starting PRE phase.
Jul 27 16:50:26 thimblelp4 DRMGR: Phase PRE started for scripts,kernel extensions and
applications.
Jul 27 16:50:26 thimblelp4 DRMGR: Starting PRE phase for scripts.
Jul 27 16:50:26 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:50:26 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:50:26 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:50:27 thimblelp4 DRMGR: kernel operations complete
Jul 27 16:50:27 thimblelp4 DRMGR: firmware operations complete
Jul 27 16:50:27 thimblelp4 DRMGR: ODM update complete
Jul 27 16:50:27 thimblelp4 DRMGR: Starting POST phase.
Jul 27 16:50:27 thimblelp4 DRMGR: Phase POST started for scripts,kernel extensions and
applications.
Jul 27 16:50:27 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:50:27 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:50:27 thimblelp4 DRMGR: Starting POST phase for scripts.
Jul 27 16:50:27 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:50:27 thimblelp4 DRMGR: ~~~~ End: CPU addition operation ~~~~
```

CPU removal

Example B-2 shows a sample syslog output when a CPU removal DLPAR operation is successfully performed.

Example: B-2 Sample syslog output for a CPU removal request

```
Jul 27 16:47:58 thimblelp4 syslogd: restart
Jul 27 16:48:08 thimblelp4 DRMGR: ==== Start: CPU Removal operation ====
Jul 27 16:48:08 thimblelp4 DRMGR: Starting CHECK phase for cpu Remove operation.
Jul 27 16:48:08 thimblelp4 DRMGR: Phase CHECK started for scripts,kernel extensions and
applications.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting CHECK phase for Scripts.
Jul 27 16:48:08 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting the phase for kernel extensions.
Jul 27 16:48:08 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:48:08 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting PRE phase.
Jul 27 16:48:08 thimblelp4 DRMGR: Phase PRE started for scripts,kernel extensions and
applications.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting PRE phase for scripts.
Jul 27 16:48:08 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:48:08 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:48:08 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:48:08 thimblelp4 DRMGR: kernel operations complete
Jul 27 16:48:08 thimblelp4 DRMGR: Cpu: 0x1002 has been isolated and unallocated
Jul 27 16:48:08 thimblelp4 DRMGR: Firmware operations complete
Jul 27 16:48:09 thimblelp4 DRMGR: ODM update complete
Jul 27 16:48:09 thimblelp4 DRMGR: Starting POST phase.
Jul 27 16:48:09 thimblelp4 DRMGR: Phase POST started for scripts,kernel extensions and
applications.
Jul 27 16:48:09 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:48:09 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:48:09 thimblelp4 DRMGR: Starting POST phase for scripts.
Jul 27 16:48:09 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:48:09 thimblelp4 DRMGR: ~~~~ End: CPU Removal operation ~~~~
```

Memory addition

Example B-3 shows a sample syslog output when a memory addition DLPAR operation is successfully performed.

Example: B-3 Sample syslog output for a memory addition request

```
Jul 27 16:49:51 thimblelp4 syslogd: restart
Jul 27 16:51:34 thimblelp4 DRMGR: ==== Start: MEM Addition operation ====
Jul 27 16:51:34 thimblelp4 DRMGR: Configured LMB addr: 0x0
Jul 27 16:51:34 thimblelp4 DRMGR: Total Megabytes to add is 0
Jul 27 16:51:34 thimblelp4 DRMGR: Starting CHECK phase for mem Add operation.
Jul 27 16:51:34 thimblelp4 DRMGR: Phase CHECK started for scripts,kernel extensions and
applications.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting CHECK phase for Scripts.
Jul 27 16:51:34 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting the phase for kernel extensions.
Jul 27 16:51:34 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:34 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting PRE phase.
Jul 27 16:51:34 thimblelp4 DRMGR: Phase PRE started for scripts,kernel extensions and
applications.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting PRE phase for scripts.
Jul 27 16:51:34 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:34 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:34 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:35 thimblelp4 DRMGR: ODM operations complete
Jul 27 16:51:35 thimblelp4 DRMGR: Starting POST phase.
Jul 27 16:51:35 thimblelp4 DRMGR: Phase POST started for scripts,kernel extensions and
applications.
Jul 27 16:51:35 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:35 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:35 thimblelp4 DRMGR: Starting POST phase for scripts.
Jul 27 16:51:35 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:35 thimblelp4 DRMGR: ~~~~~ End: DR operation ~~~~~
```

Memory removal

Example B-4 shows a sample syslog output when a memory removal DLPAR operation is successfully performed.

Example: B-4 Sample syslog output for a memory removal request

```
Jul 27 16:49:51 thimblelp4 syslogd: restart
Jul 27 16:51:07 thimblelp4 DRMGR: ==== Start: MEM Removal operation ====
Jul 27 16:51:07 thimblelp4 DRMGR: Starting CHECK phase for mem Remove operation.
Jul 27 16:51:07 thimblelp4 DRMGR: Phase CHECK started for scripts,kernel extensions and
applications.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting CHECK phase for Scripts.
Jul 27 16:51:07 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting the phase for kernel extensions.
Jul 27 16:51:07 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:07 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting PRE phase.
Jul 27 16:51:07 thimblelp4 DRMGR: Phase PRE started for scripts,kernel extensions and
applications.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting PRE phase for scripts.
Jul 27 16:51:07 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:07 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:07 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:08 thimblelp4 DRMGR: LMB index:0xf has been sucessfully removed
Jul 27 16:51:08 thimblelp4 DRMGR: Firmware operations complete
Jul 27 16:51:08 thimblelp4 DRMGR: ODM operations complete
Jul 27 16:51:08 thimblelp4 DRMGR: Starting POST phase.
Jul 27 16:51:08 thimblelp4 DRMGR: Phase POST started for scripts,kernel extensions and
applications.
Jul 27 16:51:08 thimblelp4 DRMGR: Starting the phase for application signal handlers.
Jul 27 16:51:08 thimblelp4 DRMGR: Completed the phase for kernel extensions.
Jul 27 16:51:08 thimblelp4 DRMGR: Starting POST phase for scripts.
Jul 27 16:51:08 thimblelp4 DRMGR: Completed the phase for Scripts.
Jul 27 16:51:08 thimblelp4 DRMGR: ~~~~~ End: DR operation ~~~~~
```

Using the AIX system trace facility

The AIX system trace facility records the detailed activity of DLPAR operations when it is configured, as explained in “AIX system trace facility” on page 208.

Note: The trace hook ID for DLPAR operations is 38F.

CPU addition trace output

Example B-5 shows a sample system trace output when a CPU addition DLPAR operation is successfully performed.

