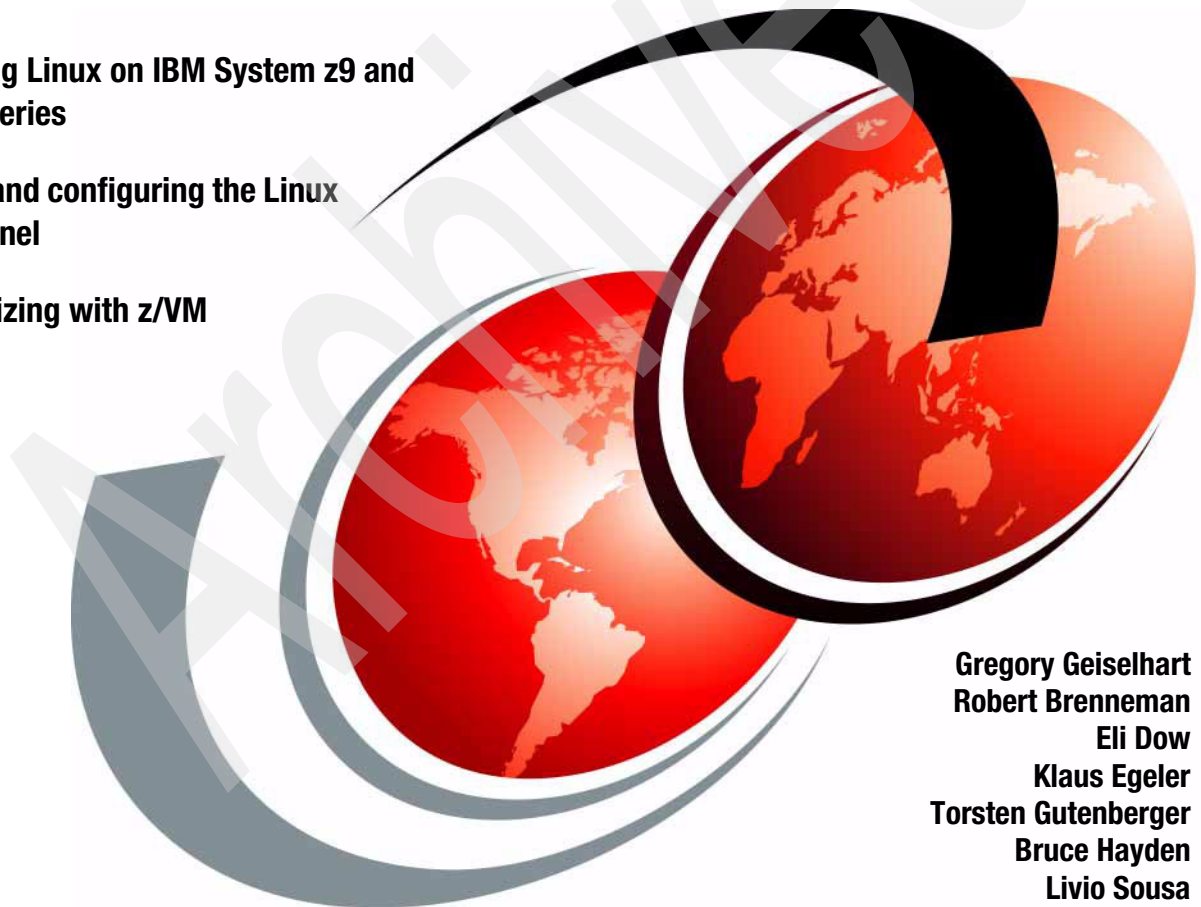


Linux for IBM System z9 and IBM zSeries

Running Linux on IBM System z9 and
IBM zSeries

Using and configuring the Linux
2.6 kernel

Virtualizing with z/VM



Gregory Geiselhart
Robert Brenneman
Eli Dow
Klaus Egeler
Torsten Gutenberger
Bruce Hayden
Livio Sousa



International Technical Support Organization

Linux for IBM System z9 and IBM zSeries

January 2006

Archived

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

First Edition (January 2006)

This edition applies to Version 5, Release 1 of z/VM and multiple distributions of Linux for zSeries based on the 2.6 Linux kernel.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook	ix
Become a published author	x
Comments welcome	xi
Chapter 1. Introduction to z/Architecture	1
1.1 The zSeries and System z9 servers	2
1.2 The central processing complex	2
1.2.1 Processing units	2
1.3 The channel subsystem	2
1.4 Logical partition concepts	4
1.4.1 LPAR dispatching	4
Chapter 2. Linux 2.6 kernel.	9
2.1 Choosing a 64-bit or 31-bit Linux distribution	10
2.1.1 Linux 31-bit compatibility mode	10
2.2 Performance and scalability enhancements	11
2.2.1 The new Linux process scheduler	12
2.2.2 System swap setting	13
2.2.3 Kernel preemption	13
2.2.4 Fast user-space mutual exclusion	13
2.2.5 I/O performance and scalability enhancements	14
2.2.6 I/O schedulers	14
2.2.7 Extended attributes and access control lists	16
2.2.8 Memory management enhancements	17
2.2.9 Native POSIX Threading Library	17
2.2.10 NFS version 4	19
2.3 Introduction to sysfs	20
2.3.1 Organization of the /sys directory	20
2.3.2 New sysfs command for Linux on zSeries	22
2.3.3 Device configuration with sysfs	24
2.3.4 SLES9 hardware configuration	24
2.3.5 RHEL4 hardware configuration	28
2.3.6 Device node naming with udev	32
2.3.7 Hotplug detection and device node naming	33
2.3.8 Distribution considerations when naming devices	36

2.4 S/390 tools and utilities	37
2.4.1 The cmsfsfst command	38
2.4.2 The tunedasd command	38
2.4.3 The dasdview command	39
Chapter 3. Virtualization with z/VM	41
3.1 z/VM and virtualization	42
3.1.1 z/VM guest support	42
3.2 Installing z/VM 5.1	43
3.2.1 Installation to FCP-attached SCSI disk	43
3.3 Distributions for Linux on zSeries	47
3.3.1 SUSE Linux Enterprise Server 9	47
3.3.2 Red Hat Enterprise Linux 4	47
3.3.3 Non-commercial distributions	48
3.3.4 IBM middleware for Linux on zSeries	48
3.4 Running Linux under z/VM	49
3.4.1 Managing guest virtual memory	49
3.4.2 Managing z/VM storage	49
3.4.3 Managing the VM scheduler	53
3.4.4 Performance monitoring	56
Chapter 4. Networking Overview	59
4.1 Networking options	60
4.2 Physical networking options	60
4.2.1 Open Systems Adapter-2 (OSA-2)	60
4.2.2 Open Systems Adapter-Express (OSA-Express)	60
4.2.3 Channel-to-channel adapter	76
4.2.4 Common Link Access to Workstation (CLAW)	82
4.3 HiperSockets	83
4.4 Virtualization technology	91
4.5 Point-to-point connectivity	91
4.5.1 Guest LAN	93
4.6 Guest LAN configuration	96
4.6.1 Create a z/VM Guest LAN	96
4.6.2 Persistent Guest LANs	97
4.6.3 The VMLAN statement	98
4.6.4 Create a virtual Network Interface Card	100
4.6.5 NIC definition in the user directory	100
4.6.6 NIC definition using CP commands	102
4.6.7 Connect the virtual NIC to the Guest LAN	103
4.6.8 Example of building a z/VM Guest LAN	104
4.6.9 Undoing the definitions	106
4.6.10 Configuring Linux to connect to a Guest LAN	107

4.7 Virtual Switch	108
4.8 Introduction to VLANs	109
4.8.1 What is a Virtual LAN	109
4.8.2 VLAN standards	113
4.8.3 How IEEE 802.1Q VLANs work	114
4.8.4 VLAN support on z/VM Guest LAN	119
4.8.5 VLANs on z/VM Virtual Switch	119
4.8.6 VLAN isolation	120
4.9 VSWITCH configuration	121
4.9.1 Transport mode: IP or Ethernet	121
4.9.2 Configure controller service machines	121
4.9.3 Defining a VSWITCH	124
4.9.4 VSWITCH failover support	126
4.10 Layer 2 test scenario	126
4.10.1 Requirements	126
4.10.2 Test overview	128
4.10.3 External devices	132
4.10.4 z/VM system configuration	132
4.10.5 Configuring Linux	136
4.10.6 Setting up the file server	137
4.10.7 NFS server configuration	141
4.10.8 Setting up the Web servers	142
4.10.9 Setting up the firewalls	144
4.10.10 Setting up the load balancer	145
4.10.11 Testing	150
4.10.12 Recommendations	152
4.11 Summary	152
Chapter 5. FCP-attached SCSI disks	155
5.1 FCP configuration using sysfs	156
5.1.1 Querying FCP device information	158
5.2 SCSI device configuration files	159
5.2.1 FCP device configuration in SLES9	160
5.2.2 FCP device configuration in RHEL4	162
5.3 Persistence SCSI disk naming with udev	164
5.3.1 Persistent SCSI disk naming in RHEL4	165
5.3.2 Persistent SCSI disk naming in SLES9	167
5.4 The FCP LUN Access Control feature	168
5.4.1 FCP LUN Access Control operation	170
5.4.2 Installing the FCP LUN Access Control feature	171
5.4.3 Using the configuration utility	172
5.4.4 A sample scenario	173
5.5 Multipathing for SCSI disks	175

5.5.1 Multipathing using EVMS on SLES9.....	176
5.5.2 Multipathing using mdadm on RHEL4.....	184
Chapter 6. Using ECKD storage.....	187
6.1 ESS operating mode.....	188
6.2 Configuring DASD at IPL with SLES9.....	189
6.3 Configuring DASD at IPL with RHEL4.....	189
6.4 Parallel Access Volumes.....	190
6.4.1 Defining PAV devices.....	191
6.4.2 Preparing PAV devices in Linux.....	192
6.4.3 Defining PAV devices with EVMS.....	192
6.4.4 Using PAV on with mdadm.....	198
6.4.5 Using PAV devices with LVM2.....	201
Related publications.....	205
IBM Redbooks.....	205
Other publications.....	206
Online resources.....	207
How to get IBM Redbooks.....	207
Help from IBM.....	207
Index.....	209

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

DB2®
developerWorks®
Domino®
ECKD™
Enterprise Storage Server®
Enterprise Systems
Architecture/370™
Enterprise Systems
Architecture/390®
ESCON®
@server®
eServer®
eServer™

FICON®
HiperSockets™
IBM®
Informix®
Lotus®
Multiprise®
OS/390®
PR/SM™
RACF®
Rational®
Redbooks (logo) ™
Redbooks™
RS/6000®

S/390®
System z9™
System/360™
System/370™
Tivoli®
TotalStorage®
VTAM®
WebSphere®
z/Architecture™
z/OS®
z/VM®
z9™
zSeries®

The following terms are trademarks of other companies:

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

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

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

Preface

This IBM Redbook discusses configuring and administering Linux® systems running on IBM System z9 and zSeries. It is intended for system administrators and IT architects responsible for deploying Linux servers on System z9 and zSeries servers.

We examine new features and enhancements of the Linux 2.6 kernel. We demonstrate Linux device configuration using sysfs and udev and discuss running Linux servers under z/VM. Networking options available for Linux on IBM System z9 and zSeries are examined in detail. Configuration of FCP-attached SCSI disks and traditional ECKD storage is demonstrated.

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, Poughkeepsie Center.

Gregory Geiselhart is a Project Leader for Linux on zSeries at the International Technical Support Organization, Poughkeepsie Center in New York.

Robert Brenneman is a Software Engineer at the Test and Integration Center for Linux in Poughkeepsie, New York.

Eli Dow is a Software Engineer at the Test and Integration Center for Linux in Poughkeepsie, New York. He holds a Bachelor of Science degree along with a Masters of Computer Science from Clarkson University and is the author of numerous IBM developerWorks articles in the subject area of Linux.

Klaus Egeler is an IT Systems Management Specialist with IBM Global Services Germany. He has more than fifteen years of experience as a VSE and VM systems programmer. He has worked with Linux for zSeries and S/390 for more than four years. He has contributed to several Linux related IBM Redbooks.

Torsten Gutenberger is an IT specialist for s/390 and zSeries, working in the EMEA Central Region Hardware Support Center in Mainz, Germany. He holds a degree in IT engineering and has nine years experience in s/390 and zSeries environment, including a three-year assignment to the zSeries EMEA Product Support Group (EPSG) in Montpellier, France.

Bruce Hayden is an I/T Architect in IBM Global Services. He has 22 years of experience as a VM systems programmer including five years working with Linux on zSeries. He holds a degree in Computer Science from the University of Missouri-Rolla. His areas of expertise include z/VM, Rexx, CMS Pipelines, and Linux. He has contributed to previous IBM Redbooks.

Livio Sousa is a Technical Sales Support member of the Linux for zSeries team in Brazil. He holds a degree in Information Technology and has five years of experience in the operating systems field. He has worked for IBM since 2002 responsible for planning, implementation, running and supporting new workload projects. He has contributed to previous IBM Redbooks.

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

David Bennin, Roy Costa, Octavian Lascu
International Technical Support Organization, Poughkeepsie Center

Alan Altmark
IBM Endicott, New York

Bruce Booth
IBM Toronto, Canada

Mike Maclsaac
IBM Poughkeepsie, New York

Simon Williams
IBM Australia

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

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

Archived



Introduction to z/Architecture

In this chapter, we provide an introductory overview of the current zSeries hardware and a high level introduction to some parts of the z/Architecture.

1.1 The zSeries and System z9 servers

The IBM @server zSeries and System z9 are based on z/Architecture - the next step in the evolution from the System/360 to the System/370, System/370 extended architecture (370-XA), Enterprise Systems Architecture/370 (ESA/370), and Enterprise Systems Architecture/390 (ESA/390).

1.2 The central processing complex

This section describes the central processing complex.

1.2.1 Processing units

All processor units (PUs) are physically the same. At Power-on Reset (POR), distinct microcode loaded in the PU determines its processor type:

- ▶ Central Processor (CP)

CPs are enabled to execute any operation system available on zSeries (including z/OS, z/VM, Linux, VSE, and TPF).

- ▶ Integrated Facility for Linux (IFL)

IFLs are processors enabled specifically for Linux workloads. Both z/VM and Linux (either in LPAR mode, or as a z/VM) can execute on an IFL.

- ▶ Integrated Facility for Applications (IFA)

Also known as an IBM zSeries Application Assist Processor (zAAP), an IFA is designed to exclusively execute Java™ applications running under z/OS.

- ▶ Internal Coupling Facility (ICF)

ICFs are provided exclusively for use by the Coupling Facility (CF) in a Parallel Sysplex.

- ▶ System Assist Processor (SAP)

SAPs are specialized processors that run the channel subsystem Licensed Internal Code (LIC) to control I/O operations.

1.3 The channel subsystem

I/O devices are attached to the zSeries system through the channel subsystem (CSS). Components of the channel subsystem include:

▶ Control unit

A *control unit* (CU) provides the logic to operate and control an I/O device. An I/O device can be attached to one or more control units. Control unit function can be:

- Implemented in a separate physical unit
- Integrated inside the I/O device
- Integrated within the channel itself

▶ Channel

A *channel* is a specialized processor that communicates with the control unit. Channels are located in the zSeries I/O cage and manage data movement between main storage and the control unit.

▶ Subchannel

One *subchannel* is dedicated to each I/O device accessed by the channel subsystem. The subchannel provides information about the attached I/O device to the channel subsystem (such as the CHPID, device status, and channel path availability). Subchannels are addressed using the system-unique 16-bit subchannel number. The number of available subchannels depends on the system model, and is limited to a maximum of 65,536 per system.

▶ Channel path

A control unit is attached to the channel subsystem by one or more *channel paths*. Depending on the zSeries model and configuration, an I/O device can be accessed by as many as eight different channel paths. Types of channel paths supported on zSeries include:

- Enterprise Systems Connection (ESCON)
- Fiber Connection (FICON)
- Open Systems Adapter-2 (OSA-2)
- OSA Express

Note: Although supported on the z800 and z900, parallel channels are *not* supported on the z890 and z990.

Channel paths are addressed using the system-unique, eight-bit channel path identifier (CHPID). The actual number of available channel paths depends on the system model, and is limited to a maximum of 256 per system.

▶ SAP

The *SAP* schedules an I/O operation. It finds an available channel path to the intended device and guarantees completion of the I/O operation. However, the SAP does not move data between main storage and the channel.

1.4 Logical partition concepts

Processor Resource/System Manager (PR/SM) is a standard feature of all zSeries CPCs. This allows a CPC to be divided into multiple logical partitions (LPARs). LPARs allow workloads to be isolated in different system images, so you can run production work separately from test work, or even consolidate multiple servers into a single processor.

An LPAR has the following properties:

- ▶ Each LPAR is a set of physical resources (processor, storage, and channels) controlled by an operating system image (such as z/VM, z/OS, or Linux). An LPAR is defined through IOCP/HCD. A Power-on-Reset (POR) is required to add or remove LPARs.
- ▶ LPAR options (such as the number of logical CPs, LPAR weight, LPAR capping, and storage size) are defined in the Activation Profiles on the Hardware Management Console (HMC).
- ▶ Physical processors and channels can be shared between multiple LPARs, or dedicated to single LPAR. In general, sharing processors maximizes zSeries system utilization.
- ▶ Physical storage used by an LPAR is dedicated to the LPAR.

1.4.1 LPAR dispatching

The LPAR Scheduler Licensed Internal Code (LIC) is responsible for dispatching logical CPs on a physical processor. The LPAR scheduler LIC runs on all physical CPs and uses the Start Interpretive Execution (SIE) instruction to dispatch logical CPs. This instruction copies a logical CP's status (PSW, registers, etc.) from the HSA to the physical CP. The physical CP then executes the code specific to the logical CP. When intercepted, the logical CP status is saved in HSA. At this point, another logical CP is dispatched (starting the process over).

Note: A *physical CP* is a physical processor (CP or IFL) on the CEC. A *logical CP* is a processor available to an LPAR for dispatching work. The number of logical CPs assigned to an LPAR must be less than or equal to the number of available shared or dedicated processors.

To illustrate LPAR dispatching, we consider the example in Figure 1-1.

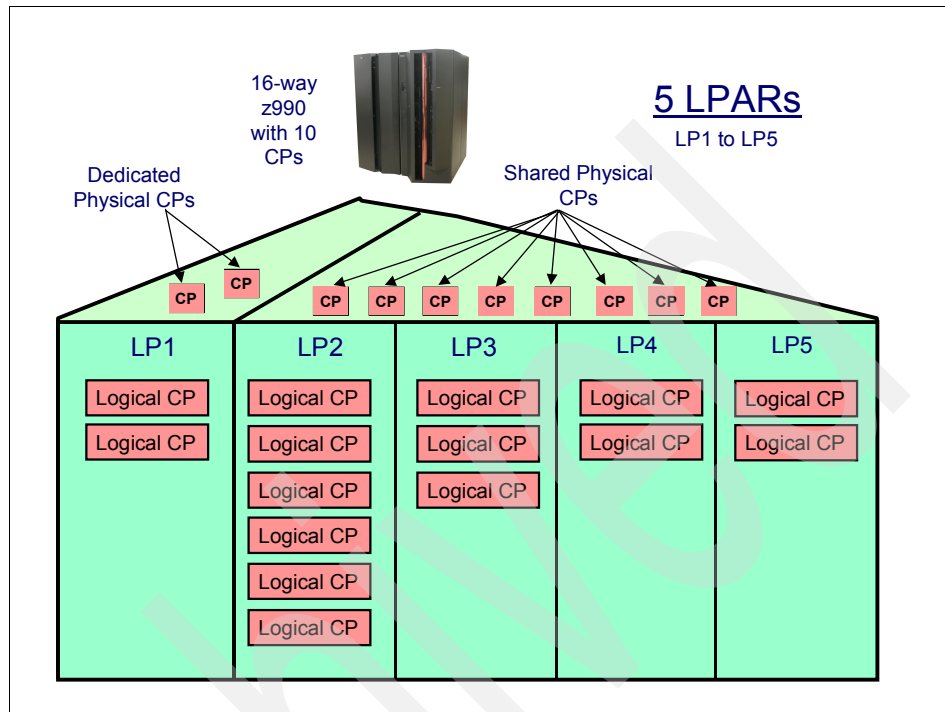


Figure 1-1 LPAR dispatching example scenario

The figure shows a 10-CP IBM zSeries 990 16-way processor configured with five LPARs:

- ▶ LP1 is defined with two dedicated processors.
- ▶ LP2 through LP5 share the remaining eight processors:
 - LP2 is defined with six logical CPs.
 - LP3 is defined with three logical CPs.
 - LP4 and LP5 are each defined with two logical CPs.

This yields a total of 13 logical CPs.