Example: B-5 Sample system trace output for a CPU addition request

ID	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
001	0.000000000	0.000000				TRACE ON channel 0 Mon Sep 23 11:40:52 2002
38F	73.305720033	73305.720033				DYNAMIC RECONFIG: Dr_register: DR
Operation: 0000000000000008 FORCE Option: 0000						
38F	73.305723429	0.003396				DYNAMIC RECONFIG: get_user_data: DR
Operation: 0000000000000008 input data: 000000002FF21EF8						
38F	73.305725392	0.001963				DYNAMIC RECONFIG: Addcpu_validate: DR
Phase: 0000 Input: F00000002FF3A4D8						
38F	73.305727203	0.001811				DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 40000000						
38F	73.305729377	0.002174				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0003 DR Phase: 0001						
38F	73.305729602	0.000225				DYNAMIC RECONFIG: Addcpu_validate: DR
Phase: 0001 Input: 0000000000FB73C0						
38F	73.306332726	0.603124				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000						
38F	73.306334115	0.001389				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0001						
38F	73.306334355	0.000240				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0001 Requested Phase: 0002 Flags: 0001						
38F	73.306336885	0.002530				DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases						
38F	73.306337096	0.000211				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0003 DR Phase: 0002						
38F	73.306538971	0.201875				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0004 Timeout in secs: 003C Input: 0000000000000000						
38F	73.306539560	0.000589				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0004						
38F	73.306539800	0.000240				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0004						

38F 73.306545297	0.005497	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases		
38F 73.306546257	0.000960	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0001		
38F 73.308395576	1.849319	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000		
38F 73.308396070	0.000494	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000		
38F 73.308536626	0.140556	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000		
38F 73.308537287	0.000661	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0002		
38F 73.308537527	0.000240	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0002		
38F 73.308538189	0.000662	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases		
38F 73.308538414	0.000225	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching		
38F 73.308730538	0.192124	DYNAMIC RECONFIG: Dr_reconfig: Flags: 0001
DR Info: 000000002FF22610 DR Operation: 0008 DR Phase: 0002		
38F 73.308737235	0.006697	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0001		
38F 83.309400202	10000.662967	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000		
38F 83.309402034	0.001832	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0001		
38F 83.309402260	0.000226	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0003 Flags: 0001		
38F 83.309404165	0.001905	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases		
38F 83.309404703	0.000538	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0003 DR Phase: 0003		
38F 83.309406957	0.002254	DYNAMIC RECONFIG: Addcpu_pre: Logical CPU
coming online: 0002		
38F 83.309607727	0.200770	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000		
38F 83.309608258	0.000531	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0002		
38F 83.309608483	0.000225	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0003 Flags: 0002		
38F 83.309608825	0.000342	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases		
38F 83.309610301	0.001476	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching		
38F 83.309738306	0.128005	DYNAMIC RECONFIG: Dr_reconfig: Flags: 0001
DR Info: 000000002FF22610 DR Operation: 0008 DR Phase: 0003		
38F 83.309751780	0.013474	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0001		

```

38F 93.310361840 10000.610060 DYNAMIC RECONFIG: Dr_notify: DR Phase: 0005
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000
38F 93.310363963 0.002123 DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0005 Flags: 0001
38F 93.310364254 0.000291 DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0005 Flags: 0001
38F 93.310365163 0.000909 DYNAMIC RECONFIG: Kernel_notify: Perform DR
Kernel Phase
38F 93.310365454 0.000291 DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0003 DR Phase: 0005
38F 93.310367432 0.001978 DYNAMIC RECONFIG: Addcpu_doit: Logical CPU:
0002 Physical ID: 0015
38F 93.310371860 0.004428 DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 80000008
38F 93.310372747 0.000887 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0002
38F 93.310394779 0.022032 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000
38F 93.310395245 0.000466 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000
38F 93.310416484 0.021239 DYNAMIC RECONFIG: Call MPC freeze handler
01
38F 93.310445504 0.029020 DYNAMIC RECONFIG: Start_bs_proc: Starting a
new cpu: Physical ID: 0015 Gserver: 00000000000000FF Server: 0000000000000015
38F 93.991532779 681.087275 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0004
38F 93.991536952 0.004173 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000
38F 93.991537709 0.000757 DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000
38F 93.991556098 0.018389 DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 80000008
38F 94.015313247 23.757149 DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000
38F 94.015314549 0.001302 DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0001
38F 94.015314905 0.000356 DYNAMIC RECONFIG: validate_phase: Current
Phase: 0005 Requested Phase: 0006 Flags: 0001
38F 94.015315930 0.001025 DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases
38F 94.015316366 0.000436 DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0003 DR Phase: 0006
38F 94.015382834 0.066468 DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000
38F 94.015383336 0.000502 DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0002
38F 94.015383561 0.000225 DYNAMIC RECONFIG: validate_phase: Current
Phase: 0006 Requested Phase: 0006 Flags: 0002

```

38F	94.015383903	0.000342	DYNAMIC RECONFIG: Run_notify: Perform DR
	Check/Pre/Post/Posterror Phases		
38F	94.015384768	0.000865	DYNAMIC RECONFIG: dr_send_signal: Posting
	signal (003A) to all processes catching		
38F	94.015521608	0.136840	DYNAMIC RECONFIG: Dr_reconfig: Flags: 0001
	DR Info: 000000002FF22610 DR Operation: 0008 DR Phase: 0006		
38F	94.015556140	0.034532	DYNAMIC RECONFIG: dr_send_signal: Number of
	processes posted: 0001		
38F	104.016466326	10000.910186	DYNAMIC RECONFIG: Dr_unregister:
	Unregistering DR operation		
38F	104.016468027	0.001701	DYNAMIC RECONFIG: dr_callout: DR Callout
	index: 0003 DR Phase: 0009		
38F	104.016468507	0.000480	DYNAMIC RECONFIG: Clearing DR Kernel
	Data...		
38F	104.016470936	0.002429	DYNAMIC RECONFIG: Unregister_dr_event: DR
	Operation: 40000000		
002	322.179517597	218163.046661	TRACE OFF channel 0000 Mon Sep 23 11:46:14
	2002		

CPU removal trace output

Example B-6 shows a sample system trace output when a CPU removal DLPAR operation is successfully performed.

Example: B-6 Sample system trace output for a CPU removal request

D	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
001	0.000000000	0.000000				TRACE ON channel 0 Sat Jul 27 17:22:09 2002
38F	8.210889322	8210.889322				DYNAMIC RECONFIG: Dr_register: DR
	Operation: 0000000000000004 FORCE Option: 0000					
38F	8.210890417	0.001095				DYNAMIC RECONFIG: get_user_data: DR
	Operation: 0000000000000004 input data: 000000002FF22AF0					
38F	8.210892128	0.001711				DYNAMIC RECONFIG: Rmcpu_validate: DR Phase:
	0000 CPU id: 0001 CPU Type: 0002					
38F	8.210892971	0.000843				DYNAMIC RECONFIG: Register_dr_event: DR
	Operation: 40000000					
38F	8.210896029	0.003058				DYNAMIC RECONFIG: dr_callout: DR Callout
	index: 0002 DR Phase: 0001					
38F	8.210896238	0.000209				DYNAMIC RECONFIG: Rmcpu_validate: DR Phase:
	0001 CPU id: 0001 CPU Type: 0002					
38F	8.212724871	1.828633				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
	Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000					
38F	8.212725498	0.000627				DYNAMIC RECONFIG: Validate_notify: DR
	Phase: 0002 Flags: 0001					
38F	8.212726175	0.000677				DYNAMIC RECONFIG: validate_phase: Current
	Phase: 0001 Requested Phase: 0002 Flags: 0001					

38F	8.212728618	0.002443	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.212728821	0.000203	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0002 DR Phase: 0002			
38F	8.212836663	0.107842	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0004 Timeout in secs: 003C Input: 0000000000000000			
38F	8.212837119	0.000456	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0004			
38F	8.212837322	0.000203	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0004			
38F	8.212842109	0.004787	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.212842392	0.000283	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0010			
38F	8.212843444	0.001052	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	8.212844231	0.000787	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			
38F	8.212914941	0.070710	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.212915409	0.000468	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0002			
38F	8.212915605	0.000196	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0002			
38F	8.212915858	0.000253	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.212916116	0.000258	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	8.212995181	0.079065	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	8.213781181	0.786000	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.213781637	0.000456	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0001			
38F	8.213781858	0.000221	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0003 Flags: 0001			
38F	8.213782153	0.000295	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.213782332	0.000179	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0002 DR Phase: 0003			
38F	8.213783243	0.000911	DYNAMIC RECONFIG: Rmcpu_pre: DR Phase: 0003
Logical CPU id: 0001			
38F	8.213980321	0.197078	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.213980795	0.000474	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0002			
38F	8.213980992	0.000197	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0003 Flags: 0002			

38F	8.213981262	0.000270	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.213981515	0.000253	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	8.214019626	0.038111	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	8.214186223	0.166597	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0005
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.214186709	0.000486	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0005 Flags: 0001			
38F	8.214187109	0.000400	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0005 Flags: 0001			
38F	8.214187429	0.000320	DYNAMIC RECONFIG: Kernel_notify: Perform DR
Kernel Phase			
38F	8.214187601	0.000172	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0002 DR Phase: 0005			
38F	8.214189318	0.001717	DYNAMIC RECONFIG: Rmcpu_doit: DR Phase:
0005 CPU Guard Operation: 0000			
38F	8.214192929	0.003611	DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 80000004			
38F	8.214193618	0.000689	DYNAMIC RECONFIG: Rmcpu_doit: Controlling
LCPU: 0000 Highest Bind cpuid: 0001			
38F	8.214213369	0.019751	DYNAMIC RECONFIG: Rmcpu_doit: Invoke HA
Handlers...			
38F	8.218559640	4.346271	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0020			
38F	8.218561874	0.002234	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	8.218562286	0.000412	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			
38F	8.218562489	0.000203	DYNAMIC RECONFIG: Migrating all
PROCESSOR_CLASS_ANY work from the cpu being removed			
38F	8.218778851	0.216362	DYNAMIC RECONFIG: Initializing/Rerouting
the Interrupts... From Physical CPU: 0013 To Physical CPU: 0011 Phase: 0001 Flags: 0001			
38F	8.218966349	0.187498	DYNAMIC RECONFIG: Rmcpu_doit: Disable
Decrementer...			
38F	8.218980956	0.014607	DYNAMIC RECONFIG: Call MPC remove handler
01			
38F	8.218982174	0.001218	DYNAMIC RECONFIG: Initializing/Rerouting
the Interrupts... From Physical CPU: 0013 To Physical CPU: 0011 Phase: 0002 Flags: 0002			
38F	8.218982470	0.000296	DYNAMIC RECONFIG: Rmcpu_doit: Enable
Decrementer...			
38F	8.225044879	6.062409	DYNAMIC RECONFIG: DR: Stopping logical CPU:
0001			
38F	8.226081506	1.036627	DYNAMIC RECONFIG: Updating System
Topology...			
38F	8.226114591	0.033085	DYNAMIC RECONFIG: Move_threads: Moving
threads from logical cpu 0001 to 0000			

38F	8.226234647	0.120056	DYNAMIC RECONFIG: migrate_watchdogs: From
LCPU: 0001 To LCPU: 0000			
38F	8.226243790	0.009143	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0040			
38F	8.226244399	0.000609	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	8.226244916	0.000517	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			
38F	8.226245137	0.000221	DYNAMIC RECONFIG: Rmcpu_doit: DR CPU
Removal: CPU Guard: 0000 Status: 0000			
38F	8.226245839	0.000702	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 80000004			
38F	8.407728629	181.482790	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.407729373	0.000744	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0001			
38F	8.407729650	0.000277	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0005 Requested Phase: 0006 Flags: 0001			
38F	8.407731693	0.002043	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.407731878	0.000185	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0002 DR Phase: 0006			
38F	8.407907882	0.176004	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	8.407908350	0.000468	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0002			
38F	8.407908553	0.000203	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0006 Requested Phase: 0006 Flags: 0002			
38F	8.407908848	0.000295	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	8.407909297	0.000449	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	8.407989156	0.079859	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	8.409821659	1.832503	DYNAMIC RECONFIG: Dr_unregister:
Unregistering DR operation			
38F	8.409823339	0.001680	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0002 DR Phase: 0009			
38F	8.409823727	0.000388	DYNAMIC RECONFIG: Clearing DR Kernel
Data...			
38F	8.409826052	0.002325	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 40000000			
002	10.460418340	2050.592288	TRACE OFF channel 0000 Sat Jul 27 17:22:19
2002			

Memory addition trace output

Example B-7 shows a sample system trace output when a memory addition DLPAR operation is successfully performed.

Example: B-7 Sample system trace output for a memory addition request

ID	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
001	0.000000000	0.000000				TRACE ON channel 0 Sat Jul 27 17:21:13 2002
38F	5.368745028	5368.745028				DYNAMIC RECONFIG: Dr_register: DR
Operation: 0000000000000002 FORCE Option: 0000						
38F	5.368746271	0.001243				DYNAMIC RECONFIG: get_user_data: DR
Operation: 0000000000000002 input data: 000000002FF22988						
38F	5.368748763	0.002492				DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 40000000						
38F	5.368751162	0.002399				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0001						
38F	5.370106721	1.355559				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000						
38F	5.370107638	0.000917				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0001						
38F	5.370108136	0.000498				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0001 Requested Phase: 0002 Flags: 0001						
38F	5.370110819	0.002683				DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases						
38F	5.370111003	0.000184				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0002						
38F	5.370229053	0.118050				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0004 Timeout in secs: 003C Input: 0000000000000000						
38F	5.370229515	0.000462				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0004						
38F	5.370229724	0.000209				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0004						
38F	5.370232068	0.002344				DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases						
38F	5.370232345	0.000277				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0100						
38F	5.370234357	0.002012				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000						
38F	5.370234806	0.000449				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000						
38F	5.370313004	0.078198				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000						
38F	5.370313490	0.000486				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0002						
38F	5.370313705	0.000215				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0002						

38F	5.370314013	0.000308	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	5.370314241	0.000228	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	5.370385443	0.071202	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	5.371572355	1.186912	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	5.371572823	0.000468	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0001			
38F	5.371573063	0.000240	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0003 Flags: 0001			
38F	5.371573352	0.000289	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	5.371573555	0.000203	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0003			
38F	5.371688418	0.114863	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	5.371688892	0.000474	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0002			
38F	5.371689199	0.000307	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0003 Flags: 0002			
38F	5.371689446	0.000247	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	5.371689679	0.000233	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	5.371726129	0.036450	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	5.373202390	1.476261	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0005
Flags: 0009 Timeout in secs: 0000 Input: 000000002FF22988			
38F	5.373202857	0.000467	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0005 Flags: 0009			
38F	5.373203134	0.000277	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0005 Flags: 0009			
38F	5.373203430	0.000296	DYNAMIC RECONFIG: get_user_data: DR
Operation: 0000000000000002 input data: 000000002FF22988			
38F	5.373203774	0.000344	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0004			
38F	5.373204199	0.000425	DYNAMIC RECONFIG: Kernel_notify: Perform DR
Kernel Phase			
38F	5.373204359	0.000160	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0005			
38F	5.373205799	0.001440	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0200			
38F	5.373206199	0.000400	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	5.373206666	0.000467	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			

38F	5.373212290	0.005624	DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 80000002			
38F	5.432293377	59.081087	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 80000002			
38F	5.432298927	0.005550	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 0400			
38F	5.432309307	0.010380	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	5.432309707	0.000400	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			
38F	5.533246164	100.936457	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	5.533247161	0.000997	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0001			
38F	5.533248219	0.001058	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0005 Requested Phase: 0006 Flags: 0001			
38F	5.533250176	0.001957	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	5.533250348	0.000172	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0006			
38F	5.533429817	0.179469	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	5.533430291	0.000474	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0002			
38F	5.533430518	0.000227	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0006 Requested Phase: 0006 Flags: 0002			
38F	5.533430789	0.000271	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	5.533431349	0.000560	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	5.533512937	0.081588	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	5.535515956	2.003019	DYNAMIC RECONFIG: Dr_unregister:
Unregistering DR operation			
38F	5.535517217	0.001261	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0001 DR Phase: 0009			
38F	5.535517777	0.000560	DYNAMIC RECONFIG: Clearing DR Kernel
Data...			
38F	5.535520023	0.002246	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 40000000			
38F	5.548827288	13.307265	DYNAMIC RECONFIG: HA_proc: Checking with
Kernel for BAD CPU: Input: 0001 Event: 0000000000000001 Retry: 0000000000000000			
002	7.719713425	2170.886137	TRACE OFF channel 0000 Sat Jul 27 17:21:21
2002			

Memory removal trace output

Example B-8 shows a sample system trace output when a memory removal DLPAR operation is successfully performed.