The two dedicated CPs are for use exclusively by LP1. For this LPAR, the number of physical CPs is equal to the number of logical CPs. Any of the remaining eight shared processors can be used by LP2 through LP5.

Every logical CP represents a dispatchable unit of work. When a logical CP is ready to run, it is placed on the logical CP ready queue. This queue is ordered by a priority based on LPAR weight and the number of logical CPs. The steps to dispatch logical CPs are shown in Figure 1-2 on page 6.

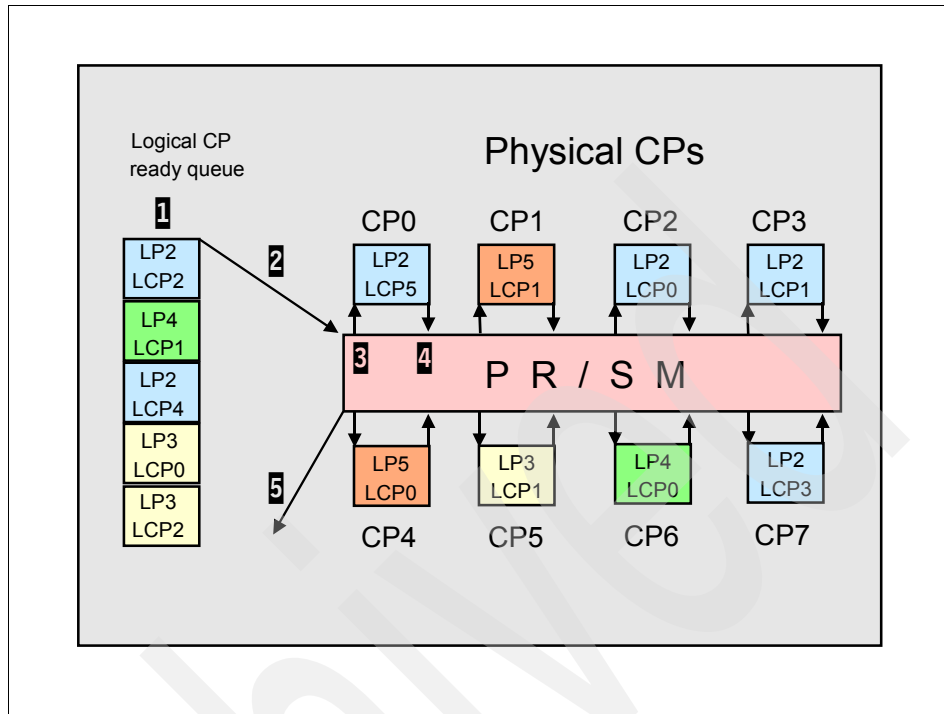


Figure 1-2 Logical CP dispatching

To dispatch a logical CP, these steps are followed:

1. A logical CP is selected from the logical CP ready queue.
2. The LPAR scheduler LIC dispatches the selected logical CP on a physical processor.
3. The logical CP executes on physical CP until it is *intercepted*. When a physical CP is taken away from a logical CP, it is said to be intercepted. An intercept can occur because:
 - The logical CP's time slice expires.
 - A logical CP running over its target weight is preempted by an I/O operation on a logical CP under its target weight.
 - The operating system running in the logical CP voluntarily relinquishes its time slice (when it has no work to perform, for instance).
4. When the time slice expires or the logical CP is intercepted, the state of the logical CP is saved and control returns to PR/SM (which starts executing on the physical CP again).

5. LPAR scheduler LIC determines why the logical CP ended. If it has additional work to perform, it is requeued on the logical CP ready queue.

This process occurs on each physical CP.

LPAR weights

LPAR weights control the distribution of shared CPs between LPARs. LPARs with dedicated CPs do not use LPAR weights. Weights determine the guaranteed minimum amount of physical CP resources an LPAR should receive. The guaranteed minimum can become a maximum when either:

- ▶ All LPARs use all of their guaranteed amount (for instance, all LPARs are completely CPU-bound).
- ▶ The LPAR is capped using traditional LPAR capping.

An LPAR might use less than the guarantee if it does not have much work to do. Similarly, it can use more than its weight if the other LPARs use less than their guaranteed amount. An LPAR's *weight* and *number of logical CPs* determine the priority of its logical CPs in the logical CP ready queue.

Archived

Linux 2.6 kernel

Linux 2.6 kernel provides support for many new architectures, file Systems, And Devices. Enterprise Ready Linux Distributions Featuring The Linux 2.6 kernel show improved performance, scalability, speed, and stability. This chapter discusses some of these new functions. Topics in this chapter include:

- ▶ Choosing a 64-bit or 31-bit Linux distribution
- ▶ Performance and scalability enhancements
- ▶ Introduction to sysfs
- ▶ S/390 tools and utilities

2.1 Choosing a 64-bit or 31-bit Linux distribution

The Linux 2.6 kernel has undergone a substantial testing in 64-bit mode, and is intended for large-scale, highly available systems. The IBM strategy is focused on middleware applications running on 64-bit platforms. Whenever possible, a 64-bit enterprise Linux distribution is recommended. The 64-bit kernel offers greater memory addressability, and provides greater flexibility for running Linux on zSeries. With the 64-bit kernel, you can run many small Linux images or fewer but larger Linux images in a given z/VM partition. Applications that can benefit most from 64-bit technology include:

- ▶ Databases
- ▶ Applications requiring access to large amounts of data
- ▶ Java applications

2.1.1 Linux 31-bit compatibility mode

Although most middleware has been or will be ported to 64-bit, it is important to note that not all will benefit from the larger address space. In these cases, the middleware continues to operate in 31-bit addressing mode on a 64-bit Linux distribution using *compatibility mode*. See Figure 2-1 on page 11.

Note: For information about the availability of IBM middleware on Linux for zSeries, see:

<http://www.ibm.com/linux/matrix/linuxmatrixhwz.html>

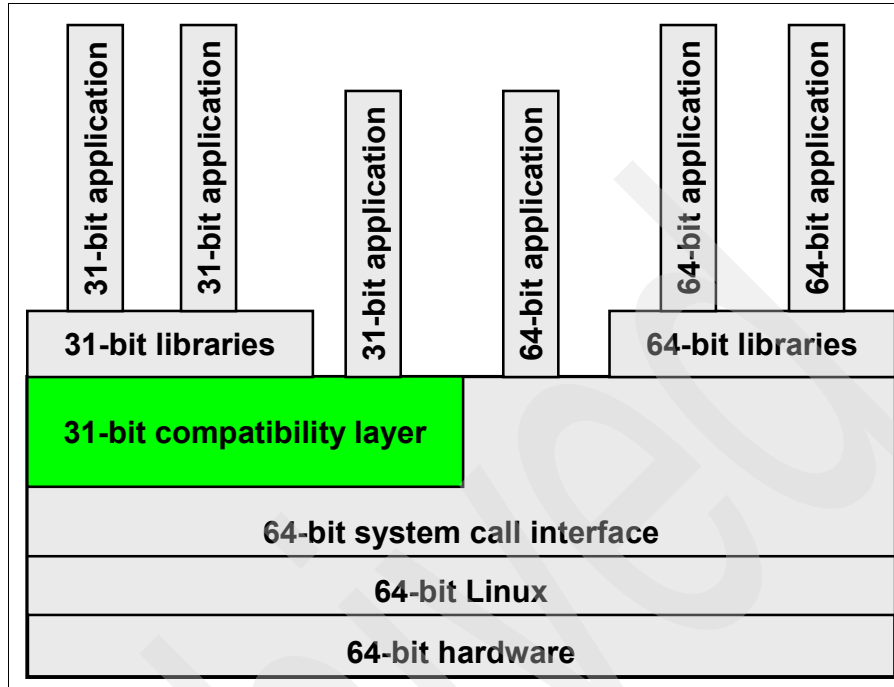


Table 2-1 Compatibility mode system call emulation

2.2 Performance and scalability enhancements

Much of the Linux 2.6 kernel development is intended to increase performance and improve scalability. Some of the major changes made to the kernel include:

- ▶ Reduction in the number of global locks

To improve system responsiveness, many global locks have been replaced with fine-grained locks. For instance, both the block I/O layer lock and all global locks in the virtual memory layer have been replaced.
- ▶ Increase in the number of unique users and groups

The user and group ID variables have increased from a 16-bit value to a 32-bit value, increasing the maximum number from 65,000 to over 4 billion. Similarly, the number unique processes has increased from 32,000 to over 1 billion.
- ▶ Increase in the number of simultaneously attached devices

Prior to the Linux 2.6 kernel, a Linux system was limited to 255 different types of devices (the device *major* number); each device type is limited to 255

specific instances (the device *minor* number). With 2.6, up to 4095 device types can be connected. Each device type can now support more than 1 million connected instances.

We look at other performance and scalability improvements in 2.6:

- ▶ The new Linux process scheduler
- ▶ System swap setting
- ▶ Kernel preemption
- ▶ Fast user-space mutual exclusion
- ▶ I/O performance and scalability enhancements
- ▶ I/O schedulers
- ▶ Extended attributes and access control lists
- ▶ Memory management enhancements
- ▶ Native POSIX Threading Library
- ▶ NFS version 4

2.2.1 The new Linux process scheduler

The Linux 2.6 kernel implements a major rewrite to the process scheduler. Often referred to as the *O(1) scheduler*, the new scheduler requires a constant number of operations to select the next process regardless of the number waiting in the queue. Its efficiency leads to excellent scalability for multiprocessor systems, even under large workloads. Other benefits available in the new process scheduler include:

- ▶ Soft CPU affinity

With soft CPU affinity, processes tend to run on a single processor as long as possible, and resist migration to another processor unless necessary. This avoids situations where tasks bounce needlessly between CPUs.

- ▶ Hard CPU affinity

By default, processes are runnable on all processors and pass CPU soft affinity on to their children. With hard CPU affinity, the scheduler uses two arrays (each position in the array corresponding to a list of processes with the same priority value). Processes in the *active* array run for one time slice. This is calculated based on their priority and prior blocking rate. They then move to the *expired* array. When all processes in the active array have expired, the arrays are switched and the scheduling begins again. Hard affinity is designed to provide higher scalability for SMP systems.

- ▶ Interactive workload performance

Because a processes time slice is calculated partly by its prior blocking rate, the scheduler tends to favor interactive tasks. Interactive workloads should see improved performance even under heavy load.

Note: For more details on the O(1) process scheduler, see "Kernel Korner - What's New in the 2.6 Scheduler" by Rick Lindsley in Linux Journal, March 2004 available at:

<http://www.linuxjournal.com/article/7178>

2.2.2 System swap setting

A new interface to the proc virtual filesystem allows greater control over system paging. When needed, the Linux virtual memory manager attempts to obtain memory pages from the page cache or by swapping pages from user processes. In the 2.6 kernel, the new `/proc/sys/vm/swappiness` parameter influences how memory pages are obtained:

- ▶ Values closer to 0 indicate the system is less likely to swap.
- ▶ Values closer to 100 indicate the system is more likely to swap.

The default setting for Linux on zSeries is 60, and has been chosen as appropriate for a typical system.

Note: Do not change this value unless testing has indicated a different value is appropriate for your workload.

2.2.3 Kernel preemption

With Linux 2.4 kernel, tasks running in kernel mode cannot be preempted. With Linux 2.6 kernel, some parts of the kernel are preemptable. This can decrease kernel latency and provide greater responsiveness, particularly to interactive tasks.

Note: Kernel preemption is disabled on Linux for zSeries distributions because this option might expose timing issues and have an adverse effect on performance for traditional workloads.

2.2.4 Fast user-space mutual exclusion

Responsiveness in the Linux 2.6 kernel has been improved by the introduction of fast mutexes (referred to as *futexes*). Designed as a fast synchronization

primitive, futexes operate in both user-space, and kernel-space. Futex operations begin in user-space. If no contention exists for the futex, the operation completes in user-space. Expensive system calls to kernel-space are required only if contention exists for the futex.

2.2.5 I/O performance and scalability enhancements

Several improvements have been made to the I/O subsystem. In the Linux 2.6 kernel, there is no global I/O request lock. Instead, each I/O device has its own I/O request lock. Both the asynchronous I/O and direct I/O layers have been rewritten:

- ▶ The asynchronous I/O system allows applications to perform reads and writes without waiting for the operation to complete. Previously, processes issuing writes in rapid succession (database write transactions for instance) could experience performance penalties.
- ▶ Direct I/O allows data transfers to occur directly between user buffers and the I/O device (without using kernel buffers), and replaces raw devices in the Linux 2.4 kernel. Applications access block device node (such as /dev/sdb1) instead of /dev/raw01.

Both features are exploited at the application level. With asynchronous I/O, database servers can realize an overall throughput improvement of up to 50%. For more details on I/O options for database servers, see:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_rec_database_GenRec.html

Both Informix database server for Linux 2.6 kernel and IBM DB2 UDB ESE v8.2 can use these options.

Note: For details on the asynchronous and direct I/O subsystems, see *Asynchronous I/O Support in Linux 2.5* by Suparna Bhattacharya, et al reprinted from the Proceedings of the Linux Symposium, July 23-26, 2003 at:

<http://archive.linuxsymposium.org/o1s2003/Proceedings/All-Reprints/Reprint-Pulavarty-0LS2003.pdf>

2.2.6 I/O schedulers

The biggest changes to the I/O subsystem is the additional of several new I/O schedulers. The I/O scheduler is responsible for reducing device seek time by:

- ▶ Sorting I/O requests to the device by block number
As I/O requests are handled by kernel, the I/O scheduler sorts requests by block number. This allows the device read/write head to perform a sequential scan of the disk surface.
- ▶ Merging I/O requests to a device
New I/O requests are merged into the sorted list to reduce the number of required read/write head scans.

Note: For details on operation of the I/O scheduler, see "Kernel Korner - I/O Schedulers" by Rob Love in Linux Journal, February 2004 at:

<http://www.linuxjournal.com/article/6931>

Four new I/O schedulers are available in Linux 2.6 kernel:

- ▶ Completely fair queuing (cfq)
This is the default scheduler on Linux for zSeries. The cfq scheduler attempts to service all users (processes) of a particular device equally over time. That is, in a given time interval each process using a device should have a roughly equal number of I/O requests serviced. Systems with RAID 5 storage work well with this scheduler.
- ▶ Anticipatory (as)
The as scheduler attempts to minimize situations where many write requests are interrupted by a few read requests. After an I/O operation, the scheduler pauses in anticipation of an I/O request for an adjacent block. This is a behavior typically suitable for desktop workstations. The as scheduler is *not* intended to use for by storage servers.
- ▶ Deadline
With the deadline scheduler, I/O requests are assigned a deadline by which they must be processed. If a request reaches its deadline, the scheduler immediately processes the request (even if it is out of sorted order).
- ▶ Noop
The noop scheduler processes requests in the merged order in which they are received. No further optimization is performed.

Selecting an I/O scheduler

The specific I/O scheduler to use is determined at system IPL by the elevator parameter in the `/etc/zipl.conf` file (Example 2-1 on page 16):

Example 2-1 Elevator parameter of the /etc/zipl.conf file

```
# cat /etc/zipl.conf
[defaultboot]
    default = ipl

[ipl]
    target = /boot/zipl
    image = /boot/image
    ramdisk = /boot/initrd
    parameters = "root=/dev/dasda1 selinux=0 TERM=dumb elevator=cfq"
```

Valid elevator parameter values are:

fcq	Selects the completely fair scheduler
as	Selects the anticipatory scheduler
deadline	Selects the deadline scheduler
noop	Selects the noop scheduler

Note: Remember to run the `zipl` command after changing the `/etc/zipl.conf` file. Changes take effect after the next system IPL.

In kernel versions greater than 2.6.9, the I/O scheduler can be selected at runtime using the `/sys/block/dev/queue/scheduler` interface (where `dev` is the DASD device number).

2.2.7 Extended attributes and access control lists

The 2.6 kernel supports extended attributes (EAs) and access control lists (ACLs):

- ▶ EAs are name/value pairs associated with file objects that describe characteristics of the object (such as mime type and character set encoding).
- ▶ ACLs are EAs that extend the traditional set of Linux permissions (read, write, and execute for owner, group, and other). Using ACLs, permissions to specific file objects can be granted regardless of owner and group membership (greatly simplifying Linux user and group management).

ACL support offers greater interoperability between Linux and other POSIX-compliant operating systems. ACLs are supported on the `ext2`, `ext3`, `jfs`, `xfs`, and `reiserfs` filesystems. The `getfacl` and `setfacl` commands are used to view and modify ACLs.

2.2.8 Memory management enhancements

Major changes have been made to the memory management system in the 2.6 kernel with the implementation of reverse page mapping (rmap). Reverse page mapping allows the kernel to map virtual pages efficiently to physical addresses. Previously, mapping physical pages back to their virtual addresses was a costly operation (in the worst case, it required examining every page table in the system). With rmap, additional kernel structures have been added for the backwards translation. This feature allows the virtual memory manager to make more intelligent decisions on what to swap out to disk (as well as what to bring back from disk).

2.2.9 Native POSIX Threading Library

Native POSIX Thread Library (NPTL) is the default thread library for 2.6, replacing the LinuxThreads library. NPTL offers significant performance improvements over LinuxThreads. In addition, LinuxThreads is not compliant with the POSIX standard for threads.

Current Linux distributions offer the ability to enable your earlier and existing threading functionality. That means you can use LinuxThreads without code changes. However, it is highly recommended that you make any necessary code changes to allow your enterprise applications to take advantage of NPTL. Another advantage to using NPTL is that in process listings, threads are now shown under the processes to which they belong. Previously, each thread was listed as a separate process. NPTL also supports thread local storage and futex-based synchronization. Each of these can result in dramatic performance gains.

If you are running with NPTL but still want to run it with LinuxThreads, you have to tell your system to use LinuxThreads. To do this, the `LD_ASSUME_KERNEL` variable has to be set to 2.4.19 with the command:

```
export LD_ASSUME_KERNEL=2.4.19
```

Process information for threaded applications displayed by the `ps` command varies according to the thread library used:

- ▶ With LinuxThreads, a thread appears as a child processes of the main thread. This can lead to confusion: Is the child entity a thread or a process?
- ▶ With NPTL, as expected, no child thread information is displayed. The `ps` command displays only child processes.

To illustrate the difference, The `Teste.class` Java application creates numerous child threads. Figure 2-1 on page 18 shows the process list displayed when using the LinuxThreads library.

```

# ps -a -x -f
  PID TTY          STAT       TIME COMMAND
    1 ?            S           0:05 init [5]
.
.
.
 1182 ?            Ss          0:00 /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid
 1523 ?            Ss          0:00 \_ sshd: root@pts/0
 1526 pts/0        Ss          0:00 | \_ -bash
 2794 pts/0        R+          0:00 | | \_ java Teste class
 2795 pts/0        S+          0:00 | | | \_ java Teste class
 2796 pts/0        S+          0:00 | | | | \_ java Teste class
 2797 pts/0        S+          0:00 | | | | | \_ java Teste class
 2798 pts/0        S+          0:00 | | | | | | \_ java Teste class
 2799 pts/0        S+          0:00 | | | | | | | \_ java Teste class
 2800 pts/0        S+          0:00 | | | | | | | | \_ java Teste class
.
.
.

```

Figure 2-1 Running Test.class using LinuxThreads

In the figure, PIDs 2794 through 2800 are threads, but appears as processes. Figure 2-2 shows a process list taken from a Linux 2.6 kernel system using NPTL.

```

# ps -a -x -f
  PID TTY          STAT       TIME COMMAND
    1 ?            S           0:05 init [5]
.
.
.
 1182 ?            Ss          0:00 /usr/sbin/sshd -o PidFile=/var/run/sshd.init.pid
 1523 ?            Ss          0:00 \_ sshd: root@pts/0
 1526 pts/0        Ss          0:00 | \_ -bash
 1594 pts/0        S+          0:00 | \_ java Teste class
.
.
.

```

Figure 2-2 Running Test.class using NPTL

In this case, only a single process (PID 1594) is reported. Information on threads running in the process is available in the /proc/1594 directory.

To determine which threading library in use, use the **getconf** command, as in Example 2-2 on page 19.

Example 2-2 The getconf command on SLES9

```
# getconf GNU_LIBPTHREAD_VERSION
NPTL 0.61
# export LD_ASSUME_KERNEL=2.4.19
# getconf GNU_LIBPTHREAD_VERSION
linuxthreads-0.10
```

The previous example executes on a SLES9 system. On a RHEL4 system, the **getconf** command returns (Example 2-3):

Example 2-3 The getconf command on RHEL4

```
# getconf GNU_LIBPTHREAD_VERSION
NPTL 2.3.4
# export LD_ASSUME_KERNEL=2.4.19
# getconf GNU_LIBPTHREAD_VERSION
linuxthreads-0.10
```

If the threading library is NPTL, **getconf** returns:

- ▶ **NPTL 0.61** for SLES9
- ▶ **NPTL 2.3.3** for RHEL4

In both cases, the response is `linuxthreads-0.1` if LinuxThreads is used.