Example: B-8 Sample system trace output for a memory removal request

ID	ELAPSED_SEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
001	0.000000000	0.000000				TRACE ON channel 0 Sat Jul 27 17:20:16 2002
38F	7.821123474	7821.123474				DYNAMIC RECONFIG: Dr_register: DR
Operation: 0000000000000001 FORCE Option: 0000						
38F	7.821125437	0.001963				DYNAMIC RECONFIG: get_user_data: DR
Operation: 0000000000000001 input data: 000000002FF22970						
38F	7.821127517	0.002080				DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 40000000						
38F	7.821128637	0.001120				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0001						
38F	7.822487468	1.358831				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000						
38F	7.822488219	0.000751				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0001						
38F	7.822488601	0.000382				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0001 Requested Phase: 0002 Flags: 0001						
38F	7.822489610	0.001009				DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases						
38F	7.822489813	0.000203				DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0002						
38F	7.822603894	0.114081				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0004 Timeout in secs: 003C Input: 0000000000000000						
38F	7.822604356	0.000462				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0004						
38F	7.822604534	0.000178				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0004						
38F	7.822608060	0.003526				DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases						
38F	7.822608417	0.000357				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 1000						
38F	7.822610244	0.001827				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000						
38F	7.822610736	0.000492				DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000						
38F	7.822686190	0.075454				DYNAMIC RECONFIG: Dr_notify: DR Phase: 0002
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000						
38F	7.822687070	0.000880				DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0002 Flags: 0002						
38F	7.822687316	0.000246				DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0002 Flags: 0002						

38F	7.822687629	0.000313	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	7.822687863	0.000234	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	7.822772761	0.084898	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	7.824002622	1.229861	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	7.824003040	0.000418	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0001			
38F	7.824003268	0.000228	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0002 Requested Phase: 0003 Flags: 0001			
38F	7.824003612	0.000344	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	7.824003858	0.000246	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0003			
38F	7.824117669	0.113811	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0003
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	7.824118088	0.000419	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0003 Flags: 0002			
38F	7.824118328	0.000240	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0003 Flags: 0002			
38F	7.824118592	0.000264	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	7.824118832	0.000240	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	7.824154322	0.035490	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	7.825608291	1.453969	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0005
Flags: 0009 Timeout in secs: 0000 Input: 000000002FF22970			
38F	7.825608752	0.000461	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0005 Flags: 0009			
38F	7.825608961	0.000209	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0003 Requested Phase: 0005 Flags: 0009			
38F	7.825609287	0.000326	DYNAMIC RECONFIG: get_user_data: DR
Operation: 0000000000000001 input data: 000000002FF22970			
38F	7.825609632	0.000345	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0004			
38F	7.825610013	0.000381	DYNAMIC RECONFIG: Kernel_notify: Perform DR
Kernel Phase			
38F	7.825610167	0.000154	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0005			
38F	7.825611625	0.001458	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 2000			
38F	7.825617182	0.005557	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	7.825617643	0.000461	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			

38F	7.825622836	0.005193	DYNAMIC RECONFIG: Register_dr_event: DR
Operation: 80000001			
38F	7.909427355	83.804519	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 80000001			
38F	7.909764006	0.336651	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
action: 4000			
38F	7.909765157	0.001151	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
Number of reconfig handlers waiting for: 0000			
38F	7.909765551	0.000394	DYNAMIC RECONFIG: Invoke Reconfig Handlers:
All reconfig handlers completed, Status: 0000			
38F	9.184001504	1274.235953	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0001 Timeout in secs: 0000 Input: 0000000000000000			
38F	9.184002919	0.001415	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0001			
38F	9.184003412	0.000493	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0005 Requested Phase: 0006 Flags: 0001			
38F	9.184007042	0.003630	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	9.184007214	0.000172	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0006			
38F	9.184189925	0.182711	DYNAMIC RECONFIG: Dr_notify: DR Phase: 0006
Flags: 0002 Timeout in secs: 0000 Input: 0000000000000000			
38F	9.184190528	0.000603	DYNAMIC RECONFIG: Validate_notify: DR
Phase: 0006 Flags: 0002			
38F	9.184190756	0.000228	DYNAMIC RECONFIG: validate_phase: Current
Phase: 0006 Requested Phase: 0006 Flags: 0002			
38F	9.184191051	0.000295	DYNAMIC RECONFIG: Run_notify: Perform DR
Check/Pre/Post/Posterror Phases			
38F	9.184192196	0.001145	DYNAMIC RECONFIG: dr_send_signal: Posting
signal (003A) to all processes catching			
38F	9.184340075	0.147879	DYNAMIC RECONFIG: dr_send_signal: Number of
processes posted: 0000			
38F	9.186468805	2.128730	DYNAMIC RECONFIG: Dr_unregister:
Unregistering DR operation			
38F	9.186471334	0.002529	DYNAMIC RECONFIG: dr_callout: DR Callout
index: 0000 DR Phase: 0009			
38F	9.186472743	0.001409	DYNAMIC RECONFIG: Clearing DR Kernel
Data...			
38F	9.186475444	0.002701	DYNAMIC RECONFIG: Unregister_dr_event: DR
Operation: 40000000			
38F	9.201400247	14.924803	DYNAMIC RECONFIG: HA_proc: Checking with
Kernel for BAD CPU: Input: 0001 Event: 0000000000000001 Retry: 0000000000000000			
002	11.867866488	2666.466241	TRACE OFF channel 0000 Sat Jul 27 17:20:28
2002			

Using the AIX error log facility

AIX generates an error log entry when a DLPAR operation fails due to a kernel, kernel extension, or other platform failures. The following examples provide sample error log entries:

- ▶ Example B-9
- ▶ Example B-10 on page 292
- ▶ Example B-11 on page 293

See Table 5-14 on page 211 for further detailed information about these error log entries.