2.2.10 NFS version 4

The Linux 2.6 kernel includes revision 4 of the Networking File System (NFSv4). NFS allows sharing filesystems across a network (a machine can locally mount filesystems on remote machines). Improvements include:

- ▶ Secure network transactions

With NFSv4, remote procedure calls (RPC) transactions can be secured using the General Security Service (GSS) API. The GSS framework provides mechanisms for authentication, privacy, and integrity checking between client and server.
- ▶ Reduced number of remote procedure calls

Prior to version 4, NFS transactions translated to numerous low-level RPC transactions. NFSv4 implements compound procedure calls. This combines several low-level RPC transactions into a single high-level RPC transaction. With compound procedure calls, network traffic (and the associated latency) is reduced.

- ▶ New file locking mechanism

NFSv4 implements byte-range file locking, replacing the lockd protocol formerly provided by the Network Lock Manager. With this model, the lockd and mountd daemons are no longer used, resulting in significant CPU savings for NFS servers.

In this newer implementation, clients request a lock from the server. After the lease is granted, the server maintains a lease on the lock. Before expiration, the client must renew the lease in order to hold the lock. If the lease expires, the server releases the lock.

2.3 Introduction to sysfs

The 2.6 kernel introduces sysfs, a new virtual filesystem interface to devices attached to the system. Benefits offered by sysfs include:

- ▶ Simplification of the /proc virtual filesystem

Originally designed as an interface to processes running on the system, over time /proc incorporated interfaces to device drivers. With sysfs, device driver interfaces have been moved to the /sys directory.

Note: Some device driver interfaces remain in the /proc directory. However these should be treated as read-only.

- ▶ Consistent device configuration

With the /proc device interface, individual drivers were free to choose their specific file formats. With sysfs, device driver interfaces are consistent:

- The directory path for a device under /sys indicates the type of device, and the subchannel for the device.
- Each file in a device subdirectory accesses a single attribute of the device.

- ▶ Simplification of the /dev directory

Prior to 2.6, the /dev directory contained the statically created device nodes for all possible devices. In 2.6, device nodes are dynamically created in /dev as devices are attached to the system.

2.3.1 Organization of the /sys directory

Figure 2-3 on page 21 illustrates a portion of the /sys directory. As shown in the figure, the /sys directory structure provides information about the device type, its device node name, and the subchannel used to the device. Regular files in a

device subdirectory access attributes of the device. Symbolic links allow the device to be accessed from various points in the tree.

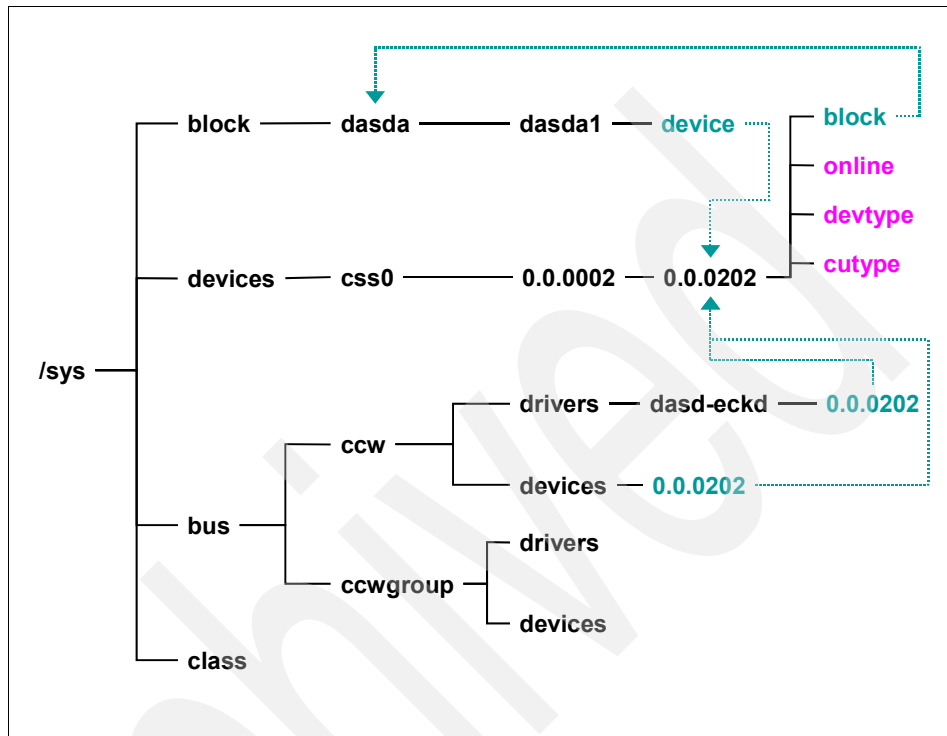


Figure 2-3 A portion of the /sys directory

Directories found in the tree include:

- The /sys/devices directory

The /sys/devices/css0 directory contains all subchannels detected by the Linux 2.6 kernel. Subchannel directory names take the form 0.0.nnnn where nnnn is the hexadecimal subchannel number. Subchannel directories in turn contain device directories of the form 0.0.xxxx where xxxx is unit address of the device. Status and configuration options for a device exist as regular files under the device directory.

Active network devices are found in the /sys/devices/type/0.0.addr directory where type is the type of network device and addr is its unit address. Figure 2-4 on page 22 shows the device directory for the QDIO device at address 0x0600.

```
# ls /sys/devices/qeth/0.0.0600/
.                canonical_macaddr checksumming      if_name          priority_queueing state
..               card_type         chpid            ipa_takeover     recover          ungroup
add_hhlen        cdev0             detach_state     online           route4           vipa
broadcast_mode   cdev1             fake_broadcast   portname         route6
buffer_count     cdev2             fake_ll          portno           rxip
```

Figure 2-4 The `/sys/devices/qeth/0.0.0600/` directory

To query the online status of the device, run this command:

```
# cat /sys/devices/qeth/0.0.0600/online
1
```

The device is online as indicated by the 1 (0 indicates the device is offline).

- ▶ The `/sys/bus` directory

This directory contains the `ccw` and `ccwgroup` subdirectories. CCW devices are accessed using channel command words. Devices in the `ccw` subdirectory use only one subchannel in the channel subsystem. Devices in the `ccwgroup` subdirectory use more than one subchannel per device. For example, DASD devices use a single subchannel, while a QDIO device uses three. Symbolic links under the `ccw` and `ccwgroup` subdirectories point back to the `/sys/devices` entry for the specific device.

- ▶ The `/sys/class` directory

This directory groups similar devices together.

- ▶ The `/sys/block` directory

This directory contains a subdirectory for each block device on the system.

2.3.2 New `sysfs` command for Linux on zSeries

To help manage `sysfs`, several new commands have been provided:

- ▶ The `lsdasd` command displays information for DASD devices.
- ▶ The `lscss` command displays subchannel information. Figure 2-5 on page 23 shows output from the `lscss` command.

# lscss									
Device	Subchan.	DevType	CU	Type	Use	PIM	PAM	POM	CHPIDs
0.0.0103	0.0.0000	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.0201	0.0.0001	3390/0A	3990/EC	yes		F0	F0	FF	74754849 00000000
0.0.0202	0.0.0002	3390/0A	3990/EC	yes		F0	F0	FF	74754849 00000000
0.0.0009	0.0.001F	0000/00	3215/00	yes		80	80	FF	00000000 00000000
0.0.0600	0.0.0020	1732/01	1731/01	yes		80	80	FF	06000000 00000000
0.0.0601	0.0.0021	1732/01	1731/01	yes		80	80	FF	06000000 00000000
0.0.0602	0.0.0022	1732/01	1731/01	yes		80	80	FF	06000000 00000000
0.0.000C	0.0.0023	0000/00	2540/00			80	80	FF	00000000 00000000
0.0.000D	0.0.0024	0000/00	2540/00			80	80	FF	00000000 00000000
0.0.000E	0.0.0025	0000/00	1403/00			80	80	FF	00000000 00000000
0.0.0190	0.0.0026	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.019D	0.0.0027	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.019E	0.0.0028	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.0592	0.0.0029	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.0191	0.0.002A	3390/0A	3990/EC			F0	F0	FF	74754849 00000000
0.0.B020	0.0.002B	1732/03	1731/03			80	80	FF	90000000 00000000
0.0.B030	0.0.002C	1732/03	1731/03			80	80	FF	90000000 00000000
0.0.B033	0.0.002D	1732/03	1731/03			80	80	FF	90000000 00000000
0.0.B130	0.0.002E	1732/03	1731/03			80	80	FF	91000000 00000000

Figure 2-5 Output from the lscss command

- ▶ The **lstape** command shows tape device information.
- ▶ The **chccwdev** command enables or disables devices:

```
chccwdev -e dev    bring device online
chccwdev -d dev    bring device offline
```

Note: The **chccwdev** command does not work on ccwgroup devices (such as OSA or CTC devices).

Figure 2-6 on page 24 illustrates using the **chccwdev** command.

```

# lsdasd -a 0.0.0300
0.0.0300(none) : offline
# chccwdev -e 0.0.0300
Setting device 0.0.0300 online
Done
# lsdasd
0.0.0201(ECKD) at ( 94: 0) is dasda      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0202(ECKD) at ( 94: 4) is dasdb      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0300(ECKD) at ( 94: 8) is dasdc      : active at blocksize 4096, 90000 blocks, 351 MB
# chccwdev -d 0.0.0300
Setting device 0.0.0300 offline
Done
# lsdasd
0.0.0201(ECKD) at ( 94: 0) is dasda      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0202(ECKD) at ( 94: 4) is dasdb      : active at blocksize 4096, 600840 blocks, 2347 MB

```

Figure 2-6 Using the `chccwdev` command

These commands perform actions which can be accomplished using `sysfs` directly, but are provided as merely for convenience.

2.3.3 Device configuration with `sysfs`

Linux 2.6 kernel uses `sysfs` to enable, disable and configure devices. SLES9 and RHEL4 differ in the methods used to configure devices at system IPL. In this section, we look at automatic device configuration in both distributions.

Note: When configuring a new device, ensure its device driver is first loaded. The correct module for each device, how to load it and its parameters can be found in *Linux on zSeries Device Drivers, Features, and Commands March 23, 2005, SC33-8281*.

2.3.4 SLES9 hardware configuration

SLES9 uses a new scheme to configure devices. At boot time, the system configures hardware devices based on configuration files found in the `/etc/sysconfig/hardware` directory.

Note: The `sysconfig` package provides hardware configuration tools and scripts. Sample hardware configuration files can be found in the `/etc/sysconfig/hardware/skel` directory.

Configuring FCP devices

To configure an FCP device, follow these steps:

1. Copy the `/etc/sysconfig/hardware/skel/hwcfg-zfc` file to `/etc/sysconfig/hardware/hwcfg-zfc-bus-ccw-0.0.nnnn` (where `nnnn` is the FCP device device number). Change the `FCP_LUN` and `WWPN` parameters for your specific environment (see Figure 2-7).
2. Bring the device online using the `hwup` command:

```
hwup zfc-bus-ccw-0.0.nnnn
```

If it is successful, the device will be automatically brought online at reboot.

```
# cat /etc/sysconfig/hardware/hwcfg-zfc-bus-ccw-0.0.b025
#!/bin/sh
#
# hwcfg-zfc
#
# Default configuration for a zfc device
# $Id: hwcfg-zfc,v 1.3 2004/02/24 10:47:10 hare Exp $
#

STARTMODE="auto"
MODULE="zfc"
MODULE_OPTIONS=""
MODULE_UNLOAD="yes"

# Scripts to be called for the various events.
# If called manually the event is set to 'up'.
SCRIPTUP="hwup-ccw"
SCRIPTUP_ccw="hwup-ccw"
SCRIPTUP_scsi_host="hwup-zfc"
SCRIPTDOWN="hwdown-scsi"
SCRIPTDOWN_scsi="hwdown-zfc"
# CCW_CHAN_IDS sets the channel IDs for this device
# If not set or empty the channel ID from the file name
# is used
# CCW_CHAN_IDS=""
# CCW_CHAN_NUM set the number of channels for this device
# If not set it or empty default to '1'
# CCW_CHAN_NUM=1
# fcp LUNs configured for this adapter.
# Note that IBM ESS (aka 'Shark') ignores
# all but the first 4 characters of the
# fcp LUN.
ZFCP_LUNS="0x5005076300cd9589:0x520c000000000000"
```

Figure 2-7 Hardware configuration file for FCP device at virtual address b025

Note: For more details on using FCP devices, see Chapter 5, “FCP-attached SCSI disks” on page 155.

Configuring DASD devices

To configure a DASD device, follow these steps:

1. Copy the `/etc/sysconfig/hardware/skel/hwcfg-eckd` file to `/etc/sysconfig/hardware/hwcfg-dasd-bus-ccw-0.0.nnnn` (where `nnnn` is the DASD device number).
2. Bring the device online using the `hwup` command:

```
hwup dasd-bus-ccw-0.0.nnnn
```

If successful, the device will be automatically brought online at reboot. Figure 2-8 on page 26 shows a DASD configuration file.

```
# cat /etc/sysconfig/hardware/hwcfg-dasd-bus-ccw-0.0.0202
#!/bin/sh
#
# hwcfg-dasd-bus-ccw-0.0.0202
#
# Configuration for a DASD device (ECKD mode)
#
STARTMODE="auto"
MODULE="dasd_eckd_mod"
MODULE_OPTIONS=""
MODULE_UNLOAD="yes"
# Scripts to be called for the various events.
# If called manually the event is set to 'up'.
SCRIPTUP="hwup-ccw"
SCRIPTUP_ccw="hwup-ccw"
SCRIPTDOWN="hwdown-ccw"
SCRIPTDOWN_ccw="hwdown-ccw"
# DASD_USE_DIAG selects whether DIAG access mode
# should be activated for this device
# DASD_USE_DIAG="0"
```

Figure 2-8 Hardware configuration file for DASD device at virtual address 202

Configuring QDIO network devices

To configure a QDIO network device, follow these steps:

1. Copy the `/etc/sysconfig/hardware/skel/hwcfg-qeth` file to `/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.nnnn` (where `nnnn` is the base QDIO device number).

2. Edit the new hardware configuration file, supplying your specific hardware parameters (see Figure 2-9).
3. Bring the device online using the **hwup** command:

```
hwup qeth-bus-ccw-0.0.nnnn
```

```
# cat /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.0700
#!/bin/sh
#
# hwcfg-qeth-bus-ccw-0.0.0700
#
# Hardware configuration for a qeth device at 0.0.0700
# Automatically generated by netsetup
#
STARTMODE="auto"
MODULE="qeth"
MODULE_OPTIONS=""
MODULE_UNLOAD="yes"

# Scripts to be called for the various events.
SCRIPTUP="hwup-ccw"
SCRIPTUP_ccw="hwup-ccw"
SCRIPTUP_ccwgroup="hwup-qeth"
SCRIPTDOWN="hwdown-ccw"

# CCW_CHAN_IDS sets the channel IDs for this device
# The first ID will be used as the group ID
CCW_CHAN_IDS="0.0.0700 0.0.0701 0.0.0702"

# CCW_CHAN_NUM set the number of channels for this device
# Always 3 for an qeth device
CCW_CHAN_NUM=3

# CCW_CHAN_MODE sets the port name for an OSA-Express device
CCW_CHAN_MODE="suselin7"
```

Figure 2-9 Hardware configuration file for QDIO interface at virtual address 700

Note: Similar skeletal hardware configuration files exist for IUCV, CTC, and LSC devices. The file name reflects the type of device to configure.

Be sure to create a network configuration file in the `/etc/sysconfig/network` directory. Network configuration file names take the form `ifcfg-xxxx-bus-ccw-0.0.nnnn` where `xxxx` denotes the interface name, and `nnnn` denotes the device number.

2.3.5 RHEL4 hardware configuration

RHEL4 device configuration differs according to the device.

Configuring FCP devices

To configure an FCP device, follow these steps:

1. Modify the `/etc/modules.conf` file:

```
alias eth0 qeth
options dasd_mod dasd=201,202
alias scsi_hostadapter zfcp
```

2. Map the FCP devices in the `/etc/zfcp.conf` file (Example 2-4):

Example 2-4 Mapping the devices

```
0.0.b020 0x01 0x5005076300cd9589 0x01 0x5210000000000000
0.0.b022 0x02 0x5005076300cd9589 0x01 0x5211000000000000
0.0.b030 0x03 0x5005076300c19589 0x01 0x5210000000000000
0.0.b033 0x04 0x5005076300c19589 0x01 0x5211000000000000
```

3. Generate a new initial ramdisk containing the `zfcp` module and the FCP mapping. The steps are illustrated in Figure 2-10 on page 29.

```

# cd /boot
# mv initrd-2.6.9-6.37.EL.img initrd-2.6.9-37.EL.img.orig
# mkinitrd -v --with=scsi_mod --with=zfcpl --with=sd_mod initrd-2.6.9-6.37.EL
2.6.9-6.37.EL
Creating initramfs
Looking for deps of module ide-disk
.
.
.
Using modules: ./kernel/drivers/md/dm-mod.ko ./kernel/drivers/s390/block/dasd_mod.ko ...
/sbin/nash -> /tmp/initrd.uU2014/bin/nash
/sbin/insmod.static -> /tmp/initrd.uU2014/bin/insmod
/sbin/udev.static -> /tmp/initrd.uU2014/sbin/udev
/etc/udev/udev.conf -> /tmp/initrd.uU2014/etc/udev/udev.conf
copy from /lib/modules/2.6.9-6.37.EL/./kernel/drivers/md/dm-mod.ko(elf64-s390) to ...
.
.
.
Loading module dm-mod
.
.
.
Loading module sd_mod

```

Figure 2-10 Generating a new initial ramdisk containing the zfcpl device driver

4. Execute the `zip1` command to update the bootloader, as shown in Figure 2-11 on page 30:

```

# zipl -V
Using config file '/etc/zipl.conf'
Target device information
Device.....: 5e:00
Partition.....: 5e:01
Device name.....: dasda
DASD device number.....: 0201
Type.....: disk partition
Disk layout.....: ECKD/compatible disk layout
Geometry - heads.....: 15
Geometry - sectors.....: 12
Geometry - cylinders.....: 3338
Geometry - start.....: 24
File system block size.....: 4096
Physical block size.....: 4096
Device size in physical blocks...: 25596
Building bootmap '/boot//bootmap'
Building menu 'rh-automatic-menu'
Adding #1: IPL section 'linux' (default)
  kernel image.....: /boot/vmlinuz-2.6.9-6.37.EL at 0x10000
  kernel parmline...: 'root=LABEL=/' at 0x1000
  initial ramdisk...: /boot/initrd-2.6.9-6.37.EL.img at 0x800000
Preparing boot device: dasda (0201).
Preparing boot menu
  Interactive prompt.....: enabled
  Menu timeout.....: 15 seconds
  Default configuration...: 'linux'
Syncing disks...
Done

```

Figure 2-11 Updating the bootloader with zipl

Note: For more details on using FCP devices, see Chapter 5, “FCP-attached SCSI disks” on page 155. The RHEL4 installation guide also provides information about FCP device configuration:

<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/s390-multi-in-stall-guide/s1-s390info-zfcp.html>

Configuring DASD devices

To configure a DASD device, follow these steps, follow these steps:

1. Update the /etc/modprobe.conf file with the new DASD addresses.

```

alias eth0 qeth
options dasd_mod dasd=201,202,300

```

2. Generate a new initial ramdisk (Example 2-5):

Example 2-5 Generating a new ramdisk

```
cd /boot
# mv initrd-2.6.9-6.37.EL.img initrd-2.6.9-37.EL.img.orig
# mkinitrd -v --with=scsi_mod --with=zfcplib --with=sd_mod \
initrd-2.6.9-6.37.EL.img 2.6.9-6.37.EL
Creating initramfs
Looking for deps of module ide-disk
.
.
.
```

Note: If your system uses FCP devices, be sure to include the zfcplib device driver in the initial ramdisk. Also ensure the devices are mapped in the /etc/zfcplib.conf file.