Example: B-9 Sample AIX error log entry: DR_MEM_UNSAFE_USE

LABEL: DR_MEM_UNSAFE_USE
IDENTIFIER: 12337A8D

Date/Time: Fri May 24 07:47:39 CDT
Sequence Number: 637
Machine Id: 003579124C00
Node Id: thimblelp4
Class: S
Type: TEMP
Resource Name: DR_KER_MEM

Description
Affected memory not available for DR removal

Probable Causes
Kernel extension not DR aware

Failure Causes
Memory marked as non removable

Recommended Actions
Contact kernel extension owner

Detail Data
Return Code
114
Memory Address
0000 0000 6927 2000
LR Value
0000 0000 0010 30DC
Module Name
/usr/lib/drivers/testmod

Example: B-10 Sample AIX error log entry: DR_DMA_MIGRATE_FAIL

LABEL: DR_DMA_MIGRATE_FAIL
IDENTIFIER: 4DA8FE60

Date/Time: Fri May 24 04:10:29 CDT
Sequence Number: 622
Machine Id: 003579124C00
Node Id: thimblelp4
Class: S
Type: TEMP
Resource Name: DR_KER_MEM

Description
Memory related DR operation failed

Probable Causes
DMA activity to memory being removed

Failure Causes
DMA specific memory migration failed

Recommended Actions
Quiesce the device causing DMA to the memory

Detail Data
Return Code
0 2
Memory Address
0000 0003 FF11 1000
Hypervisor return code
-2
LIOBN
0000 0008
DMA Address
0000 0000 0000 0000 0080 C000

Example: B-11 Sample AIX error log entry: DR_DMA_MAPPAER_FAIL

LABEL: DR_DMA_MAPPER_FAIL
IDENTIFIER: 268DA6A3

Date/Time: Fri May 24 04:10:29 CDT
Sequence Number: 621
Machine Id: 003579124C00
Node Id: thimblelp4
Class: S
Type: TEMP
Resource Name: DR_KER_MEM

Description
Memory related DR operation failed

Probable Causes
DMA Mapper DR handler failure

Failure Causes
DMA specific memory mapper failed

Recommended Actions
Try DR operation on other memory resources

Detail Data
Return Code
4 -16
Memory Address
0000 0000 4096 A000
Handler Address
0000 0000 01F2 A1A4
Module Name
/usr/lib/drivers/pci_busdd

Abbreviations and acronyms

ABI	Application Binary Interface	CLIO/S	Client Input/Output Sockets
ACL	Access Control List	CLVM	Concurrent LVM
AIO	Asynchronous I/O	CMD	Command
AIX	Advanced Interactive Executive	CMOS	Complimentary Metal-Oxide Semiconductor
ANSI	American National Standards Institute	CMP	Certificate Management Protocol
APAR	Authorized Program Analysis Report	CPU	Central Processing Unit
API	Application Programming Interface	CSM	Cluster Systems Management
ARP	Address Resolution Protocol	CSR	Customer Service Representative
ASCII	American National Standards Code for Information Interchange	CSS	Communication Subsystems Support
ASMI	Advanced Systems Management Interface	CST	Central Standard Time
ASR	Address Space Register	CSU	Customer Set-Up
BOOTP	Boot Protocol	CUoD	Capacity Upgrade on Demand
BOS	Base Operating System	CWS	Control Workstation
BPF	Berkeley Packet Filter	DASD	Direct Access Storage Device
BSC	Binary Synchronous Communications	DAT	Digital Audio Tape
BSD	Berkeley Software Distribution	DCM	Dual Chip Module
CD	Compact Disk	DCUoD	Dynamic Capacity Upgrade on Demand
CD-R	CD Recordable	DDR	Double Data Rate
CD-ROM	Compact Disk-Read Only Memory	DDS	Digital Data Storage
CDT	Central Daylight Time	DES	Data Encryption Standard
CE	Customer Engineer	DHCP	Dynamic Host Configuration Protocol
CEC	Central Electronics Complex	DIMM	Dual In-Line Memory Module
CGE	Common Graphics Environment	DLPAR	Dynamic LPAR
CHRP	Common Hardware Reference Platform	DMA	Direct Memory Access
		DVD	Digital Versatile Disk
		EC	Engineering Change

ECC	Error Checking and Correcting	IEEE	Institute of Electrical and Electronics Engineers
EEH	Extended Error Handling	IOCTL	I/O Control
EEPROM	Electrically Erasable Programmable Read Only Memory	IOCLI	I/O Command Line Interface
		IP	Internetwork Protocol (OSI)
EFI	Extensible Firmware Interface	IPL	Initial Program Load
ENV	Environment	IPSec	IP Security
FAQ	Frequently Asked Questions	ITSO	International Technical Support Organization
FC	Fibre Channel		
FCAL	Fibre Channel Arbitrated Loop	JBOD	Just a Bunch of Disks
FCC	Federal Communication Commission	KB	Kilobyte
		KDB	Kernel Debugger
FCP	Fibre Channel Protocol	L1	Level 1
FDDI	Fiber Distributed Data Interface	L2	Level 2
		L3	Level 3
FDPR	Feedback Directed Program Restructuring	LACP	Link Aggregation Control Protocol
FFDC	First Failure Data Capture	LAN	Local Area Network
FTP	File Transfer Protocol	LED	Light Emitting Diode
GB	Gigabyte	LMB	Logical Memory Block
GID	Group ID	LP	Logical Partition
GPFS	General Parallel File System	LPAR	Logical Partition
GUI	Graphical User Interface	LRDMA	Logical Remote Direct Memory Access
HACMP	High Availability Cluster Multi Processing		
		LUN	Logical Unit Number
HMC	Hardware Management Console	LV	Logical Volume
		LVCB	Logical Volume Control Block
HMT	Hardware Multithreading	LVD	Low Voltage Differential
HostRM	Host Resource Manager	LVM	Logical Volume Manager
HPT	Hardware Page Table	LVT	LPAR Validation Tool
I/O	Input/Output	MAC	Media Access Control
IBM	International Business Machines	MPC-3	Multimedia PC-3
		MTU	Maximum Transmission Unit
ICMP	Internet Control Message Protocol	NDP	Neighbor Discovery Protocol
		NFS	Network File System
ID	Identification	NIC	Numeric Intensive Computing
IDE	Integrated Device Electronics		

NIM	Network Installation Management	RAM	Random Access Memory
NIMOL	NIM on Linux	RAN	Remote Asynchronous Node
NVRAM	Non-Volatile Random Access Memory	RAS	Reliability, Availability, and Serviceability
OS	Operating System	RDISC	ICMP Router Discovery
ODM	Object Data Model	RDP	Router Discovery Protocol
OEM	Original Equipment Manufacture	RFC	Request for Comments
PCI	Peripheral Component Interconnect	RIO	Remote I/O
PCI-X	Peripheral Component Interconnect at 133 MHz	RIP	Routing Information Protocol
PFT	Page Frame Table	RIPL	Remote Initial Program Load
PHB	Processor Host Bridges	RISC	Reduced Instruction-Set Computer
PHYP	POWER Hypervisor	RMC	Resource Monitoring and Control
PIC	Pool Idle Count	ROLTP	Relative Online Transaction Processing
PID	Process ID	RPA	RS/6000 Platform Architecture
PIR	Processor Identification Register	RPC	Remote Procedure Call
PLIC	Partition Licensed Internal Code	RPL	Remote Program Loader
PLM	Partition Load Manager	RPM	Redhat Package Manager
PMB	Physical Memory Block	RSC	RISC Single Chip
PMTU	Path MTU	RSCT	Reliable Scalable Cluster Technology
POSIX	Portable Operating Interface for Computing Environments	RSE	Register Stack Engine
POST	Power-On Self-test	RSVP	Resource Reservation Protocol
POWER	Performance Optimization with Enhanced Risc (Architecture)	RTC	Real-Time Clock
PTE	Page Table Entry	RVSD	Recoverable Virtual Shared Disk
PURR	Processor Utilization Resource Register	SA	Secure Association
PV	Physical Volume	SACK	Selective Acknowledgments
PVID	Physical Volume Identifier	SAN	Storage Area Network
QoS	Quality of Service	SAR	Solutions Assurance Review
RAID	Redundant Array of Independent Disks	SASL	Simple Authentication and Security Layer
		SBCS	Single-Byte Character Support

ScaLAPACK	Scalable Linear Algebra Package	SNMP	Simple Network Management Protocol
SCB	Segment Control Block	SOI	Silicon-on-Insulator
SCSI	Small Computer System Interface	SP	IBM RS/6000 Scalable POWER parallel Systems
SCSI-SE	SCSI-Single Ended	SP	Service Processor
SDK	Software Development Kit	SPCN	System Power Control Network
SDLC	Synchronous Data Link Control	SPEC	System Performance Evaluation Cooperative
SDR	System Data Repository	SPI	Security Parameter Index
SDRAM	Synchronous Dynamic Random Access Memory	SPM	System Performance Measurement
SE	Single Ended	SPOT	Shared Product Object Tree
SEPBU	Scalable Electrical Power Base Unit	SPS	SP Switch
SGI	Silicon Graphics Incorporated	SPS-8	Eight-Port SP Switch
SGID	Set Group ID	SRC	System Resource Controller
SHLAP	Shared Library Assistant Process	SRN	Service Request Number
SID	Segment ID	SSA	Serial Storage Architecture
SIT	Simple Internet Transition	SSC	System Support Controller
SKIP	Simple Key Management for IP	SSL	Secure Socket Layer
SLB	Segment Lookaside Buffer	STFDU	Store Float Double with Update
SLIH	Second Level Interrupt Handler	STP	Shielded Twisted Pair
SLIP	Serial Line Internet Protocol	SUID	Set User ID
SLR1	Single-Channel Linear Recording 1	SUP	Software Update Protocol
SM	Session Management	SVC	Switch Virtual Circuit
SMB	Server Message Block	SVC	Supervisor or System Call
SMIT	System Management Interface Tool	SWVPD	Software Vital Product Data
SMP	Symmetric Multiprocessor	SYNC	Synchronization
SMS	System Management Services	TCB	Trusted Computing Base
SNG	Secured Network Gateway	TCE	Translate Control Entry
SNIA	Storage Networking Industry Association	Tcl	Tool Command Language
		TCP/IP	Transmission Control Protocol/Internet Protocol
		TCQ	Tagged Command Queuing
		TGT	Ticket Granting Ticket

TLB	Translation Lookaside Buffer	VLAN	Virtual Local Area Network
TLS	Transport Layer Security	VMM	Virtual Memory Manager
TOS	Type Of Service	VP	Virtual Processor
TPC	Transaction Processing Council	VPD	Vital Product Data
TPP	Toward Peak Performance	VPN	Virtual Private Network
TSE	Text Search Engine	VSD	Virtual Shared Disk
TSE	Text Search Engine	VSM	Visual System Manager
TTL	Time To Live	VSS	Versatile Storage Server™
UCS	Universal Coded Character Set	VT	Visualization Tool
UDB EEE	Universal Database and Enterprise Extended Edition	WAN	Wide Area Network
UDF	Universal Disk Format	WBEM	Web-based Enterprise Management
UDI	Uniform Device Interface	WLM	Workload Manager
UIL	User Interface Language	WTE	Web Traffic Express
ULS	Universal Language Support	XCOFF	Extended Common Object File Format
UNI	Universal Network Interface	XIE	X Image Extension
UP	Uniprocessor	XIM	X Input Method
USB	Universal Serial Bus	XKB	X Keyboard Extension
USLA	User-Space Loader Assistant	XL F	XL Fortran
UTF	UCS Transformation Format	XML	Extended Markup Language
UTM	Uniform Transfer Model	XOM	X Output Method
UTP	Unshielded Twisted Pair	XPM	X Pixmap
UUCP	UNIX-to-UNIX Communication Protocol	XSSO	Open Single Sign-on Service
VACM	View-based Access Control Model	XTF	Extended Distance Feature
VESA	Video Electronics Standards Association	XVFB	X Virtual Frame Buffer
VFB	Virtual Frame Buffer		
VG	Volume Group		
VGDA	Volume Group Descriptor Area		
VGSA	Volume Group Status Area		
VHDCI	Very High Density Cable Interconnect		
VIPA	Virtual IP Address		

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 303.