3. Execute the **zipl** command to update the bootloader:

```
# zipl -V
```

For more information, see the RHEL4 installation guide at:

<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/s390-multi-inst-all-guide/s1-s390info-adddasd.html>

Configuring network devices

To configure a network device, follow these steps:

1. Ensure the /etc/modprobe.conf file has an alias for the interface name to the device driver.

```
alias eth0 qeth
alias eth1 qeth
options dasd_mod dasd=201,202,300
```

2. Create the appropriate network configuration file in the /etc/sysconfig/network-scripts directory. Configurations file names have the form ifcfg-*ndev* where *ndev* is the name of the network interface. A network configuration file for the eth1 adapter (a QDIO device) is shown in Figure 2-12 on page 32.

```
# IBM QETH
DEVICE=eth1
BOOTPROTO=static
BROADCAST=9.12.5.255
IPADDR=9.12.4.83
NETMASK=255.255.254.0
NETTYPE=qeth
NETWORK=9.12.4.0
ONBOOT=yes
PORTNAME=lnxrh2
SUBCHANNELS=0.0.0700,0.0.0701,0.0.0702
TYPE=Ethernet
```

Figure 2-12 The `/etc/sysconfig/network-scripts/ifcfg-eth1` configuration file

For more information about adding network interfaces, see:

<http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/s390-multi-inst-all-guide/s1-s390info-addnetdevice.html>

2.3.6 Device node naming with udev

In the Linux 2.6 kernel, dynamic device node naming is accomplished with `udev` - a user space tool that creates device nodes in `/dev`. The `udevinfo` command is used to show information about the device nodes created by `udev`:

```
# udevinfo -q symlink -n /dev/dasda
disk/by-path/ccw-0.0.0201
disk/by-id/OX0201
```

A device node can be referenced more than one name:

```
# udevinfo -q symlink -n /dev/dasda
disk/by-path/ccw-0.0.0201
disk/by-id/OX0201
```

The symbolic links to device nodes differ based on distribution. For instance, with DASD device at address 201:

- ▶ In SLES9, `udev` creates symbolic links:
 - `/dev/disk/by-path/ccw-0.0.0201`
 - `/dev/disk/by-id/OX0201`
- ▶ In RHEL4, `udev` creates the `/dev/dasd/0.0.0201/disc` symbolic link.

Note: With RHEL4, udev dynamically manages all device node naming. SLES9 also uses udev, however 2.4 static device node names are defined in `/dev` for backwards compatibility.

2.3.7 Hotplug detection and device node naming

When devices are attached to the system, events are generated by `/sbin/hotplug`. These events are intercepted by `udev` (running in user space). From the hotplug event, `udev` obtains device information from `sysfs`, and dynamically adds the appropriate device node to the `/dev` directory.

In order to ensure consistent and deterministic device node naming, `udev` uses device naming conventions specified in a rules file. Rules in this file specify what device node name is assigned by `udev` (based on attributes of the device obtained from `sysfs`). Each distribution for Linux on zSeries uses its specific `udev` rules file:

- ▶ SLES9 uses the `/etc/udev/udev.rules` file.
- ▶ RHEL4 uses the `/etc/udev/rules.d/50-udev.rules` file.

Rules rely on regular expression matching to map device attributes to device node names. Attributes can refer to the bus a device is attached to, a unique vendor ID for the device, the kernel name for the device, or a device attribute from `sysfs`. Based on the regular expression, `udev` creates a symbolic link to the device node in the `/dev` directory. Figure 2-13 illustrates sample `udev` rules for DASD and FCP devices.

```
KERNEL="dasd*[a-z]", SYMLINK="dasd/%b/disk", NAME="%k"
KERNEL="dasd*[0-9]", SYMLINK="dasd/%b/part%n", NAME="%k"

KERNEL="sd*[a-z]", SYMLINK="scsi/%s{fcp_lun}-%s{wwpn}/disk", NAME="%k"
KERNEL="sd*[0-9]", SYMLINK="scsi/%s{fcp_lun}-%s{wwpn}/part%n", NAME="%k"
```

Figure 2-13 Sample `udev` rules for DASD and FCP devices

When creating device names, `udev` uses the first matching regular expression in the ruleset file. With this ruleset, `udev` creates device nodes:

- ▶ DASD devices:
 - `/dev/dasd n` for the entire device (for instance, `/dev/dasda`)
 - `/dev/dasd nm` for each partition on the device (for instance, `/dev/dasda1`)

In addition, symbolic links of the form `/dev/dasd/0.0. $nnnn$ /disk` and `/dev/dasd/0.0. $nnnn$ /part m` are created (where $nnnn$ is the device unit address and m is the partition number on the device).

► FCP devices:

- /dev/sd*n* for the entire device (for instance, /dev/sda)
- /dev/sd*nm* for each partition on the device (for instance, /dev/sda1)

In addition, symbolic links of the form /dev/scsi/*lun-wwpn*/disk and /dev/scsi/*lun-wwpn*/part*m* are created (where *lun* is the LUN, *wwpn* is the device WWPN, and *m* is the partition number on the device).

Figure 2-14 on page 34 shows the device nodes created by this example.

```
# ls -lah /dev/dasd/
total 0
drwxr-xr-x 10 root root 200 Apr 18 18:03 .
drwxr-xr-x  8 root root 3.1K Apr 18 18:03 ..
drwxr-xr-x  2 root root 100 Apr 18 17:03 0.0.0201
drwxr-xr-x  2 root root  80 Apr 18 17:03 0.0.0202
# ls -lah /dev/dasd/0.0.0201
total 0
drwxr-xr-x  2 root root 100 Apr 18 17:03 .
drwxr-xr-x 10 root root 200 Apr 18 18:03 ..
lrwxrwxrwx  1 root root  11 Apr 18 17:03 disc -> ../../dasda
lrwxrwxrwx  1 root root  12 Apr 18 17:03 part1 -> ../../dasda1
# ls -lah /dev/scsi/
total 0
drwxr-xr-x  5 root root 100 Apr 18 17:03 .
drwxr-xr-x  8 root root 3.1K Apr 18 18:03 ..
drwxr-xr-x  2 root root  80 Apr 18 17:03 0x5210000000000000-0x5005076300cd9589
drwxr-xr-x  2 root root  60 Apr 18 17:03 0x5212000000000000-0x5005076300cd9589
# ls -lah /dev/scsi/0x5212000000000000-0x5005076300cd9589/
total 0
drwxr-xr-x  2 root root  60 Apr 18 17:03 .
drwxr-xr-x  5 root root 100 Apr 18 17:03 ..
lrwxrwxrwx  1 root root  9 Apr 18 17:03 disk -> ../../sda
```

Figure 2-14 Device nodes created by the sample udev ruleset

Tip: Use the udev man page when creating customized udev rules. The udevinfo command is also useful in creating custom rules as illustrated in Figure 2-15 on page 35.


```

# udevinfo -a -p /sys/block/dasda/

udevinfo starts with the device the node belongs to and then walks up the
device chain, to print for every device found, all possibly useful attributes
in the udev key format.
Only attributes within one device section may be used together in one rule,
to match the device for which the node will be created.

  looking at class device '/sys/block/dasda':
    SYSFS{dev}="94:0"
    SYSFS{range}="4"
    SYSFS{removable}="0"
    SYSFS{size}="4806720"
    SYSFS{stat}="      837      322      37488      530      1236      1291      20224      8930
0      5800      9460"

follow the class device's "device"
  looking at the device chain at '/sys/devices/css0/0.0.0001/0.0.0201':
    BUS="ccw"
    ID="0.0.0201"
    SYSFS{availability}="good"
    SYSFS{cmb_enable}="0"
    SYSFS{cutype}="3990/ec"
    SYSFS{detach_state}="0"
    SYSFS{devtype}="3390/0a"
    SYSFS{discipline}="ECKD"
    SYSFS{online}="1"
    SYSFS{readonly}="0"
    SYSFS{use_diag}="0"

  looking at the device chain at '/sys/devices/css0/0.0.0001':
    BUS="css"
    ID="0.0.0001"
    SYSFS{chpids}="74 75 48 49 00 00 00 00 "
    SYSFS{detach_state}="0"
    SYSFS{pimpampom}="f0 f0 ff"

  looking at the device chain at '/sys/devices/css0':
    BUS=""
    ID="css0"
    SYSFS{detach_state}="0"

```

Figure 2-15 Using udevinfo to create custom device naming rules

2.3.8 Distribution considerations when naming devices

In SLES9, familiar Linux 2.4 kernel device names are statically defined in `/dev`. If a device is detached from a Linux guest, on reboot udev correctly creates new symbolic links to the device node. However, symbolic links to the detached device node are *not* removed.

To illustrate, we consider a system originally configured with four DASD devices:

- ▶ Device 0201 assigned to `/dev/dasda`
- ▶ Device 0202 assigned to `/dev/dasdb`
- ▶ Device 0300 assigned to `/dev/dasdc`
- ▶ Device 0400 assigned to `/dev/dasdd`

In Figure 2-16, device 0300 is detached and the system rebooted.

```
# lsdasd
0.0.0201(ECKD) at ( 94: 0) is dasda      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0202(ECKD) at ( 94: 4) is dasdb      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0400(ECKD) at ( 94: 8) is dasdc      : active at blocksize 4096, 90000 blocks, 351 MB
# ls -lah /dev/disk/by-path
total 8.0K
drwxr-xr-x  2 root root 4.0K Apr 18 19:49 .
drwxr-xr-x  4 root root 4.0K Apr 15 21:56 ..
lrwxrwxrwx  1 root root  11 Apr 15 21:56 ccw-0.0.0201 -> ../../dasda
lrwxrwxrwx  1 root root  12 Apr 15 21:56 ccw-0.0.0201p1 -> ../../dasda1
lrwxrwxrwx  1 root root  11 Apr 15 21:56 ccw-0.0.0202 -> ../../dasdb
lrwxrwxrwx  1 root root  12 Apr 15 21:56 ccw-0.0.0202p1 -> ../../dasdb1
lrwxrwxrwx  1 root root  11 Apr 18 14:14 ccw-0.0.0300 -> ../../dasdc
lrwxrwxrwx  1 root root  12 Apr 18 14:14 ccw-0.0.0300p1 -> ../../dasdc1
lrwxrwxrwx  1 root root  11 Apr 18 19:49 ccw-0.0.0400 -> ../../dasdc
lrwxrwxrwx  1 root root  12 Apr 18 14:16 ccw-0.0.0400p1 -> ../../dasdc1
```

Figure 2-16 SLES9 symbolic links

On reboot, device 0400 (which was previous named `/dev/dasdd`) now becomes `/dev/dasdc`. Although, as expected udev has created the `/dev/disk/by-path/ccw-0.0.0400` symbolic link to `/dev/dasdc`, the previous `/dev/disk/by-path/ccw-0.0.0400` symbolic link has not been removed.

In general, this is not a problem *unless* automation scripts use the symbolic link name and expect those symbolic links to be consistent. To avoid this problem, SLES9 provides the `/etc/init.d/boot.udev` script. This reloads and updates all udev information in the `/dev` directory. However, it must be activated at IPL:

```
insserv boot.udev
```

With RHEL4, udev completely replaces and updates the /dev directory at IPL (no static device nodes are defined in the /dev directory). For more information about creating udev rules, see the *Linux 2.4 to 2.6 Transition Guide*, LNUX-14MG.

2.4 S/390 tools and utilities

The s390 tools package contains tools such as the `zipl` and `dasdfmt` commands specifically for running Linux on zSeries. For SLES9, these tools are provided in the s390-tools RPM package. With RHEL4, they are included in the s390utils RPM package. Use the `rpm` command to list the entire contents of the package:

```
rpm -qil s390-tools respective rpm -qil s390utils
```

Partial contents of the s390utils package is shown in Figure 2-17.

```
# rpm -qi s390utils
Name       : s390utils                Relocations: (not relocatable)
.
.
.
This package contains utilities related to Linux for S/390.
The most important programs contained in this package are:

- The cmstools suite to list, check, copy and cat files from a CMS volume.
- chccwdev, a script to generically change attributes of a ccw device.
- dasdfmt, which is used to low-level format eckd-dasds with
  either the classic linux disk layout or the new z/OS
  compatible disk layout.
- dasdview, which displays DASD and VTOC information and dumps the content
  of a DASD to the console.
- fdasd, which is used to create or modify partitions on
  eckd-dasds formatted with the z/OS compatible disk layout.
- getharp to query and purge address data in the OSA and HiperSockets hardware
- gethconf to configure IBM QETH function IPA, VIPA and Proxy ARP.
- tunedasd, a tool to adjust tunable parameters on DASD devices
- zipl, which is used to make either dasds or tapes bootable
  for system IPL or system dump.
- zdump, which is used to retrieve system dumps from either
  tapes or dasds.
```

Figure 2-17 Content summary of the s390utils package

2.4.1 The cmsfs1st command

The `cmsfs1st` command lists files on a CMS formatted volume accessed from a Linux guest. In Figure 2-18 on page 38, the `cmsfs1st` command output lists CMS files on the 302 minidisk (accessed as Linux device `/dev/dasdf`).

```
# lsdasd
0.0.0200(ECKD) at ( 94: 0) is dasda      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0201(ECKD) at ( 94: 4) is dasdb      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0202(ECKD) at ( 94: 8) is dasdc      : active at blocksize 4096, 180000 blocks, 703 MB
0.0.0300(ECKD) at ( 94: 12) is dasdd     : active at blocksize 4096, 900 blocks, 3 MB
0.0.0301(ECKD) at ( 94: 16) is dasde     : active at blocksize 4096, 900 blocks, 3 MB
0.0.0302(ECKD) at ( 94: 20) is dasdf     : active at blocksize 4096, 900 blocks, 3 MB
# cmsfs1st -d /dev/dasdf
FILENAME FILETYPE FM FORMAT LRECL      RECS      BLOCKS      DATE      TIME
          DIRECTOR PO F          64          6          1 4/19/2005 14:28:34
          ALLOCMAP PO F          4096         1          1 4/19/2005 14:28:34
PROFILE  EXEC      D1 V          70         22          1 4/19/2005 14:28:34
RHEL4    EXEC      D1 V          54         12          1 4/19/2005 14:28:34
RHEL4U1  EXEC      D1 V          29          9          1 4/19/2005 14:28:34
SLES8    EXEC      D1 V          25          9          1 4/19/2005 14:28:34
```

Figure 2-18 Using the `cmsfs1st` command

2.4.2 The tunedasd command

The `tunedasd` command reads and modifies performance parameters for DASD devices. Provided functions include:

- ▶ Read and reset performance statistics profile for a DASD device.
- ▶ Read and set the caching mode on the storage server.
- ▶ Reserve, release, or unconditionally reserve a device. This function is only available for ECKD devices.

Figure 2-19 on page 39 illustrates `tunedasd` usage. First, we check the caching mode for device `/dev/dasda`, then we set the caching mode for device `/dev/dasdc` to `prestige` (Sequential Prestage) with 100 cylinders to be cached.

```

# tunedasd -g /dev/dasda
normal (0 cyl)
# echo set on > /proc/dasd/statistics
# tunedasd -c prestage -n 100 /dev/dasdc
WARNING: This is a very large number of cylinders ;) 100
Setting cache mode for device </dev/dasdc>...
Done.

```

Figure 2-19 Using the dasdtune command

2.4.3 The dasdview command

The `dasdview` command displays DASD and VTOC information, and dumps DASD content to the console (to dump DASD content, specify the start point and offset and you can print the volume label and VTOC entries. Figure 2-20 uses the `dasdview` command to list information for `/dev/dasda`.

```

# lsdasd
0.0.0200(ECKD) at ( 94: 0) is dasda      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0201(ECKD) at ( 94: 4) is dasdb      : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0300(ECKD) at ( 94: 8) is dasdc      : active at blocksize 4096, 900 blocks, 3 MB
0.0.0301(ECKD) at ( 94: 12) is dasdd     : active at blocksize 4096, 900 blocks, 3 MB
0.0.0302(ECKD) at ( 94: 16) is dasde    : active at blocksize 4096, 900 blocks, 3 MB
# dasdview -i -f /dev/dasda

--- general DASD information -----
device node       : /dev/dasda
device number     : hex 200          dec 512
type              : ECKD
device type       : hex 3390        dec 13200

--- DASD geometry -----
number of cylinders : hex d0a        dec 3338
tracks per cylinder : hex f          dec 15
blocks per track    : hex c          dec 12
blocksize           : hex 1000       dec 4096

```

Figure 2-20 Using the dasdview command

Archived

Virtualization with z/VM

This chapter describes the benefits of running Linux as guest systems under z/VM and covers the following topics:

- ▶ z/VM and virtualization
- ▶ Installing z/VM 5.1
- ▶ Distributions for Linux on zSeries
- ▶ Running Linux under z/VM

3.1 z/VM and virtualization

z/VM provides a highly flexible test and production environment for deploying the latest On Demand Business solutions. Using IBM virtualization technology, z/VM provides the capability to run full-function operating systems guests such as Linux on zSeries and z/OS. Both 64-bit and 31-bit guests are supported.

3.1.1 z/VM guest support

z/VM presents a unique approach to computer operating systems. It provides each end user with an individual working environment known as a virtual machine (VM). The *virtual machine* simulates the existence of a dedicated, real machine, including server functions, storage, and I/O resources. However, virtual machines support much more than just end users. Application programs and operating systems can run in virtual machines. z/VM can support multiple z/Architecture and Enterprise Systems Architecture/390 (ESA/390) operating systems. For example, you can run multiple Linux and z/OS images on the same z/VM system that runs z/VM applications and end users. As a result, application development, testing, and production environments can share a single physical computer.

With the virtual machine capability of z/VM, you can:

- ▶ Run multiple Linux images
Many Linux images can run concurrently and independently on a single z/VM image.
- ▶ Test applications that terminate abnormally.
Using the isolation provided by z/VM, applications that cause exceptions can be tested in one VM, while production applications can continue operating normally in other VMs.
- ▶ Test new operating system releases and maintenance levels.
New operating system releases can be tested at the same time current releases continue to perform production work. You can perform installation and testing of program temporary fixes (PTFs) at the same time normal production operations are in progress.
- ▶ Provide backup facilities for the primary system.
A z/VM system is not model-dependent, and can operate on various server models, provided minimum hardware requirements are met. This enables a smaller server model with less real storage, fewer channels, and fewer DASD devices to provide backup for a larger production model, normally at a reduced level of performance.

- ▶ Perform operator training concurrent with production work processing.
The real machine does not have to be dedicated to training additional or new operators or to providing initial training when a new operating system is installed. Operator errors cannot cause termination of real machine operations.
- ▶ Simulate new system configurations before the installation of additional channels and I/O devices.
The relative load on channels and I/O devices can be determined using a simulated I/O configuration.
- ▶ Test customer-written system exits.
Customer-written system exits can be tested without disrupting production work.

Read more about zSeries virtualization capabilities on the Web at this Web site:
<http://www.ibm.com/servers/eserver/zseries/virtualization/features.html>

3.2 Installing z/VM 5.1

On major enhancement that was introduced with z/VM 5.1 is the installation from DVD. You can either use the Hardware Management Console (HMC) equipped with a DVD drive, or a workstation with a DVD drive accessible over FTP. DVD installation requires IBM Hardware Management Console Version, 1.8.0 or later. For details on installing z/VM 5.1 from DVD, consult *z/VM: Guide for Automated Installation and Service*, GC24-6099.

3.2.1 Installation to FCP-attached SCSI disk

z/VM 5.1 supports Fibre Channel Protocol (FCP) attached SCSI disks for use as both system and guest storage devices. This enables z/VM to be installed on and operate from either SCSI or traditional extended count key data (ECKD) disks.

Prior to z/VM 5.1, SCSI disk support was limited to Linux guests configured for FCP SCSI devices. Native SCSI disk support z/VM 5.1 is provided for logical units defined in an IBM TotalStorage Enterprise Storage Server (ESS) connected to a Fibre Channel fabric.

For details on how to configure and use FCP SCSI devices in a z/VM Linux guest, see *Linux for zSeries: Fibre Channel Protocol Implementation Guide*, SG24-6344.

When used by CMS and CP, SCSI disks are emulated as 9336 model 20 fixed-block-architecture (FBA) disks. With z/VM 5.1, SCSI disks can be used for

system paging, spooling, directory services, and minidisks. z/VM guests that support FBA disks (such as CMS and Linux) can use SCSI disks without requiring specific SCSI support (using FBA emulation). Figure 3-1 illustrates SCSI support in z/VM 5.1.

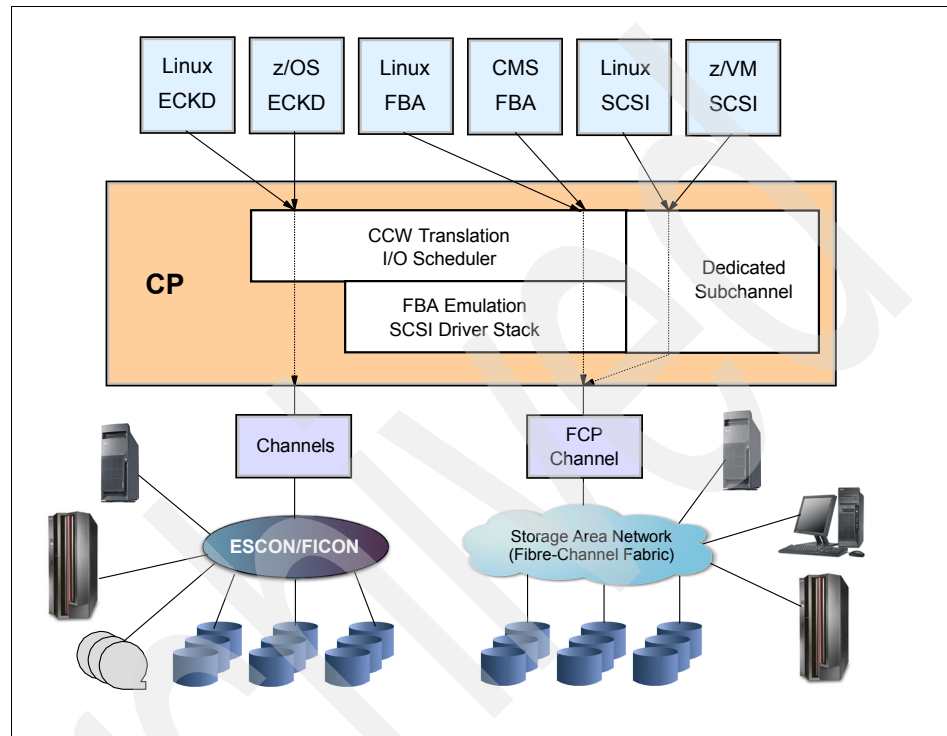


Figure 3-1 SCSI support in z/VM 5.1

z/VM supports emulated FBA disks up to one terabyte minus one page in size. However, directory, paging, and spool storage must be allocated within the first 64GB of a CP-formatted volume. Other CP allocations (such as TDSK, PERM, and PARM) can be allocated past the first 64GB.

Note: The maximum size of FBA SCSI disks allocated for use by CMS or GCS guests is 381GB. However, FBA SCSI disks used by CMS should not be larger than 22GB. CMS file system control and status data structures must reside below 16 MB in virtual storage. With larger minidisks, the system might not be able to obtain sufficient virtual storage below 16 MB to access the disks. For more details, consult *z/VM: CP Planning and Administration*, SC24-6043.

Figure 3-2 illustrates our system configuration for installing z/VM 5.1 to SCSI disk.

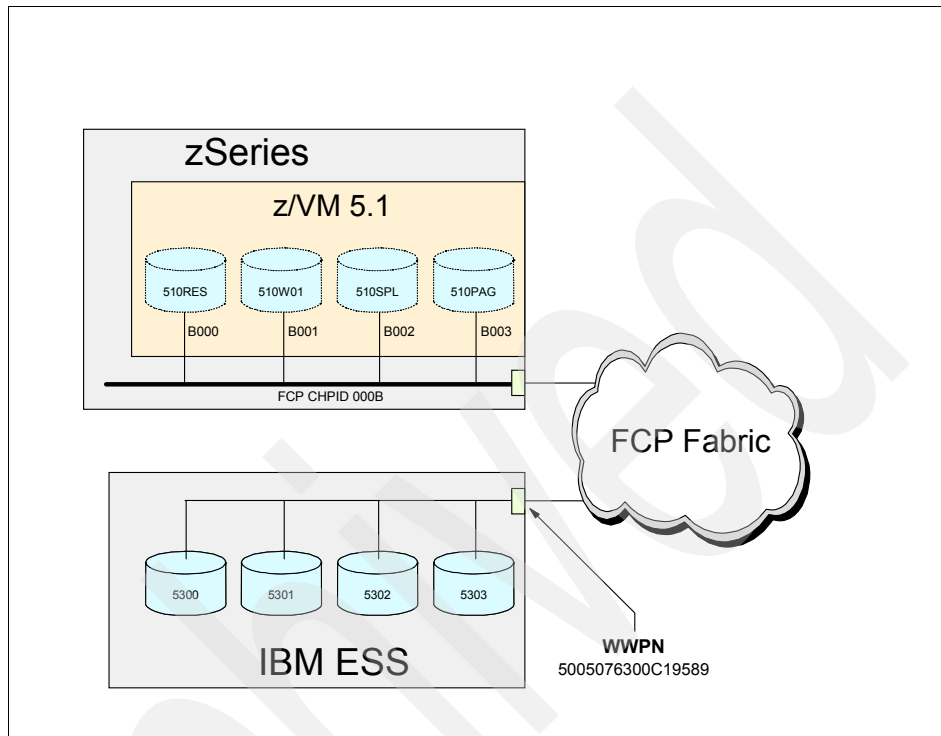


Figure 3-2 Configuration for installation to SCSI disk

In the ESS, three Logical Unit Numbers (LUNs) are defined for the z/VM system volumes:

- ▶ LUN 5300 is allocated for the 510RES volume.
- ▶ LUN 5301 is allocated for the 510W01 volume.
- ▶ LUN 5302 is allocated for the 510SPL volume.
- ▶ LUN 5303 is allocated for the 510PAG volume.

The LUNs are connected to the Fibre Channel fabric using Worldwide Port Name 5005076300C19589. The z/VM 5.1 system defines FCP devices B000-B003 connected to FCP CHPID 000B. The CHPID connects to the fabric through a FICON or FICON Express card.

Note: For details on FCP concepts and defining FCP-attached SCSI devices, consult *Linux for zSeries: Fibre Channel Protocol Implementation Guide*, SG24-6344.

To install z/VM 5.1 on a SCSI disk, an emulated FBA device must be created for the disk. z/VM 5.1 provides two methods to define an emulated device:

- ▶ The EDEVICE statement in the SYSTEM CONFIG file, described in *z/VM: CP Planning and Administration*, SC24-6043
- ▶ The CP SET EDEVICE command, described in *z/VM: CP Command and Utility Reference*, SC24-6008

Note: Emulated FBA SCSI disks can also be defined using z/VM's Hardware Configuration Manager (HCM) and Hardware Configuration Definition (HCD) support. For details, see *z/OS and z/VM: Hardware Configuration Manager User's Guide*, SC33-7989 and *z/VM: I/O Configuration*, SC24-6100.

To create an emulated FBA device for an FCP-attached SCSI disk, we need to supply:

- ▶ The Worldwide Port Name (WWPN) used to access the SCSI disk
- ▶ The Logical Unit Number (LUN) address of the SCSI disk
- ▶ The FCP device number used access the SCSI disk
- ▶ An emulated device number to access the emulated FBA disk

Table 3-1 contains the information we need to define four, emulated FBA devices for installing z/VM 5.1.

Table 3-1 SCSI disks used for z/VM 5.1 installation

Volume	WWPN	LUN	Device number	
			FCP	Emulated
510RES	5005076300C19589	5300000000000000	B000	5300
510W01	5005076300C19589	5301000000000000	B001	5301
510SPL	5005076300C19589	5302000000000000	B002	5302
510PAG	5005076300C19589	5303000000000000	B003	5303

The SET EDEVICE command syntax to define an emulated FBA device is:

```
SET EDEVICE rdev TYPE FBA ATTR 2105 fcpdev WWPN wwpn LUN lun
```

Parameters to the command include:

rdev	The emulated device number to create
fcpdev	The FCP device used to access the SCSI disk
wwpn	The WWPN on the ESS used to access the SCSI disk
lun	The LUN address of the SCSI disk

3.3 Distributions for Linux on zSeries

In this section, we look at some of the Linux distributions available for zSeries.

3.3.1 SUSE Linux Enterprise Server 9

Released in August 2004, SUSE Linux Enterprise Server 9 (SLES9) from Novell is based on the Linux 2.6 kernel. A 30-day evaluation copy of SLES9 for IBM zSeries is available at this Web site:

<http://www.novell.com/products/linuxenterpriseserver/eval.html>

Note: If you use the evaluation copy, we recommend installing the latest updates from SUSE to take advantage of any new features and any security fixes. After updating, check the `/var/adm/rpmconfigcheck` file for a list of updates that could not be applied.

Release notes for SLES9 are available on the Novell SUSE Web site:

http://www.novell.com/products/linuxenterpriseserver/release_notes_final.txt

Novell charges an annual subscription fee for SUSE SLES8 updates. The price depends on the hardware model and the number of engines used to run SLES9. For pricing details, see the Novell SUSE Web site:

<http://www.novell.com/products/linuxenterpriseserver/pricing.html>

3.3.2 Red Hat Enterprise Linux 4

Released in February 2005, Red Hat Enterprise Linux 4 (RHEL4) is based on the Linux 2.6.9 kernel. An evaluation copy of RHEL4 for IBM zSeries is available at Red Hat:

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

Note: You will receive a 30 day free access to RHEL 4 packages and we always recommend installing the latest update from Red Hat in order to benefit from new features, the most current device drivers, and security and bug fixes.

For more information about RHEL 4, see the Red Hat Web site:

<http://www.redhat.com/docs/manuals/enterprise/>

Because RHEL4 is a commercial distribution, Red Hat charges an annual subscription fee when you use their Linux distribution. The price depends on the amount of processors that are used for the Linux workload. Red Hat also offers

two types of contracts, the Standard Edition and the Premium Edition. The major difference between the two contract types is the response time when you report a problem. For the complete information about the contracts and the pricing please see the Red Hat Web site:

<http://www.redhat.com/software/rhel/compare/z/>

3.3.3 Non-commercial distributions

Several non-commercial Linux distributions are available for zSeries:

► Debian

Debian is a non-commercial Linux distribution available for several architectures, including the IBM S/390 platform. The latest stable release is Debian GNU/Linux 3.0, also called *woody*, S/390. This release is based on the Linux kernel 2.4.17. For more information, see the Debian Web site:

<http://www.debian.org>

► Slack/390

Slack/390 is a port of Slackware Linux to the IBM S/390 and zSeries distributed under GNU Public Licence (GPL). For more information, consult the Slack/390 Web site:

<http://www.slack390.org>

Note: Commercial support for Slack/390 can be obtained from Sine Nominee Associates. For details, consult Sine Nominee's Web site:

<http://sinenomine.net/node/484>

► Build your own

IBM provides patches on developerWorks that are meant to be applied to the vanilla versions of the kernel, gcc, gdb, and so forth. You can get these packages from the official Web sites where they are hosted.

<http://www.kernel.org>

http://www.ibm.com/developerworks/linux/linux390/april2004_recommended.shtml#downloads

<http://www.gnu.org>

3.3.4 IBM middleware for Linux on zSeries

Many IBM middleware products run on Linux for zSeries. A complete list of IBM products that run on Linux, consult *IBM Middleware Available on Linux*:

<http://www-1.ibm.com/linux/matrix/linuxmatrixhwz.html>

Specific IBM middleware product availability for can be found from the Web sites listed in Table 3-2.

Table 3-2 IBM middleware available for Linux on zSeries

IBM middleware product	URL
Data management	http://ibm.com/zseries/os/linux/db2.html
Lotus Domino	http://ibm.com/zseries/os/linux/lotus.html
Rational	http://ibm.com/zseries/os/linux/rational.html
Tivoli	http://ibm.com/zseries/os/linux/tivoli.html
WebSphere	http://ibm.com/zseries/os/linux/websp.html

3.4 Running Linux under z/VM

In this section, we look at some of the factors to consider when running Linux guests under z/VM.

3.4.1 Managing guest virtual memory

One of the major values of running Linux guests under z/VM is the ability to overcommit real memory. With z/VM, the sum of virtual memory allocated to Linux guests typically exceeds the amount of real memory in the LPAR. z/VM paging algorithms are optimized to overcommit memory.

The extent to which memory can be overcommitted depends workloads running on the z/VM system. For example, a system might run with 95% of its Linux guests idle 95% of the time. They are never all active at the same time. In this case, it might be possible to overcommit memory by a ratio of 15:1, virtual to real memory. By contrast, a system where all Linux guests are always busy can require a 2:1 ratio of virtual:real memory.

3.4.2 Managing z/VM storage

z/VM uses three types of storage:

- ▶ *Main storage* is allocated from physical memory. Programs execute in main storage. In addition, all I/O operations occur in main storage.
- ▶ *Expanded storage* is also allocated from physical memory. Expanded storage is addressable only as whole pages. Expanded storage acts as a fast paging device. As demand for main storage increase, z/VM can quickly page to expanded storage.

► Paging space

Paging space resides on DASD. When demand exceeds the capacity of expanded storage, z/VM uses paging space.

Figure 3-3 illustrates the three types of z/VM storage.

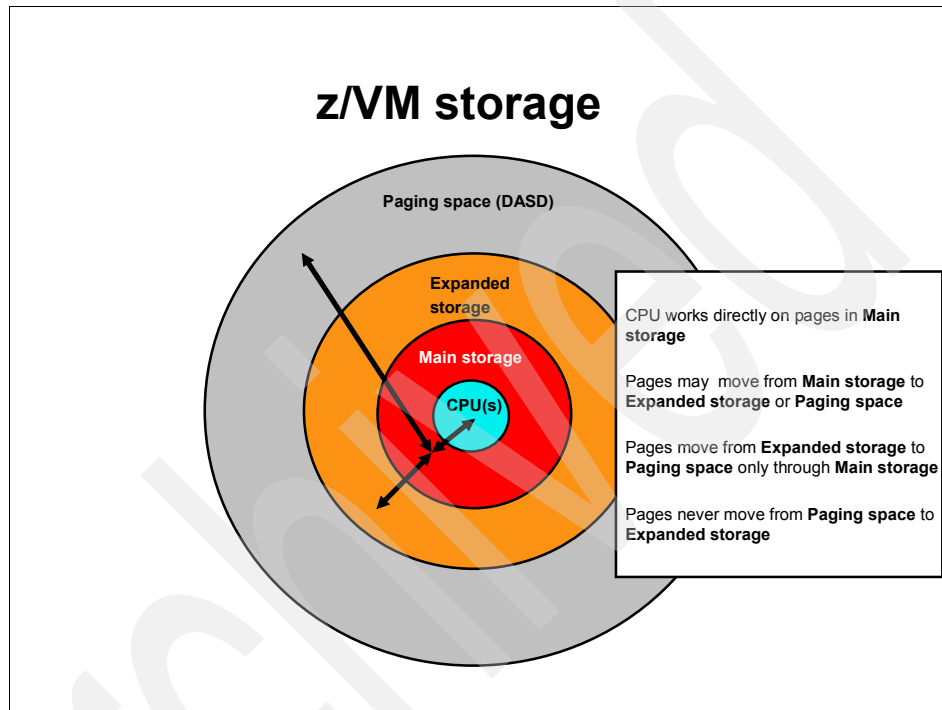


Figure 3-3 z/VM storage usage

Configuring expanded storage

Always configure some system memory as expanded storage. The VM paging subsystem is optimized for expanded storage, and can achieve higher overall throughput when expanded storage is available.

Expanded storage should be used even with a 64-bit address space. For example, in a system with 10 GB of real storage, it is usually better to define 8 GB of main storage and 2 GB of expanded storage (rather than defining the entire 10 GB as main storage).

Even with 64-bit support, z/VM 5.1 still performs I/O below the 2 GB bar. This means that Linux guests performing I/O require VM system memory pages below 2 GB. In the event of memory contention below 2 GB, z/VM can utilize expanded storage, rather than move pages to DASD.

Tip: When defining expanded storage, begin by configuring 25% of available memory as expanded storage.

DASD paging subsystem

When configuring z/VM paging subsystem:

- ▶ Allocate entire DASD volumes to page space.

Do not define nonpaging storage (minidisk or spool space) on VM paging volumes. The paging subsystem uses seldom-ending channel commands to achieve high paging throughput. Minidisks or spool space on paging volumes can interrupt paging channel activity and cause lower paging throughput.

- ▶ **Use more, smaller paging volumes.**

More small volumes provide better performance than fewer large volumes. I/O operations to a single physical disk are serialized. With three 3390-3 volumes, three I/O operations can proceed in parallel. A single 3390-3 volume can only process one I/O operation at a time.

- ▶ **Distribute paging volumes across control units and I/O paths.**

On an IBM ESS, a single logical disk is mapped to several physical disks. To avoid contention, ensure the disks are distributed to different control units and RAID-5 arrays.

Note: On an IBM ESS, volumes on the same control unit with consecutive device numbers are most likely all on the same array. VM paging activity for a stressed system can easily saturate a single array on an ESS.

When defining VM paging space, begin by allocating an amount equal to the sum of the virtual storage size of all guests. Under load conditions, the system paging space utilization averages should be less than 50%. If not, add more paging space.

Tip: The CP Q **ALLOC PAGE** command shows current paging utilization.

VDISK swap device

Linux systems should always be defined with some swap space, even if they are not expected to need it. For Linux systems that are expected to swap, VDISK offers a attractive option as a fast swap device. VDISK is a virtual disk in memory accessed using the fba_dasd device driver. To Linux, VDISKS appear as a real disk devices.

Because it resides in memory, access to the VDISK is extremely fast. With a VDISK, most memory pages are not allocated until actually needed, when Linux actually writes to the swap device. However, some memory is allocated for the control blocks VM needs manage the device when the VDISK is defined.

Note: With VM 5.1 system, VDISK control blocks are allocated below 2GB. Avoid using VDISKs on system with contention for main storage below 2GB.

DCSS swap device

A new option for a virtual swap device is *Discontiguous Storage Segment* (DCSS), a separate memory area that is not part of the guest's virtual memory space. DCSS avoids a priority inversion problem that exists with VDISK devices. VDISK devices were designed to hold very important data areas such as lock files. VM prefers to steal memory pages from a running guest rather than from a VDISK. This means that VM might steal pages from your running guest to allocate them to the VDISK.

Linux guest virtual machine size

To reduce the likelihood of performing I/O, Linux attempts to uses all available memory for file system buffers. For this reason, the virtual machine size for Linux guests should be as small as possible. A Linux system that does not require 1GB of memory will eventually fill any unused memory with file system buffers. Because main storage is shared among all VM guests, this can adversely affect the VM scheduler, discussed in 3.4.3, “Managing the VM scheduler” on page 53. In general, more small Linux guests get better response time than fewer large ones.

As a rule of thumb, to find an optimal virtual machine size for a Linux guest:

1. Initially define the guest virtual machine size to be about two-thirds as large as the recommended memory size for the applications running on the guest.
2. Define a VDISK or DCSS swap device for the guest. Use a size that is twice the guest virtual machine size.
3. Start the guest and run the application with a realistic workload test. Monitor memory consumption and swap device activity during the test.

Optimal virtual machine size can be inferred based on swap device activity:

- ▶ If the guest never swaps, the virtual machine size is likely too large and should be reduced.
- ▶ If swapping affects performance, the virtual machine size is too small and should be increased.

- ▶ If the guest begins to swap as load increases, the guest virtual machine size is correct. Ideally, as the workload reaches steady state, Linux swapping slows down or even stops.

3.4.3 Managing the VM scheduler

The VM scheduler attempts to keep as many logged-on virtual machines running concurrently as possible, based on availability of processor cycles, real storage, and paging space. Logged-on virtual machines reside on one of three lists:

- ▶ *Dormant list* contains virtual machines with no immediate task to perform.
- ▶ *Eligible list* contains virtual machines waiting for a system resource.
- ▶ *Dispatch list* contains virtual machines contending for processor time.

As virtual machines have tasks to perform, they move between the three lists as shown in Figure 3-4.

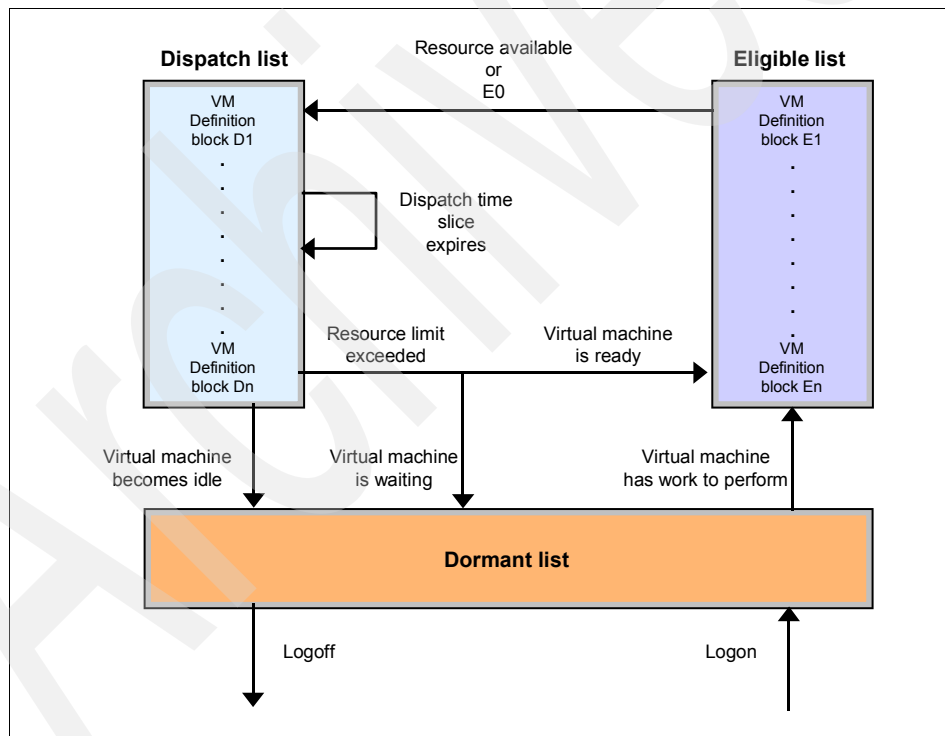


Figure 3-4 VM scheduler lists

Virtual machines on the dispatch list are characterized based on their resource consumption:

- ▶ Q1 virtual machines have not recently consumed their entire CPU time slice. These guests are considered interactive resource consumers such as users editing a VM configuration file or executing short-running VM commands.
- ▶ Q2 virtual machines have consumed at least one entire CPU time slice in Q1. These guests are considered to be intermediate resource consumers such as users executing long-running VM commands.
- ▶ Q3 virtual machines have consumed at least two entire CPU time slices, one in Q1 and one in Q2. Linux systems typically fall into Q3, as do other operating systems running under z/VM.
- ▶ Q0 virtual machines are critical users that do not wait for the resources to become available. These virtual machines are discussed in “The QUICKDISP option” on page 55.