- ▶ *Advanced POWER Virtualization on IBM @server p5 servers: Installation and Basic configuration*, SG24-7940
- ▶ *Advanced POWER Virtualization on IBM @server p5 Servers: Architecture and Performance Considerations*, SG24-5768
- ▶ *IBM @server p5-590 and p5-595 System Handbook*, SG24-9119
- ▶ *A Practical Guide for Resource Monitoring and Control*, SG24-6615
- ▶ *Linux Applications on pSeries*, SG24-6033
- ▶ *Managing AIX Server Farms*, SG24-6606

IBM Redpapers

IBM Redpapers are available in softcopy only.

- ▶ *IBM @server p5 520 Technical Overview and Introduction*, REDP-9111
- ▶ *IBM @server p5 550 Technical Overview and Introduction*, REDP-9113
- ▶ *IBM @server p5 570 Technical Overview and Introduction*, REDP-9117
- ▶ *IBM @server OpenPower 720 Technical Overview and Introduction*, REDP-1965

IBM Whitepapers

The IBM white paper, *Dynamic logical partitioning in pSeries servers*, is available at the following address.

<http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/dlpar.pdf>

pSeries and eServer p5 publications

The IBM @server Information Center provides a source of technical information about IBM @server hardware, offering technical documentation to help configure and optimize POWER5 and OpenPower servers.

With AIX 5L Version 5.3, the role of the Information Center has been expanded to provide a standardized and central repository for all relevant AIX and pSeries manuals and documentation.

The AIX Information Center application can be installed from the AIX Documentation CD. It can be installed and used on a local system or installed on a documentation server for intranet use.

For the latest AIX and pSeries information, refer to the AIX Information Center web site at:

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

For the latest IBM @server information, refer to the Information Center Web site at:

<http://publib.boulder.ibm.com/eserver/>

LPAR Validation Tool

The LPAR Validation Tool is available to assist you in the design of an LPAR system and to provide a validation report. It is intended for professionals experienced in hardware configuration.

<http://www.ibm.com/servers/eserver/series/lpar/systemdesign.htm>

Other publications

These publications are also relevant as further information sources:

- *The PowerPC Architecture, IBM, Morgan Kaufmann Publishers, Inc.,* ISBN 1-55860-316-6 PB

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ *AIX toolkit for Linux applications*
<http://www.ibm.com/servers/aix/products/aixos/linux/download.html>
- ▶ *IBM @server pSeries & RS/6000 Microcode Updates*
<http://techsupport.services.ibm.com/server/mdownload>
- ▶ *IBM @server pSeries Support Hardware Management Console*
<https://techsupport.services.ibm.com/server/hmc?fetch=home.html>
- ▶ *Electronic Service Agent for pSeries and RS/6000 User's Guide*
ftp://service.software.ibm.com/aix/service_agent_code/AIX/svcUG.pdf
- ▶ *Electronic Service Agent for pSeries HMC User's Guide*
ftp://service.software.ibm.com/aix/service_agent_code/HMC/HMCSAUG.pdf
- ▶ *Microcode Discovery Service*
<http://techsupport.services.ibm.com/server/aix.invscountMDS>
- ▶ *OpenSSH Web site*
<http://www.openssh.com>
- ▶ *VPD Capture Service*
<http://techsupport.services.ibm.com/server/aix.invscountVPD>
- ▶ *OpenPower virtualization*
<http://www.ibm.com/servers/eserver/linux/power/features/virtualization.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Index

Symbols

/usr/lib/dr/scripts/all 186
/usr/samples/dr/scripts 232
/usr/sbin 151
/var/adm/syslog.log 207
_system_configuration.max_ncpus 198
_system_configuration.ncpus 198

A

adapter sharing 127
Admin Override Timeout 171
Advanced Systems Management Interface (ASMI)
 description 64
 HMC access 65
 network configuration 65
 power control 67
 power off warning 66
 service processor 67
 web access 65
advanced virtualization
 feature code FC 7942 6
 feature description 6
AIX error log facility 210, 291
AIX error log messages, DLPAR 212
AIX system trace facility 208
AIX Version 5.3
 lparstat command 144, 226, 229
allow idle processor to be shared 93
API-based DLPAR event handling 157
ARP, Virtual Ethernet 106
ASMI 64

B

bind CPU ID 197
bindprocessor 161, 198
Boot devices 48
building block 3
busy loop 199

C

Capacity on Demand 147
capped mode 88

CD-ROM support, vSCSI 116
cfgdev, command 129
cfgmgr 155
channel ID string 208
chclass, command 161
command line interface, Virtual I/O Server 117
commands 155
 cfgdev 129
 chclass 161
 chhwres 164
 drmgr 151, 157, 160, 162–163, 171, 173–180, 186, 188
 help 118
 kill 161
 lsdev 135
 lsrset 161, 198
 lssrc 150
 mklv 131
 mktcpip 130
 mkvdev 129, 135
 mkvg 131
 ps 161
 rmdev 154
 smit 209–210
 virtual I/O server
 help 118
 whence 151
communication with external networks 101
cpu script 168, 173
current working directory 162

D

database applications 156
Decrementer 227
dedicated memory 21, 86
dedicated processor partitions 35, 86
dedicated processors 91
default channel ID, DRMGR 208
default partition profile 33
detaching pinned shared memory segments 159
detail level 164
DLPAR
 safeness 159

- DLPAR event
 - check phase 152
 - post phase 154
 - pre phase 153
 - specific information 196
- DLPAR operation 148
 - failed message 206
 - failure detailed information 207
- DLPAR script 161
 - database 161
 - naming convention 167
- DLPAR script input
 - additional command line arguments 163
 - environment variables with specified format 163
- DLPAR script output
 - Exit values 163
 - standard out with specified format 163
- DLPAR-aware 155–156
 - kernel extensions 205
- DLPAR-safe 155
- DR_prefix 163
- DR_BCPUID 165
- DR_DATE 170
- DR_DETAIL_LEVEL 164
- DR_DMA_MEM_MAPPER_FAIL 211
- DR_DMA_MEM_MIGRATE_FAIL 211
- DR_ERROR 166
- DR_EVENT_FAIL 196
- DR_FORCE 164
- DR_FREE_FRAMES 166
- dr_ibm_wlm.pl 167
- dr_info
 - add member 200
 - cpu member 200
 - mem member 200
 - rem member 200
- dr_info_t 196
- DR_LCPUID 165
- DR_LOG_DEBUG 167
- DR_LOG_EMERG 166
- DR_LOG_ERR 166
- DR_LOG_INFO 166
- DR_LOG_WARNING 166
- DR_MEM_SIZE_COMPLETED 166
- DR_MEM_SIZE_REQUEST 166
- DR_MEM_UNSAFE_USE 211
- DR_PINNABLE_FRAMES 166
- DR_QUERY 196
- dr_reconfig 195, 199
- DR_RECONFIG_HANDLER_MSG 211
- DR_RESOURCE 173
- DR_SCRIPT_MSG 211
- DR_SCRIPTINFO 170
- dr_sysadmin_wlm.pl 167