Virtual machines on the eligible list are assigned a corresponding E1, E2, E3, or E0 classification.

SRM settings

Virtual machines wait on the eligible list for resources to become available before moving to the dispatch list (if no resource constraints exist, virtual machines move immediately to the dispatch list). Resource availability is determined by SRM settings and virtual machine classification.

The CP SET SRM command is used to adjust SRM settings. The critical SRM settings are **LDUBUF** and **STORBUF**:

- ▶ **LDUBUF** controls allocation of paging resources.

Use the **SET SRM LDUBUF *i j k*** command to adjust this control:

i Sets the percentage of paging space available to Q1 virtual machines.

j Sets the percentage of paging space available to Q1 and Q2 virtual machines.

k Sets the percentage of paging space available to Q1, Q2, and Q3 virtual machines.

For systems that support only Linux guests, an appropriate LDUBUF setting is:

```
SET SRM LDUBUF 100 100 100
```

For systems with a significant number of CMS users, an appropriate LDUBUF setting is:

```
SET SRM LDUBUF 100 90 90
```

- ▶ **STORBUF** controls allocation of real storage.

Use the **SET SRM STORBUF *i j k*** command to adjust this control:

- i* Sets the percentage of main storage available to Q1 virtual machines.
- j* Sets the percentage of main storage available to Q1 and Q2 virtual machines.
- k* Sets the percentage of main storage available to Q1, Q2, and Q3 virtual machines.

For systems that support only Linux guests, an appropriate STORBUF setting is:

```
SET SRM STORBUF 125 125 125
```

For systems with a significant number of CMS users, an appropriate STORBUF setting would be:

```
SET SRM STORBUF 200 150 125
```

Note: It is possible to over-commit resources by specifying values greater than 100. When over-committing real storage, be sure to allocate enough paging space. Setting this value too high can lead to system thrashing.

As an example, we consider a system with 512MB of real storage running eight Linux guests. Each within a virtual memory size of 128 MB. If the STORBUF setting is 125 125 125, then five guests will be in Q3, and the remaining three will be in E3. In order to be dispatched, the E3 guests must wait until a guest in Q3 goes idle, or until the time slice for a Q3 guest expires.

The QUICKDISP option

As noted, the Q0 classification for critical virtual machines that do not wait for resource availability on the eligible list. Instead, these guests move immediately to the dispatch list. To designate a virtual machine as Q0, use the CP SET QUICKDISP command. Alternatively, add the OPTION QUICKDISP statement to the user's directory entry.

The QUICKDISP option is valuable in preventing service machines from becoming a bottleneck for other work. For instance, if the TCP/IP stack is placed on an eligible queue, all TCP/IP connectivity to the VM system stops until TCP/IP is runnable again.

Important: Adding QUICKDISP to guests should be done carefully. It should only be assigned to guests that service other guests, such as TCP/IP, RACF, or other vital subsystems. QUICKDISP should definitely be assigned to the guest used to maintain the VM system.

SHARE settings

Share settings control the priority of guests in the dispatch list when contention for processor time exists. There are two types of share settings:

- ▶ *Absolute* share allows a guest access to a specified percentage of CPU resource. With an absolute share of 3% a guest gets 3% of the CPU, regardless of how many other virtual machines are running. This is useful for workloads with a well known usage pattern. It ensures a guest is not completely starved for CPU time, provided SRM settings allow it to stay in the dispatch queue. The responsiveness of a guest with an absolute share does not slow down as the system load increases.
- ▶ *Relative* share allows a guest to have its importance defined relative to other running guests. Two virtual machines with a relative share of 100 have equal access to the CPU resources. A guest with a relative share of 200 has twice as much access to CPU resources as a guest with relative share of 100. Relative share is suitable for workloads without a well defined usage pattern.

Note: Share settings are a more consistent way to assign processor resources than changing the number of virtual CPUs defined to a guest. When defining a Linux guest user directory entry, define as many virtual CPUs as there are physical CPUs in the LPAR. Share settings can then be used to dynamically adjust the priorities of the guests running on the system.

Logical CPUs

On systems with more than one physical processor, assign as many logical CPUs as there are physical CPUs in the LPAR. There is no performance benefit in allocating more logical CPUs than available physical CPUs. The logical CPU units of work will simply queue up on the available physical CPUs and be executed in sequence.

Note: It can be advantageous to define more logical CPUs than physical CPUs when testing multi-threaded code. For instance, defining 16 logical CPUs on a system with two physical processors can expose timing and locking problems during system test.

3.4.4 Performance monitoring

Performance monitoring is a tool for collecting performance data critical for running a VM production system. The only way to correctly identify a performance problem is with data. Two tools which can provide that data are the VM Performance Toolkit, and Velocity Software's ESALPS tool suite.

The VM Performance Toolkit is pre installed with z/VM 5.1, but it is separately licensed from VM and is not free.

Velocity Software's ESALPS tool suite is another tool that can collect performance data to assist in diagnosing VM performance problems.

Both tools provide data on guest CPU utilization, memory utilization, general I/O activity, Paging activity, network activity, and much more. It is nearly impossible to diagnose the cause of performance problems without a performance analysis tool.

Archived

Archived

Networking Overview

This chapter describes the various networking options available for your Linux on zSeries servers. In this chapter, we discuss the following topics:

- ▶ Physical networking options
- ▶ HiperSockets
- ▶ Virtualization technology
- ▶ Virtual Switch
- ▶ Introduction to VLANs
- ▶ Layer 2 test scenario

4.1 Networking options

The networking options available to Linux on zSeries can be split broadly into two categories: physical hardware and virtualization technology. *Physical hardware*, as the term suggests, covers physical network interfaces, or in the case of HiperSockets, a networking implementation that requires zSeries hardware. *Virtualization technology* covers the networking options available to those users who run Linux in a z/VM environment.

The z/VM operating system can use any of the physical networking options. Linux systems running as virtual machines in a z/VM environment have the choice of using any of the physical options, any of the virtualization technology options, or a combination of both, including:

- ▶ Physical networking options
 - Open Systems Adapter-2 and Open Systems Adapter-Express
 - Channel-to-channel adapter
 - Common Link Access to Workstations (CLAW)
 - HiperSockets
- ▶ Virtualization technology
 - Point-to-point connectivity
 - Guest LAN
 - z/VM Virtual Switch (VSWITCH)
 - Layer 2 LAN Switching

4.2 Physical networking options

In this section, we look at the type of physical networking connections available for Linux on zSeries.

4.2.1 Open Systems Adapter-2 (OSA-2)

The Open Systems Adapter-2 (OSA-2) card was the first open systems connection interface available on S/390 servers. On zSeries servers, it has been replaced to by the OSA-Express card. The exception is that the z900 server supports the OSA-2 ENTR card as the OSA-2 token-ring feature. However, the OSA-2 card is not discussed in this book.

4.2.2 Open Systems Adapter-Express (OSA-Express)

The Open Systems Adapter-Express (OSA-Express) Gigabit Ethernet (GbE), 1000BASE-T Ethernet, Fast Ethernet (FENET), token-ring, and Asynchronous

Transfer Mode (ATM) are next generation cards that supersede the OSA-2 family of cards. OSA-Express cards provide significant enhancements over OSA-2 in function, connectivity, bandwidth, network availability, reliability, and recovery. OSA-Express cards are available for all zSeries processors and S/390 G5 and G6 processors. The OSA-Express2 Gigabit Ethernet short wavelength (GbE SX) and long wavelength (GbE LX) features, as well as the 10 Gigabit Ethernet Long Reach (10 GbE LR) feature, are the newest members of the OSA-Express family for zSeries servers. They provide increased bandwidth over the OSA-Express features. See Figure 4-1 on page 62 for a diagram of OSA-Express connectivity options.

Terminology: If not specifically stated, the term *OSA-Express* applies to both the OSA-Express and the OSA-Express2 features throughout this book.

Each OSA-Express card has one port on G5 and G6 servers and two ports on zSeries servers. The OSA-Express2 zSeries GbE SX and LX cards have two ports and the 10 GbE LR has only one port. Each individual port has a unique CHIPD associated with it. They can be attached directly to a LAN or ATM network. For Linux connectivity, these cards are recognized by the hardware I/O configuration as one of the following channel types:

- ▶ OSD (Queued Direct I/O)
- ▶ OSE (Non-Queued Direct I/O)

Note: For detailed descriptions of OSA-Express operating modes and limitations, refer to the IBM Redbook *OSA-Express Implementation Guide*, SG24-5948.

QDIO mode

Queued Direct I/O (QDIO) is a highly efficient data transfer mechanism. It reduces system overhead and improves throughput by using system memory queues and a signaling protocol to directly exchange data between the OSA-Express microprocessor and TCP/IP stack.

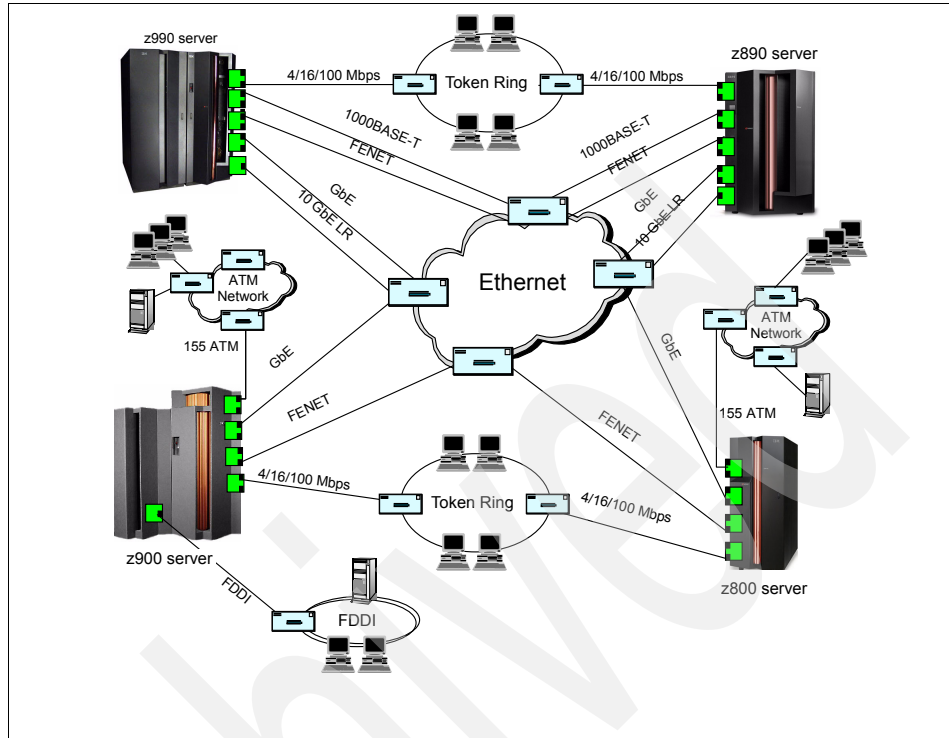


Figure 4-1 OSA-Express connectivity

QDIO versus non-QDIO

Figure 4-2 on page 63 illustrates the much shorter I/O path length of QDIO mode compared to non-QDIO mode. In QDIO mode, I/O interrupts and I/O path lengths are minimized and measurements have shown that there is a significant improvement in performance versus non-QDIO mode, in particular, a reduction of System Assist Processor (SAP) utilization and improved response time.

Benefits of running in QDIO mode include:

- ▶ Dynamic OSA Address Table (OAT) update
- ▶ LPAR-to-LPAR communication
- ▶ Internet Protocol (IP) Assist functions
- ▶ Checksum offload

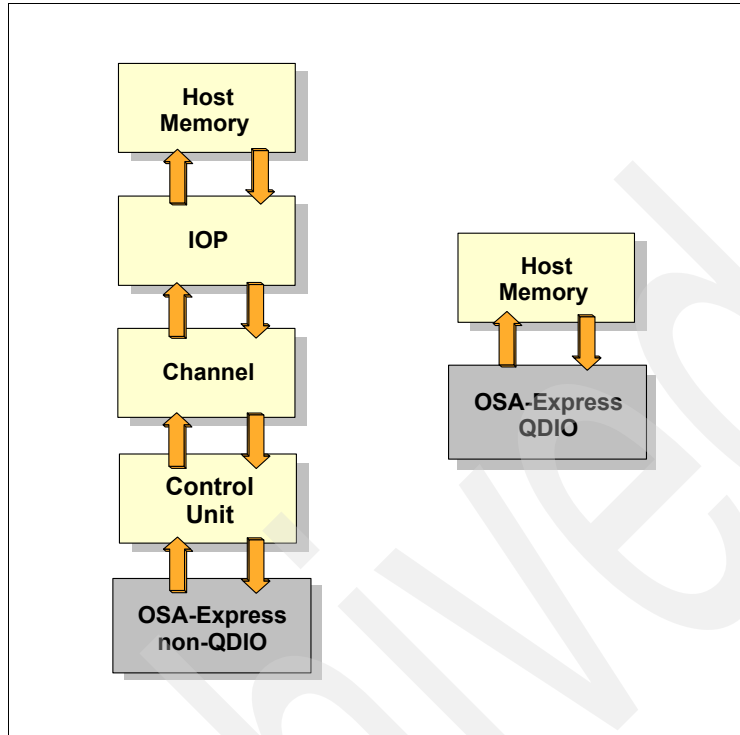


Figure 4-2 QDIO and non-QDIO data paths

Dynamic OSA Address Table (OAT) update

The TCP/IP stack of each operating system that shares a port on an OSA-Express card in QDIO mode dynamically registers all its IP addresses with the card. Whenever IP addresses are deleted from or added to a network stack, the device drivers download the resulting IP address list changes to the OSA-Express card.

For OSA-Express cards shared by multiple systems, this removes the requirement to manually enter the information into the OAT using Open Systems Adapter/Support Facility (OSA/SF), which is a program product that runs on z/OS or z/VM. A user might still have a requirement to use OSA/SF however, for example, if you need to enter SNA definitions into the card when the card is running in non-QDIO mode.

Transport Modes

For IP and non-IP workloads, the OSA-Express2 features and the current OSA-Express features on z890 and z990 systems support two transport modes:

Layer 2 (Link or Ethernet Layer) and Layer 3 (Network or IP Layer). Each mode processes network data differently.

Layer 2 has these characteristics:

- ▶ Uses the MAC destination address to identify hosts and send and receive Ethernet frames.
- ▶ Transports Ethernet frames (not IP datagrams) to and from the operating system TCP/IP stack and the physical network.
- ▶ Does not ARP offload; ARP processing performed by each operating system TCP/IP stack.
- ▶ Supports MAC level unicast, multicast, and broadcast.

Layer 3 has the following characteristics:

- ▶ Data is transmitted based on IP address.
- ▶ Only IP addresses are used to identify hosts on the LAN segment.
- ▶ All Address Resolution Protocol (ARP) processing is done by the adapter itself, not by any operating system sharing the adapter. See “Address Resolution Protocol and OSA-Express” on page 67 for more information.
- ▶ A single Media Access Control (MAC) address is managed by the adapter for all guests sharing the adapter port.

LPAR-to-LPAR communication

Using EMIF, a port on the OSA-Express card can be shared across multiple LPARs, as depicted in Figure 4-3 on page 65. Also, access to a port on the card can be shared concurrently among multiple TCP/IP stacks within the same LPAR.

When port sharing, the OSA-Express card running in QDIO mode has the ability to send and receive IP traffic between LPARs without sending the IP packets over the network.

For outbound packets, OSA-Express uses the next-hop address provided by the TCP/IP stack to determine where to send the packet. If this next-hop address had been registered by another TCP/IP stack sharing this OSA-Express, the packet is delivered directly to that TCP/IP stack, and not sent out over the LAN. This makes possible the routing of IP packets within the same host system.

Restriction: Port sharing is supported only between ports that are of the same transport mode, for example Layer 2 with Layer 2 and Layer 3 with Layer 3. Attempted communications between a Layer 2 connection and a Layer 3 connection sharing the same OSA-Express port result in a network timeout condition. To resolve this, you should have the Layer 2 connection and the Layer 3 connection on separate OSA-Express ports that are connected to the same LAN segment. With this solution, the communication between these connections is now sent out onto the physical LAN segment, and full MAC resolution will be achieved.

LPAR-to-LPAR communication also applies to OSA-Express FENET when the mode is non-QDIO.

Note: HiperSockets for zSeries also provides a highly efficient way to communicate between different LPARs, with better throughput. See 4.3, “HiperSockets” on page 83 for more information.

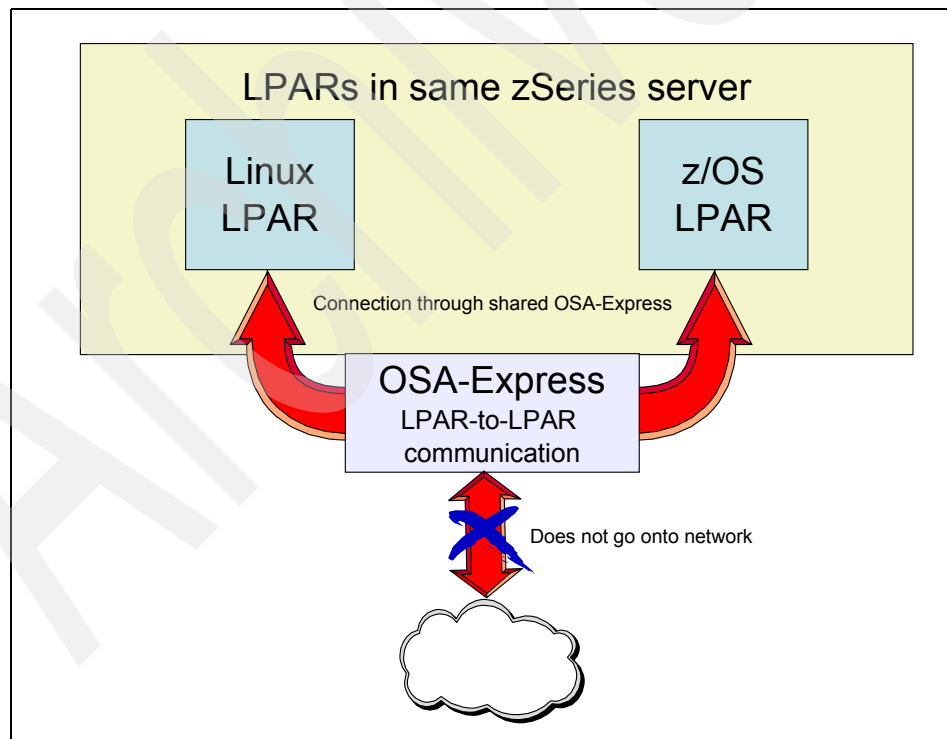


Figure 4-3 LPAR-to-LPAR communication

Internet Protocol (IP) Assist functions

When an OSA-Express port is configured for Layer 3 mode, the QDIO microcode assists in IP processing and offloads the TCP/IP stack functions for the following:

- ▶ Multicast support
- ▶ Broadcast filtering
- ▶ Building MAC and LLC headers
- ▶ ARP processing

Offloading the processing of these functions means that CP cycles are freed up to do other work. In a single guest, the effect might not be significant, but in a z/VM LPAR with Linux guests generating a moderate-to-high volume of network traffic, there will be an overall storage saving.

Checksum offload for Linux and z/OS

Checksum processing calculates the TCP/UDP and IP header checksums to verify the integrity of data packets. This function is usually performed by a host system's TCP/IP stack. OSA-Express cards on the z990 and z890 processors have the ability to perform checksum processing on behalf of the upstream TCP/IP stack using a function called checksum offload. This function is only available for IPv4 packets and is not available in Layer 2 mode.

By moving the checksum calculations to an OSA-Express Gigabit or 1000BASE-T Ethernet card, host CPU cycles are reduced. This support is available with z/OS V1R5 and later and Linux for zSeries.

Note: To use checksum offload with Linux for zSeries, you must set the checksumming attribute of the qeth device to `hw_checksumming`. Linux for zSeries supports inbound checksum offload for inbound packets only. Refer to *Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281*, for additional information about the checksumming attribute of the qeth device driver.

Non-QDIO mode

When running in non-QDIO mode, a port on the OSA-Express card is defined as channel type OSE.

Note: The zSeries OSA-Express cards have two ports. Each port has an associated CHPID. It is possible to configure one CHPID as type OSD (QDIO) and one CHPID as OSE (non-QDIO), or both CHPIDs as OSD or OSE.

In non-QDIO mode, the data follows a longer I/O path as discussed previously. Linux uses the LCS device driver to communicate with the device when it is

running in this mode. The non-QDIO mode requires the use of OSA/SF for customization of the OSA-Express if you want to share the card across multiple LPARs or Linux guests.

The OSA-Express 1000BASE-T, FENET, and token-ring cards support both non-QDIO and QDIO modes. The OSA-Express Gigabit Ethernet card and all OSA-Express2 cards only support QDIO mode.

Unless you have a specific requirement (such as supporting SNA traffic), we recommend that you always run the OSA-Express card in QDIO mode.

Address Resolution Protocol and OSA-Express

Address Resolution Protocol (ARP) is a networking protocol used to resolve IP addresses to physical hardware addresses. These hardware addresses are known as Media Access Control (MAC) addresses.

When an application running on machine X wants to send a datagram to machine Y, it typically uses machine Y's IP address as the address that it uses to try and reach the destination. However, the device driver controlling the Network Interface Card does not understand IP addresses and wants to send the datagram using a MAC address as the destination address.

The ARP protocol attempts to resolve the IP address into a MAC address. It does this by referencing a lookup table (called an ARP cache). If the address is not found in the ARP cache, an ARP request is broadcast over the network. If one of the machines that receives the broadcast recognizes its own IP address, it answers the requesting machine with an ARP reply message. This reply includes the MAC address of that host. This information is then stored in the requesting system's ARP cache. Any subsequent datagrams to this destination IP address are translated to a MAC address by referring to the ARP cache.

In the majority of computer systems, a network card is owned by a single TCP/IP stack; therefore, there is a one-to-one relationship between the IP address and MAC address. Figure 4-4 on page 68 illustrates MAC address and IP address processing in a distributed environment.

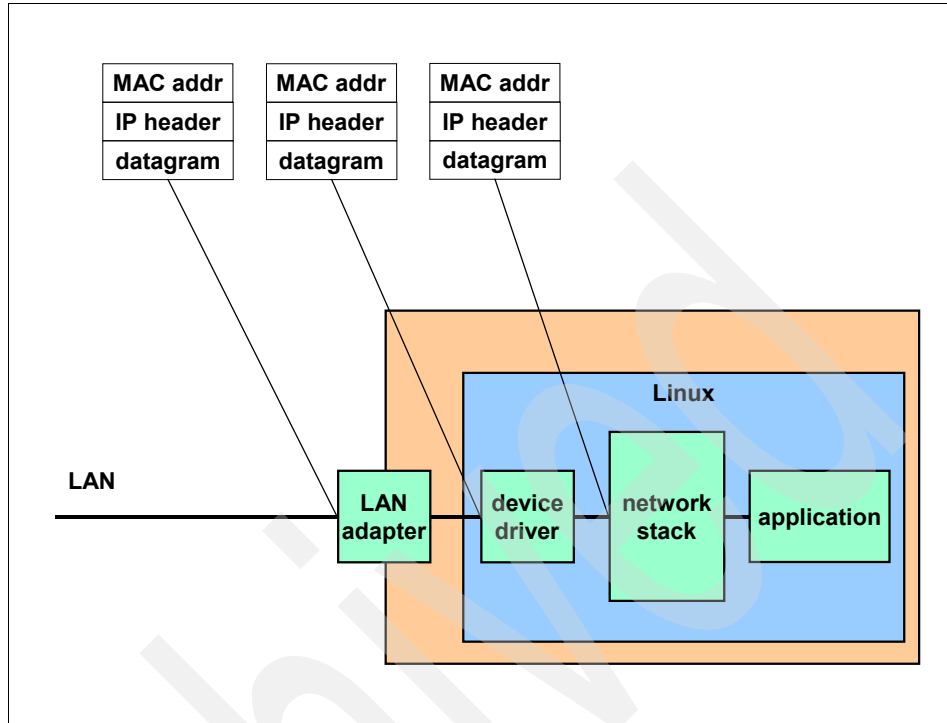


Figure 4-4 Packet processing in non-mainframe environments

OSA ports can be shared across multiple LPARs or guests. When the port is operating in Layer 3 mode, each TCP/IP stack running in these systems has a unique IP address that is dynamically registered with the OSA-Express port. However, all of the registered IP addresses are associated with the same MAC address. The OSA-Express port responds to ARP requests from other machines in the network for any IP address that is registered in the port.

The OSA-Express port removes the Logical Link Control (LLC) header, which includes the MAC address from incoming IPv4 packets, and uses the registered IP address to forward packets to the recipient TCP/IP stack. This is how the port delivers IPv4 packets in Layer 3 mode to the correct Linux image. Apart from broadcast packets, a Linux image can only receive packets for IP addresses it has configured in the stack and registered with the OSA-Express port. See “Primary and secondary router function” on page 70 for the only exception to this statement.

As the OSA-Express QDIO microcode builds LLC headers for outgoing IPv4 packets and removes them from incoming IPv4 packets, the operating system’s network stacks only send and receive IPv4 packets without LLC headers.

Figure 4-5 illustrates MAC address and IP address processing by OSA-Express in Layer 3 mode.

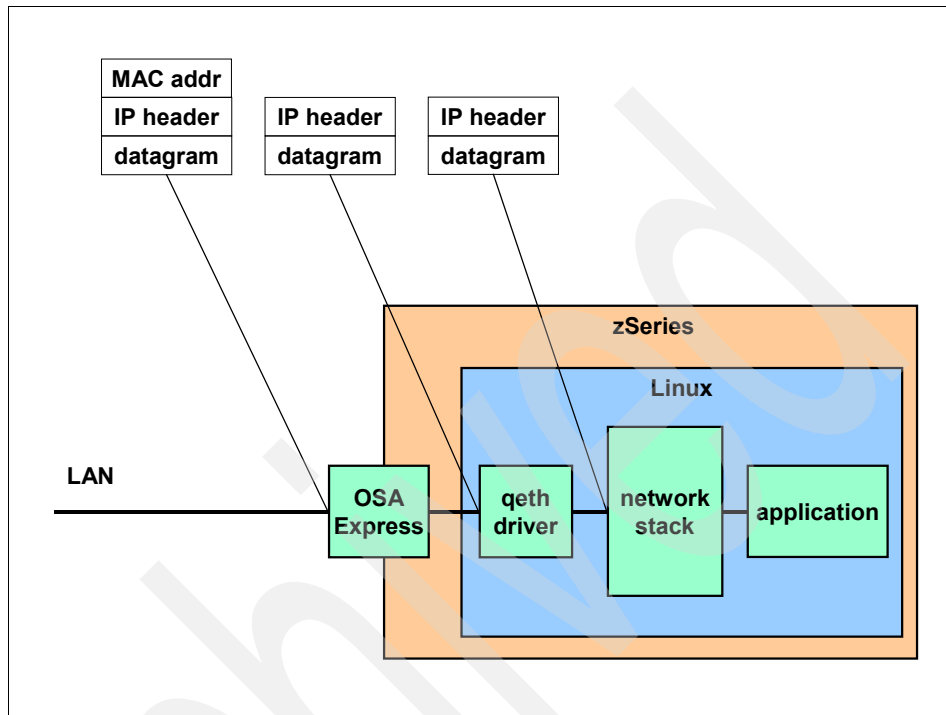


Figure 4-5 Packet processing by OSA-Express

Letting the OSA-Express hardware handle the LLC header allows multiple operating systems to share an OSA-Express adapter port. Usually, LLC processing by the OSA-Express port also yields better performance than letting the Linux images that share the port handle the LLC header themselves. For IPv6, the OSA-Express port in QDIO mode passes complete packets to the Linux image, and the driver lets the network stack compose packets with an LLC header.

If the OSA-Express port is operating in Layer 2 mode, there is once again a one-to-one relationship between the IP address and MAC address. But the OSA port can still be shared by multiple TCP/IP stacks. Each TCP/IP stack sets a MAC address for itself into the OSA and the TCP/IP stack is responsible for responding to ARP requests with its MAC address. If a z/VM VSWITCH is used in layer 2 mode, z/VM sets the MAC address on behalf of the TCP/IP stack. See , “Planning for unique MAC addresses” on page 127.

Primary and secondary router function

A port on an OSA-Express card that is operating in Layer 3 mode can be configured to forward unknown IP addresses to a particular TCP/IP stack for routing. For example, a Linux system could act as a router to provide a means of connecting an external LAN segment to systems running within a zSeries machine on a different subnet.

In order for the OSA-Express port to forward datagrams to a specific TCP/IP stack, the Linux system must have set the `primary_router` setting in the OSA-Express device driver. If the OSA-Express port is being shared by multiple systems, only one of those systems can act as the primary router. Refer to *Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281*, for detailed information about the `primary_router` setting.

Hardware configuration

The IOCP statements in Figure 4-6 show an OSA-Express card, CHPID 01. The card is being shared by two LPARs.

```
ID MSG1=' IOCP DECK' ,
MSG2=' SYS6.IODFF1 - 04-03-02 15:25'
RESOURCE PARTITION=((SC47,2),(SC69,8))
CHPID PATH=(01),SHARED,PARTITION=((SC47,SC69),(SC47,SC69)),TYPE=OSD
CNTLUNIT CUNUMBR=C100,PATH=(01),UNIT=OSA
IODEVICE ADDRESS=(C100,015),CUNUMBR=(C100),UNIT=OSA
IODEVICE ADDRESS=C10F,UNITADD=FE,CUNUMBR=(C100),UNIT=OSAD
```

Figure 4-6 OSA-Express IOCP statements

Using EMIF, devices can be shared across multiple LPARS (for example, devices C100-C102) and within LPARs. Devices C100-C102 can be used on each LPAR. Within a single LPAR, device addresses must be unique, so one TCP/IP stack can use addresses C100-C102 and a second stack can use devices C104-C106. On the z990 and z890, there is a limit of 480 total devices per OSA-Express port. The OSA-Express2 feature raises this limit to a maximum of 1920 total devices per port.

A more in-depth review of the OSA-Express and OSA-Express2 features are beyond the scope of this book. Refer to the IBM Redbook *OSA-Express Implementation Guide, SG24-5948*.

z/VM considerations

Multiple virtual machines can share the same physical OSA port. All guests can be defined with the same three *virtual* device addresses for the OSA interface. However, each guest must use a unique set of three *real* device addresses.

The syntax of the DEDICATE statement is:

```
DEDICATE virtual_address real_address
```

For example, you might choose C200-C202 as the *virtual* addresses for all of your guests. Each guest must, however, have unique *real* addresses. So in the first guest, you might use addresses C200-C202, in the second guest, C203-C205, and so on.

Important: The first device address is the OSA read device. It must be an even-numbered device. The second device is the OSA write device, and its address must be one greater than the read device. Using the examples above, you specify the device list C200, C201, C202 as the first device to the OSA-Express device driver. See the next section for the syntax and details on how to do this. The device list for the C203-C205 devices would be C204, C205, C203. By using this convention, we use every device address and thus do not waste addresses by skipping to the next even-numbered address to start the next device set. For example, C200, C201, C202 followed by C204, C205, C206 would mean we waste C203, and therefore, we are not able to fully use the OSA-Express device.

Using OSA-Express with Linux

As we previously discussed, a port on an OSA-Express card can run in one of two modes, QDIO or non-QDIO.¹ Here, we review the parameters for running the card in QDIO mode. In non-QDIO mode, the OSA-Express port acts like as an Interconnect Controller, and Linux for zSeries uses the LAN Channel Station (LCS) device driver (lcs.o) to control the port. The LCS device driver is not discussed in this book.

In QDIO mode, the following two modules are required in order for Linux to use the OSA-Express port.

- ▶ `qdio` controls the interface between the processor and the OSA-Express CHPID.
- ▶ `qeth` controls the OSA-Express port.

Two other modules, `ipv6` and `8021q`, are typically loaded automatically to provide support for IPv6 and Virtual LANs. For our purposes, we only need to configure the `qeth` module. The Linux 2.6 kernel introduced a new device driver

¹ The OSA-Express 1000BASE-T, FENET, and token-ring cards support both non-QDIO and QDIO modes. The OSA-Express Gigabit Ethernet card and all OSA-Express2 cards only support QDIO mode.

configuration interface, so instructions for configuring devices in Linux are presented separately for the Linux 2.4 and Linux 2.6 kernels.

Linux 2.4 Kernel

The file `/etc/chandev.conf` is used to specify the parameters for the device driver. Here is an example `/etc/chandev.conf` configuration for an OSA-Express port running in QDIO mode:

```
noauto;qeth-1,0xc300,0xc301,0xc302,0,0;add_parms,0x10,0xc300,0xc302
,portname:OSA1
```

Table 4-1 describes the device parameters.

Table 4-1 Device driver parameters for qeth

Parameter	Description
noauto	Stops auto-detection of channel devices.
qeth-1,0xc300,0xc301,0xc302,0,0	
qeth-1	The device interface number. A value of “-1” indicates that the next available device number will be automatically allocated. For example, if we already had qeth0 and qeth1 devices defined to the channel device layer, the next device to be defined would be qeth2.
0xc300	The read subchannel address.
0xc301	The write subchannel address.
0xc302	The data subchannel address.
0	The number of kilobytes to be allocated for read and write buffers. 0 specifies the default value (8192 KB in QDIO mode).
0	The relative port number of the CHPID. OSA-Express devices use only port 0.
add_parms,0x10,0xc300,0xc301,portname:OSACHP03	
add_parms	Used to pass additional parameters to the driver.
0x10	Identifies the device as an OSA-Express CHPID in QDIO mode.
0xc300,0xc302	The desired device address range.
portname:OSA1	Identifies the port for sharing by other operating system images. See “OSA port names” on page 74 for more information about port names and if this parameter is required.

Linux 2.6 kernel

The device driver creates files in the /sysfs filesystem when it is loaded. The driver and the devices controlled by it are configured by writing values into these files. A command to activate an OSA-Express port running in QDIO mode is:

```
echo 0.0.c300,0.0.c301,0.0.c302 > /sys/bus/ccwgroup/drivers/qeth/group
```

The device is defined as part of the qeth group. The three device numbers are specified in device bus-ID form with 0.0 preceding each device number. The device numbers are the read, write, and data channels of the device. The qeth device driver uses the device bus-ID of the read subchannel and creates a directory for the device:

```
/sys/devices/qeth/0.0.c300
```

The directory contains several files that control attributes of the device, with one file per attribute. The device driver also creates other directories that are symbolic links to the device directory. In this example, the additional directories created are:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.c300  
/sys/bus/ccwgroup/devices/0.0.c300
```

The device driver automatically senses the type of the OSA port and sets all the device attributes to their default values. If any of the attributes need to be changed, the new value is written to the appropriate file. For example, if the port name needs to be specified, this command is used:

```
echo OSA1 > /sys/devices/qeth/0.0.c300/portname
```

Note: Most systems do not need a port name specified. See “OSA port names” on page 74 to see if your system requires one.

See the document *Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281* for a description of all the qeth driver attributes.

Once the desired attributes are set, the device is brought online by writing a 1 to the online file:

```
echo 1 >/sys/devices/qeth/0.0.c300/online
```

Setting a device online associates it with an interface name which is available in the if_name file and can be retrieved by reading the file:

```
cat /sys/devices/qeth/0.0.c300/if_name  
eth0
```

This is the name used to activate the device.

OSA port names

If a port name is required, it must be one to eight upper case characters and it must match the port name specified by all operating systems sharing the port. If it is not required, we advise omitting it. It is never required on z990, z890, or later systems. It is required on all S/390 G5 and G6 systems. For z800 and z900 processors, a port name is no longer required if you have driver 3G, EC stream J11204 MCL032 (OSA Level 3.33) installed (with the appropriate levels of z/VM and Linux). Refer to the Washington Systems Center Flash *OSA-Express MCL Enhancements - October 2003* for detailed information. The full text of the WSC Flash can be found at the following Web site:

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10250>

Linux configuration files for OSA

The Linux distributions discussed in this book include configuration files and scripts to automatically define and start network connections at boot time. This section describes those files for configuration of RHEL 4 and SLES 9.

Red Hat Enterprise Linux 4 configuration

The network configuration files are found in the `/etc/sysconfig/network-scripts/` directory. The configuration file is named `ifcfg-ethX`, where `X` is the interface number starting with 0 for the first interface. These configuration files have both network and hardware information in the same file. An example of a configuration file for the `eth0` interface is in Figure 4-7.

```
# IBM QETH
DEVICE=eth0
BOOTPROTO=static
BROADCAST=9.12.5.255
IPADDR=9.12.4.84
MTU=1492
NETMASK=255.255.254.0
NETTYPE=qeth
NETWORK=9.12.4.0
ONBOOT=yes
PORTNAME=OSA1
SUBCHANNELS=0.0.0300,0.0.0301,0.0.0302
TYPE=Ethernet
```

Figure 4-7 Example `ifcfg-eth0` file for RHEL 4

The variables `NETTYPE`, `PORTNAME`, and `SUBCHANNELS` are unique to zSeries and an OSA-Express adapter. The `NETTYPE` variable specifies the driver name, in this case the `qeth` driver. The `SUBCHANNELS` variable specifies the device addresses of the OSA adapter, with 0.0 preceding each four-digit

device address. The PORTNAME variable optionally specifies the port name for this interface. Most systems do not need a port name specified. See “OSA port names” on page 74 to see if your system requires one.

SUSE Linux Enterprise Server 9 configuration

Hardware configuration files are found in the /etc/sysconfig/hardware directory and network configuration files are found in /etc/sysconfig/network. The hardware configuration files are named using the driver name and device address. For example, an OSA port at address 300 has a hwcfg (hardware configuration) file named hwcfg-qeth-bus-ccw-0.0.0300. The contents of this file are shown in Figure 4-8.