- DR_TIMEOUT 171
- DR_TOTAL_FRAMES 166
- DR_USAGE 174
- DR_VENDOR 170
- DR_VERSION 170
- drmgr 163
- drmgr, command 151, 157, 160, 162–163, 171, 173–180, 186, 188
- dynamic partitioning
 - virtual SCSI 116, 147
 - weight 144
- dynamic processor deallocation 158
- dynamic reconfiguration connectors 152

E

- emergency processing 164
- environment values
 - CPU specific 163
 - General 163
 - Memory specific 163
- error analysis facilities 207
- EtherChannel
 - overview 108
 - round robin 109
- Ethernet
 - adapter sharing 109, 126, 131
 - Ethernet adapter on the HMC 127
 - external networks 105
- Event phase summary 160
- execution process 162
- external networks 105, 108
 - routing 105

F

- feature code
 - 1103 - DVD-RAM drive 48
 - 1104 - 8mm VXA-2 tape drive 48
 - 1105 - 4mm DDS4 tape drive kit 48
 - 1106 - DVD-ROM drive 48
 - 1965 - OpenPower 710 VIO/PLM 8
 - 1965 - OpenPower 720 VIO/PLM 8
 - 2498 - PCI Ultra3 RAID adapter 121
 - 2591 - USB diskette drive 48

- 2640 - IDE DVD-ROM drive 47
- 2738 - USB adapter 48
- 2738 - USB keyboard and mouse adapter 52
- 2849 - graphics adapter 52
- 5703 - PCI-X Ultra320 RAID adapter 121
- 5709 - Dual Channel RAID enablement card 121
- 5709 - SCSI RAID enablement card 49
- 5712 - PCI-X Ultra320 SCSI adapter 121
- 5716 - 2 GB FC PCI-X adapter 121
- 5751 - IDE DVD-RAM drive 47
- 5791 - 7040-61D I/O drawer 50
- 6120 - 8mm 80/160 GB tape drive 47
- 6134 - 8mm 60/150GB tape drive 47
- 6228 - 2 GB FC PCI-X adapter - 64-bit 121
- 6239 - 2 GB FC PCI-X adapter 121
- 6258 - 4mm 36/72 GB tape drive 47
- 7310-C04 - desktop HMC 12
- 7310-CR3 - rack-mount HMC 12
- 7432 - p5 510 VIO/PLM 8
- 7940 - p5 520 VIO/PLM 8
- 7941 - p5 550 VIO/PLM 8
- 7942 - advanced virtualization 6
- 7942 - p5 570 VIO/PLM 8
- 7992 - p5 590 VIO/PLM 8
- 7992 - p5 595 VIO/PLM 8
- fileset
 - bos.adt.samples 232
 - bos.rte.methods 151
 - devices.chrp.base.rte 150
- free frames 166

G

- GID 162
- granularity 3

H

- hardware management console, HMC
 - ASMI access 65
 - desktop HMC, feature code FC 7310-C04 12
 - functions 59
 - I/O devices and slots 59
 - memory configuration 61
 - overview 58
 - rack-mount HMC, feature code FC 7310-CR3 12
 - restrictions 58
 - scheduling 64

- hardware page table, HPT 227
- hardware resources
 - virtual I/O Server 120
- hosted partition 112
- hosting partition 112
- hypervisor, POWER hypervisor
 - call functions 221–225
 - description 219
 - extensions 227
 - introduction 29
 - memory considerations 227
 - support 219
 - system resources 229

I

- I/O Server Command Line Interface (IOCLI) 104
- i5/OS 219
- IBM.DRMd daemon process 150
- IEEE 802.1Q VLAN 99
- Information Center 14
- initiator, vSCSI 112
- in-memory channel 102
- installation, Virtual I/O Server 121
- Internal activity in a DLPAR event
 - CPUs and memory 152
 - I/O slots 154
- IP fragmentation 107
- ipcs 161
- ipforwarding 105
- iSeries Partition Licensed Internal Code (PLIC) 219

K

- KDB 212
- kernel debugger 212
- kernel extensions 153
- kernel service
 - switch_cpu 198

L

- Licence Managers 156
- licensed software components 8
- limitations and considerations 116, 136
 - Shared Ethernet Adapter 110
 - virtual Ethernet 103
 - virtual SCSI, vSCSI 116
- Linux 218
- list registered DLPAR scripts 186

- LMB, logical memory block 153
 - minimum size 38
- LOG_DEBUG 167
- LOG_EMERG 166
- LOG_ERR 166
- LOG_INFO 166
- LOG_WARNING 166
- logical CPU ID 197
- logical memory block, LMB 153
 - minimum size 38
- logical partition 3
- logical volume
 - define 131
 - limitations 136
- lparstat command 144, 226, 229
- LRDMA, Logical Remote Direct Memory Access 113
- lsclass 161
- lsdev, command 135
- lspp 151
- lsslot 154
- lssrc, command 150
- LVT (LPAR Validation Tool) 15

M

- MAC address 102, 107
- man 188
- media devices
 - 4mm 36/72 GB tape drive, FC 6258 47
 - 8mm 60/150GB tape drive, FC 6134 47
 - 8mm 80/160 GB tape drive, FC 6120 47
 - IDE DVD-RAM, FC 5751 47
 - IDE DVD-ROM, FC 2640 47
- mem 168, 173
- Micro-Partitioning 35
 - dedicated memory 21, 86
 - dedicated processor partitions 35, 86
 - dedicated processors 91
 - Firmware enablement 6
 - introduction 84
 - overview 86
 - processing capacity 86
 - processing units 86
 - shared processor partition 84
 - virtual processors 89
- mkiv, command 131
- mktcpip, command 130
- mkvdev, command 129, 135

- mkvg, command 131
- MP-unsafe application 180

N

- NDP, virtual Ethernet 106
- NIMOL, NIM on Linux 9
- number of online CPUs 198
- number of potential CPUs 198
- NVRAM 33

O

- ODM 152
 - ODM lock 152
- Open Firmware 218
 - device tree 152
- OpenPower 710
 - capabilities 11
 - VIO/PLM feature code 8
- OpenPower 720 47
 - 7311-D20 support 49
 - boot devices 49
 - redpaper 301
 - VIO/PLM feature code 8
- optional name-value pair 170

P

- p5 510
 - VIO/PLM feature code 8
- p5 520
 - 7311-D20 support 49
 - boot devices 48
 - capabilities 10
 - I/O device assignment considerations 47
 - redpaper 301
 - VIO/PLM feature code 8
- p5 550
 - 7311-D20 support 49
 - boot devices 49
 - capabilities 10
 - I/O device assignment considerations 47
 - redpaper 301
 - VIO/PLM feature code 8
- p5 570
 - 7311-D10 support 50
 - 7311-D11 support 50
 - 7311-D20 support 50
 - boot devices 50

- capabilities 10
- I/O device assignment considerations 47
- redpaper 301
- VIO/PLM feature code 8
- p5 590
 - 7040-61D support 51
 - boot devices 50
 - capabilities 10
 - I/O device assignment considerations 47
 - systems handbook 301
 - VIO/PLM feature code 8
 - virtualization support 86
- p5 595
 - 7040-61D support 51
 - boot devices 50
 - capabilities 10
 - I/O device assignment considerations 47
 - systems handbook 301
 - VIO/PLM feature code 8
 - virtualization support 86
- partition
 - isolation 3
 - physical 3
 - profile 33, 149
 - resources 32
 - I/O slots 40
 - memory 36
 - processors 34
 - system profiles 33
- Partition Load Manager, PLM
 - components 8
 - description 71–72, 74–76
 - feature 6
 - installation 78
 - operation 78–79, 81
 - ordering 6
 - software license charge 8
 - support 9
- partitioning capabilities 83
- PATH environment variable 162
- PCI adapter Hot Plug capability 154
- PCI host bridges 28
- Per processor description area 197
- performance considerations
 - EtherChannel 108
 - hypervisor 229
 - virtual Ethernet 103
 - virtual SCSI 117
- performance monitor support 223
- physical CPU ID 197
- pinnable frames 166
- pipe 162
- platform-dependent commands 151
- platform-independent command 151
- PLM 78–79
 - Partition Load Manager 6, 8–9, 71–72, 74–76, 78–79, 81
- polling 156
- Port virtual LAN ID, PVID 99
- possible causes of DLPAR failures 205
- post phase 159
- POWER Hypervisor 218–230
 - debugger support 222
 - dump support 222
 - extensions for Micro-Partitioning 227
 - Machine Check Interrupt 220
 - memory migration support 222
 - performance monitor support 223
 - System (Hypervisor) Call Interrupt 220
 - System Reset Interrupt 220
 - table of calls 223
 - virtual terminal support 222
- POWER5
 - Decrementer 227
 - Time Base register 227
- ppda, per-processor description area 197
- pre phase 159
- processing
 - capacity 86
 - units 86
- product names, short 10
- programming implications of CPU DLPAR events 197
- ps, command 161
- PVID, Port virtual LAN ID 99

R

- RDMA, Remote Direct Memory Access 113
- reconfig_complete 205
- reconfig_register 205
- reconfig_unregister 205
- Redbooks Web site 303
 - Contact us xix
- register a DLPAR script 186
- Remote Direct Memory Access, RDMA 113
- removing plocks 159
- resource manager

- IBM.DRM 150, 160
- resource monitoring and controlling, RMC 148, 160
- resource set 197
- resource value
 - Desired 41
 - Maximum 41
 - Minimum 41
 - Required 41
- restricted Korn shell 118
- RMC 150
- rmdev, command 154
- root authority 196
- RS/6000 154
- rset 197
- Run-Time Abstraction Services (RTAS) 218

S

- sample code using dr_reconfig 199
- Script Timeout 171
- script-based DLPAR event handling 157
- SCSI RDMA 113
- secure and reliable connection channel 150
- server adapter, vSCSI 112
- Shared Ethernet Adapter
 - limitations and considerations 110
- shared Ethernet Adapter 105
- shared Ethernet adapter 105
 - external networks 105
- shared Ethernet adapters overview 21
- shared processing pool 90
- shared processor partition 84
- sigaction 195, 199
- signal
 - SIGABRT 171
 - SIGKILL 171
- signal handler
 - dr_func() 199
 - registration 199
- SIGRECONFIG 199
- Simultaneous multi-threading (SMT) 27
- smit, command 209–210
- SMT 7
- software license charge 8
- SSA support
 - virtual SCSI, vSCSI 116
- standard in 162
- standard out 162
- subcommand

- checkacquire 180
- checkacquire 169
- checkrelease 168, 175
- postacquire 169, 183
- postrelease 168, 178
- preacquire 169, 182
- prerelease 168, 176
- register 168, 172
- scriptinfo 168–169
- undopreacquire 169, 184
- undoprerelease 168, 179
- usage 168, 173
- sysconf(_SC_NPROCESSORS_CONF) 198
- sysconf(_SC_NPROCESSORS_ONLN) 198
- sysconfig 198
- syslog facility 167, 207
- syslog keyword
 - rotate 207
- syslog priority
 - debug 167
 - emerg 166
 - err 166
 - info 166
 - warning 166
- System Licensed Internal Code (SLIC) 219
- System profile 33

T

- tagged packets 99
- tape support
 - virtual SCSI, vSCSI 116
- target, vSCSI 112
- Technology Independent Machine Interface (TIMI) 219
- three phases in a DLPAR event 157
- Time Base 227
- timeout value 171
- Total number of frames 166
- trace hook ID 209
- Translation Control Entry (TCE) 222
- trunk flag 106
- trunk Virtual Ethernet adapter 126

U

- UID 162
- unbinding processors 159
- uncapped mode 88
- uninstall a registered DLPAR script 187

- uniprocessor 155
- untagged packets 99
- using AIX system trace facility 278
- using syslog facility 274

V

- virtual Ethernet
 - ARP 106
 - benefits 103
 - broadcast 106
 - communication with external networks 101
 - interpartition communication 100
 - limitations and considerations 103
 - multicast 106
 - NDP 106
 - packaging 7
 - performance considerations 103
- virtual Ethernet adapter
 - boot device 103
 - concepts 102
 - in-memory channel 102
 - MAC address 102
 - transmission speed 103
 - trunk flag 106
- virtual host bridge 114
- virtual I/O Server
 - adapter sharing 109, 126, 131
 - capabilities 125
 - command line interface, 117
 - hardware resources 120
 - hosted partition 112
 - hosting partition 112
 - installation 121
 - introduction 104
 - limitations and considerations 136
 - ordering 6
 - restricted Korn shell 118
 - software license charge 8
- virtual I/O server
 - 2 GB FC PCI-X adapter 121
 - 2 GB FC PCI-X adapter - 64-bit 121
 - Dual Channel RAID enablement card 121
 - PCI Ultra3 RAID adapter 121
 - PCI-X Ultra320 RAID adapter 121
 - PCI-X Ultra320 SCSI adapter 121
- virtual LAN
 - AIX support 99
 - description 98

- overview 98
- virtual processor 89, 227
 - move 146
 - reasonable settings 54
- virtual SCSI, vSCSI
 - CD-ROM support 116
 - client adapter, vSCSI 112
 - define client adapter 134
 - define disk 131
 - define server adapter 132
 - description 40
 - device configuration 114
 - dynamic partitioning 116, 147
 - hosting partition 112
 - initiator 112
 - introduction 111
 - limitations and considerations 116
 - overview 21
 - performance considerations 117
 - server adapter 112
 - SSA support 116
 - tape support 116
 - target 112
- virtual target device, create 135
- virtualization
 - media device restriction 48
- virtualized Ethernet 21
- volume group, define 131

W

- weight, uncapped 88
- Workload Manager 156



Partitioning Implementations for IBM *@server* p5 Servers

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Partitioning Implementations for IBM @server p5 Servers

**Discusses Advanced
POWER Virtualization
and Micro-Partitioning
technology**

**Describes virtualization
on IBM @server
OpenPower systems**

**Includes information on
dynamic logical
partitioning**

This redbook provides a broad understanding of logical partitioning on the IBM @server p5 servers, focusing particularly on the increased function available when these servers are combined with AIX 5L Version 5.3 and Advanced POWER Virtualization features. It also provides a discussion of available Linux support and IBM @server OpenPower systems. This redbook covers the following subject areas:

- Advanced POWER Virtualization
- Micro-Partitioning technology
- Virtual I/O
- Dynamic logical partitioning implementation

The audience for this redbook are technical support specialists, customers, and business partners.

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