```
#!/bin/sh
#
# hwcfg-qeth-bus-ccw-0.0.0300
#
# Hardware configuration for a qeth device at 0.0.0300
# Automatically generated by netsetup
#

STARTMODE="auto"
MODULE="qeth"
MODULE_OPTIONS=""
MODULE_UNLOAD="yes"

# Scripts to be called for the various events.
SCRIPTUP="hwup-ccw"
SCRIPTUP_ccw="hwup-ccw"
SCRIPTUP_ccwgroup="hwup-qeth"
SCRIPTDOWN="hwdown-ccw"

# CCW_CHAN_IDS sets the channel IDs for this device
# The first ID will be used as the group ID
CCW_CHAN_IDS="0.0.0300 0.0.0301 0.0.0302"

# CCW_CHAN_NUM set the number of channels for this device
# Always 3 for an qeth device
CCW_CHAN_NUM=3

# CCW_CHAN_MODE sets the port name for an OSA-Express device
CCW_CHAN_MODE="OSA1"
```

Figure 4-8 Example hwcfg-qeth-bus-ccw-0.0.0300 file

The comments in the file describe each variable's usage. Normally, only the device addresses and the optional port name need to be changed. Most systems do not need a port name specified. See "OSA port names" on page 74 to see if your system requires one.

Additional variables that are allowed in the `hwcfg` file are described in the sample file `/etc/sysconfig/hardware/skel/hwcfg-qeth`. For example, other options for the `qeth` driver can be specified in a variable named `QETH_OPTIONS`. If a new hardware configuration file is created for an existing adapter, activate it with the `hwup` command. For example, the command to activate the hardware configuration file shown in Example 4-8 is `hwup qeth-bus-ccw-0.0.0300`.

The network configuration files found in the `/etc/sysconfig/network` directory are also named using the driver name and device address. For example, the configuration file for the same OSA port at address 300 is named `ifcfg-qeth-bus-ccw-0.0.0300`. SUSE provides a sample file of the variable names and allowed values for a network configuration file in the `/etc/sysconfig/network/ifcfg.template` file. After the file has been customized with the correct network parameters, it is activated with the command `ifup qeth-bus-ccw-0.0.0300`.

4.2.3 Channel-to-channel adapter

Channel-to-channel (CTC) is a mainframe point-to-point connection technology, using real hardware channels. It has been in existence for a long time and is used to interconnect different physical servers, logical partitions, or both. Because all zSeries operating systems use the same link protocol, it is possible to connect a Linux server not only to another Linux, but also to a z/VM or z/OS TCP/IP stack. CTC support exists for a number of channel technologies including ESCON and FICON channels.

ESCON CTC connectivity

To connect two systems using ESCON, you must define two channels. On one side, the channel is defined as CHPID type CTC, on the other side, as CHPID type CNC. The ESCON CTC connection can either be point-to-point or switched point-to-point (that is, it can be connected to an ESCON director).

EMIF allows LPARs to share channel paths, and so optionally, they can share any control units and associated I/O devices configured to these shared channels. Sharing channel paths means that you can reduce the number of physical connections between processor complexes. Both CTC and CNC channels can be defined as shared channels; see Figure 4-9 on page 77.

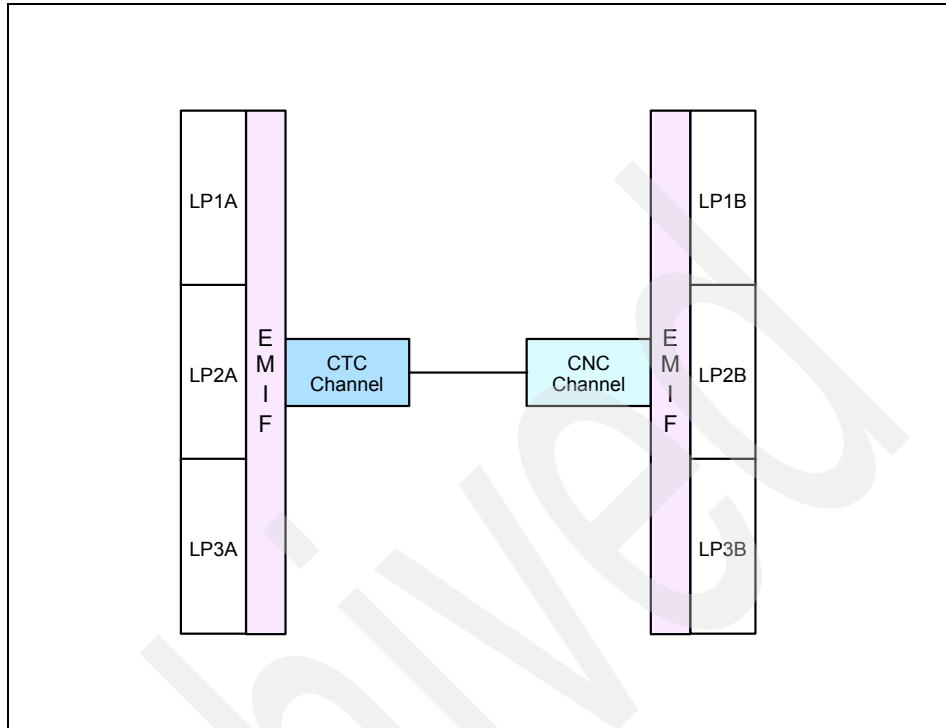


Figure 4-9 CTC connection with shared channels

FICON CTC connectivity

Channel-to-channel communication in a FICON environment is provided between two FICON (FC) channel FCTC control units. Each end of the connection requires two devices, a device for read I/O and a device for write I/O. The read device on one side needs to be connected to the write device on the other side and vice versa.

There are several differences between the ESCON and FICON CTC implementations, as shown in Table 4-2.

Table 4-2 ESCON and FICON CTC differences

Characteristic	ESCON	FICON
Number of required channels	At least 2	1 or 2
Channel dedicated to CTC function	Yes	No
Number of unit addresses supported	Up to 512	Up to 16384
Data transfer bandwidth	12-17 MBps	Up to 2 Gbps

Characteristic	ESCON	FICON
Number of concurrent I/O operations	1	Up to 32
Data transfer mode	Half duplex	Full duplex

The details of these differences are as follows:

- ▶ ESCON CTC connectivity is provided by a pair of ESCON channels, one defined as CTC and the other defined as CNC. At least two ESCON channels are required.
FICON CTC connectivity can be implemented using one or two FICON (FC) native channels.
- ▶ An ESCON channel defined as CTC can only support the CTC function. Only a control unit (type SCTC) can be defined on an ESCON CTC channel.
The FICON native (FC) channel supporting the FCTC control unit can communicate with an FCTC control unit on another machine, and simultaneously, the same FICON (FC) channel can also support operations to other I/O control unit types such as DASD and tape.
- ▶ An ESCON CTC channel supports a maximum of 512 unit addresses (devices).
A FICON native (FC) channel supports a maximum of 16,384 unit addresses (devices).
- ▶ An ESCON channel has a data transfer bandwidth of 12-17 MB, significantly less than the FICON or FICON Express channels.
- ▶ An ESCON channel supports only one actively communicating I/O operation at a time, while the FICON channel supports up to 32 concurrent I/O operations.
- ▶ An ESCON channel operates in half duplex mode, transferring data only in one direction at a time. A FICON channel operates in full duplex mode, sending and receiving concurrently.

For detailed information about FICON CTC implementation, refer to the Redpaper *FICON CTC Implementation*, REDP-0158.

Using a CTC with Linux

The CTC device driver in Linux supports both ESCON and FICON CTC devices as well as virtual CTC devices created by z/VM. It supports both network connections over these devices and TTY connections between Linux servers. Only network connections are described in this book. Refer to the appropriate level of the Linux on zSeries device drivers manual for your kernel level for information about configuring the `ctc` driver for TTY connections.

If your Linux distribution has the ctc driver compiled as a module, it must be loaded first. The device driver is a single module named ctc. Load the module with the **modprobe** command to ensure that any other required modules are loaded:

```
modprobe ctc
```

The configuration of the driver is different in the 2.4 and 2.6 kernels. We describe each level separately.

Linux 2.4 kernel

The ctc device is configured by the channel device layer using the file `/etc/chandev.conf`. An example `chandev.conf` entry for a ctc device follows:

```
ctc0,0x7c00,0x7c01,0,0,0,0
```

Table 4-3 describes the device parameters.

Table 4-3 Parameters for the ctc device driver

Parameter	Description
ctc0	The device interface name must be ctc followed by a number. The first device is ctc0.
0x7c00	The read subchannel address. It must be connected to the write subchannel address on the peer system.
0x7c01	The write subchannel address must be connected to the read subchannel address on the peer system.
0	This number represents the memory allocated for buffers. 0 specifies that the driver should decide on the allocation.
0	The CTC protocol number. The values are: <ul style="list-style-type: none"> 0 Compatibility mode (the default.) Not used with Linux, z/OS, or OS/390 peers. It is used for VM TCP/IP. 1 Extended mode used for Linux to Linux peers. 2 A CTC-based TTY, only for non-network Linux-Linux connections 3 Compatibility mode for OS/390 and z/OS peers. 4 A CTCMPC device, used by IBM Communications Server for Linux for CTC-based connections to VTAM
0	Specifies that checksumming is not done on received IP packets.
0	Hardware statistics are not gathered. This field is not supported by the ctc driver.

Linux 2.6 kernel

The device driver creates files in the `/sysfs` filesystem when it is loaded. The driver and the devices controlled by it are configured by writing values into these files. An example of a command to define a ctc device follows:

```
echo 0.0.7c00,0.0.7c01 > /sys/bus/ccwgroup/drivers/ctc/group
```

The two devices are defined as part of single ctc device. The two device numbers are specified in device bus-ID form with 0.0 preceding each device number. The first number is the read device and the second number is the write device. The ctc device driver uses the device bus-ID of the read subchannel and creates a directory for the device:

```
/sys/devices/cu3088/0.0.7c00
```

The directory contains several files that control attributes of the device, with one file per attribute. The device driver also creates other directories that are symbolic links to the device directory. In this example, the additional directories created are:

```
/sys/bus/ccwgroup/drivers/ctc/0.0.7c00  
/sys/bus/ccwgroup/devices/0.0.7c00
```

The default ctc protocol is 0, meaning a peer that is not Linux, z/OS, or OS/390. If this is incorrect, the protocol must be changed before the device is brought online by writing to the device's `protocol` file. For example, to specify that the peer system is another Linux server:

```
echo 1 >/sys/devices/cu3088/0.0.7c00/protocol
```

After the desired attributes are set, the device is brought online by writing a 1 to the `online` file:

```
echo 1 >/sys/devices/cu3088/0.0.7c00/online
```

Setting a device online associates it with an interface name which will be displayed on the Linux console. If access to the console is not available, you can use the **dmesg** command to discover the name. To confirm which device is used by a specific interface name, examine the symbolic link in the networking class directory. For example, to check the ctc0 interface name:

```
readlink /sys/class/net/ctc0/device ../../../../devices/cu3088/0.0.7c00
```

The ctc0 interface is associated with device 0.0.7c00.

Linux configuration files for CTC

The Linux distributions discussed in this book include configuration files and scripts to automatically define and start CTC network connections at boot time. This section describes those files for configuration of RHEL 4 and SLES 9.

Red Hat Enterprise Linux 4 configuration

The network configuration files are found in the `/etc/sysconfig/network-scripts/` directory. The configuration file is named `ifcfg-ctcX`, where `X` is the interface number starting with 0 for the first interface. These configuration files have both network and hardware information in the same file. An example of a configuration file for the `ctc0` interface is shown in Figure 4-10. Note that in this file, the variable `GATEWAY` defines the partner IP address on a point to point link.

```
# IBM CTC
DEVICE=ctc0
BOOTPROTO=static
IPADDR=9.12.4.84
GATEWAY=9.12.4.85
NETMASK=255.255.255.255
ONBOOT=yes
SUBCHANNELS=0.0.7c00,0.0.7c01
NETTYPE=ctc
CTCProt=0
TYPE=CTC
```

Figure 4-10 Example `ifcfg-ctc0` file for RHEL 4

SUSE Linux Enterprise Server 9 configuration

Hardware configuration files are found in the `/etc/sysconfig/hardware` directory and network configuration files are found in `/etc/sysconfig/network`. The hardware configuration files are named using the driver name and device address. For example, an CTC adapter pair at address `7C00` has a `hwcfg` (hardware configuration) file named `hwcfg-ctc-bus-ccw-0.0.7c00`. SUSE provides a sample configuration in the file `/etc/sysconfig/hardware/skel/hwcfg-ctc`. The comments in that file describe each variable. Use the `hwup` command to activate the adapter after customizing the file.

The network configuration files found in the `/etc/sysconfig/network` directory are also named using the driver name and device address. For example, the configuration file for the same CTC adapter at address `7C00` is named `ifcfg-ctc-bus-ccw-0.0.7c00`. SUSE provides a sample file of the variable names and allowed values for network configuration in the file `/etc/sysconfig/network/ifcfg.template`. Use the `ifup` command to activate the network connection after customizing the file.

Recommendations

We do not recommend using ESCON or FICON CTCs as networking connectivity options for your Linux on zSeries systems. For inter-LPAR communications, we recommend that you use HiperSockets or OSA-Express.

For communications inside a single z/VM LPAR, we recommend VSWITCH. Although CTC bandwidth is good (particularly FICON Express), CTC connectivity is less fault tolerant than other solutions. Sometimes, if one side of the link has a problem, one or even both of the systems have to be re-IPLed in order to restart the CTC link. For communications between the zSeries machine and other systems in the network, we recommend that you use any of the OSA-Express2 features, OSA-Express Gigabit Ethernet or OSA-Express 1000BASE-T.

4.2.4 Common Link Access to Workstation (CLAW)

Common Link Access to Workstation (CLAW) is a point-to-point protocol. A CLAW device is an ESCON channel-attached device that supports CLAW protocol. These devices can be used to connect your Linux for zSeries system to another system, for example, an RS/6000 or a Cisco Channel Interface Processor (CIP) card.

The CLAW device driver for Linux is not discussed in this book. For a detailed review of the driver and its parameters, refer to the appropriate level of the Linux on zSeries device drivers manual for your kernel level. The documentation can be found at the following Web site:

<http://www.ibm.com/developerworks/linux/linux390/>

The name of the documents for each kernel level at the time of the writing of this book are:

- ▶ Linux 2.4 kernel

Linux on zSeries, Device Drivers and Installation Commands, March 23, 2005, SC33-8282

- ▶ Linux 2.6 kernel

Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281

Recommendations

CLAW devices are old technology and are not as efficient or reliable as some other solutions. Instead, for communications between Linux and other systems in the network, we recommend that you use OSA-Express Gigabit or 1000BASE-T.

4.3 HiperSockets

HiperSockets provides very fast TCP/IP communications between servers running in different logical partitions (LPARs) on a zSeries machine. The z890 and z990 processors support up to 16 HiperSocket internal LANs. The z800 and z900 processors support up to four HiperSockets. Each HiperSocket is defined as a CHPID of type IQD.

To communicate between servers running in the same zSeries Central Electronics Complex (CEC), HiperSockets sets up I/O queues in the zSeries processor's memory. The packets are then transferred at memory speeds between the servers, thereby totally eliminating the I/O subsystem overhead and any external network latency.

HiperSockets implementation is based on the OSA-Express Queued Direct Input/Output (QDIO) protocol; therefore, HiperSockets is called internal QDIO (iQDIO). HiperSockets is implemented in microcode that emulates the Logical Link Control (LLC) layer of an OSA-Express QDIO interface.

So although HiperSockets is a type of virtualization technology, it relies on zSeries microcode to run. Therefore for the purposes of this book, we categorize it as a physical networking option.

Typically, before a packet can be transported on an external LAN, a LAN frame has to be built, and the MAC address of the destination host or router on that LAN has to be inserted into the frame. HiperSockets does not use LAN frames, destination hosts, or routers. TCP/IP stacks are addressed by inbound data queue addresses instead of MAC addresses. The zSeries server microcode maintains a lookup table of IP addresses for each HiperSocket. This table represents an internal LAN. At the time a TCP/IP stack starts a HiperSockets device, the device is registered in the IP address lookup table with its IP address and its input and output data queue pointers. If a TCP/IP device is stopped, the entry for this device is deleted from the IP address lookup table.

HiperSockets copies data synchronously from the output queue of the sending TCP/IP device to the input queue of the receiving TCP/IP device by using the memory bus to copy the data through an I/O instruction. The controlling operating system that performs I/O processing is identical to OSA-Express in QDIO mode. The data transfer time is similar to a cross-address space memory move, with hardware latency close to zero.

HiperSockets operations are executed on the processor where the I/O request is initiated by the operating system. HiperSockets starts write operations. The completion of a data move is indicated by the sending side to the receiving side with a Signal Adapter (SIGA) instruction. Optionally, the receiving side can use dispatcher polling instead of handling SIGA interrupts. The I/O processing is

performed without using the System Assist Processor (SAP). This new implementation is also called *thin interrupt*. HiperSockets does not contend with other system I/O activity and it does not use CPU cache resources; therefore, it has no association with other activity in the server.

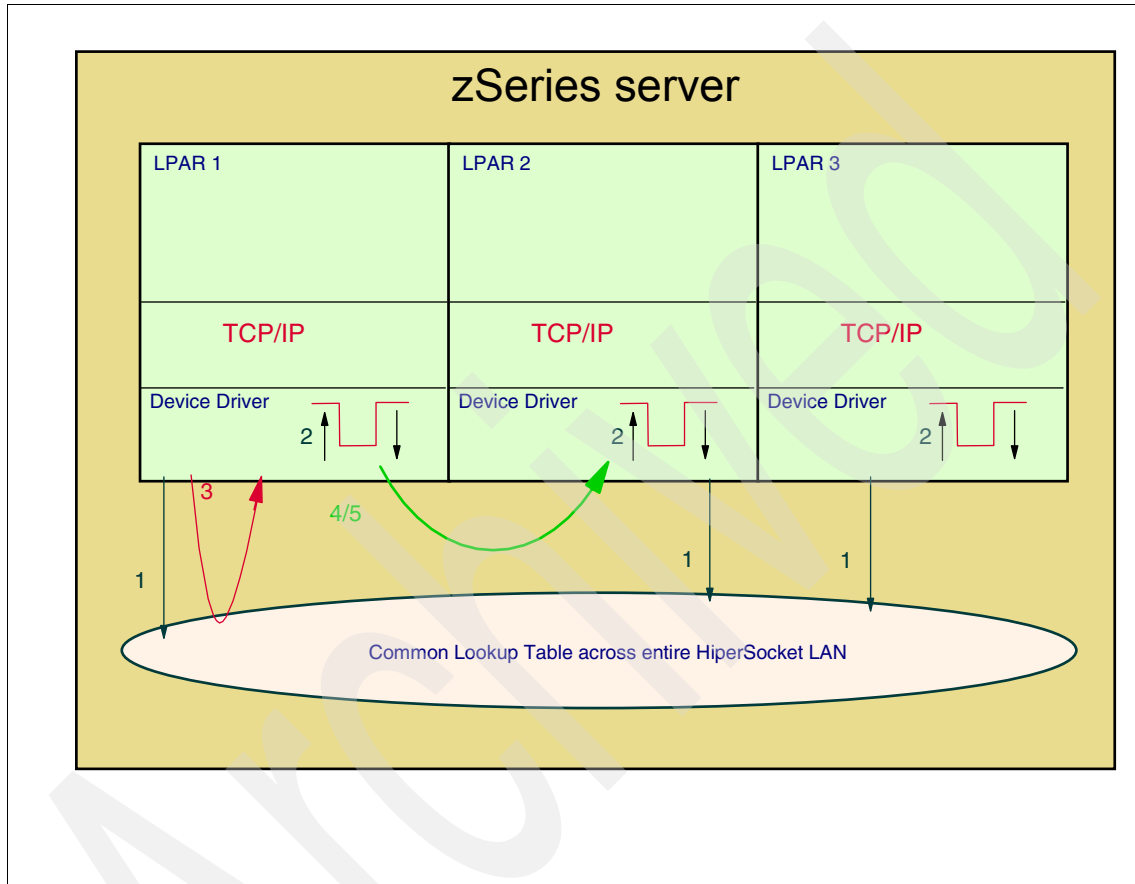


Figure 4-11 HiperSockets basic operation

The HiperSockets operational flow (Figure 4-11) consists of five steps:

1. Each TCP/IP stack (image) registers its IP addresses into HiperSockets' server-wide Common Address Lookup table. There is one lookup table for each HiperSockets LAN.
2. The address of the TCP/IP stack's receive buffers are appended to the HiperSockets queues.

3. While data is transferred, the send operation of HiperSockets performs a table lookup for the addresses of the sending and receiving TCP/IP stacks and their associated send and receive buffers.
4. The sending processor copies the data from its send buffers into the target processor's receive buffers (zSeries server memory).
5. The sending processor optionally delivers an interrupt to the target TCP/IP stack. This optional interrupt uses the thin interrupt support function of the zSeries server, which means the receiving host will look ahead, detecting and processing inbound data. This technique reduces the frequency of real I/O or external interrupts.

For a detailed review of HiperSockets, refer to the IBM Redbook *zSeries HiperSockets*, SG24-6816.

Hardware configuration

The IOCP statements in Figure 4-12 show two HiperSocket LANs. The HiperSocket LANs are being shared by multiple LPARs.

```

ID      MSG1='IODFA2',MSG2='SYS1.IODFA2 - 2004-08-04 03:26',      *
        SYSTEM=(2066,1),                                          *
        TOK=('ZAPHOD',000000011C8A2066032619730104217F00000000,0*
        0000000,'04-08-04','03:26:19','SYS1','IODFA2')
        RESOURCE PARTITION=((CF0A,E),(CF0B,F),(CF01,A),(CF02,B),(LINUX*
        1,9),(VM1,5),(VM2,6),(VM3,7),(VM4,8),(ZOSL,C),(ZOSS,D),(*
        ZOS1,1),(ZOS2,2),(ZOS3,3),(ZOS4,4))
        CHPID PATH=(FA),SHARED,                                    *
        PARTITION=((LINUX1,VM1,ZOS1,ZOS2,ZOS3),(LINUX1,VM1,VM2,V*
        M3,VM4,ZOSL,ZOSS,ZOS1,ZOS2,ZOS3,ZOS4)),TYPE=IQD
        CHPID PATH=(FB),SHARED,                                    *
        PARTITION=((LINUX1,VM1,ZOS1,ZOS2,ZOS3),(LINUX1,VM1,VM2,V*
        M3,VM4,ZOSL,ZOSS,ZOS1,ZOS2,ZOS3,ZOS4)),TYPE=IQD
        CNTLUNIT CUNUMBR=FA00,PATH=(FA),UNIT=IQD
        CNTLUNIT CUNUMBR=FB00,PATH=(FB),UNIT=IQD
        IODEVICE ADDRESS=(FA00,032),CUNUMBR=(FA00),UNIT=IQD
        IODEVICE ADDRESS=(FB00,032),CUNUMBR=(FB00),UNIT=IQD

```

Figure 4-12 HiperSockets IOCP statements

Figure 4-13 on page 86 provides a graphical representation of this configuration.

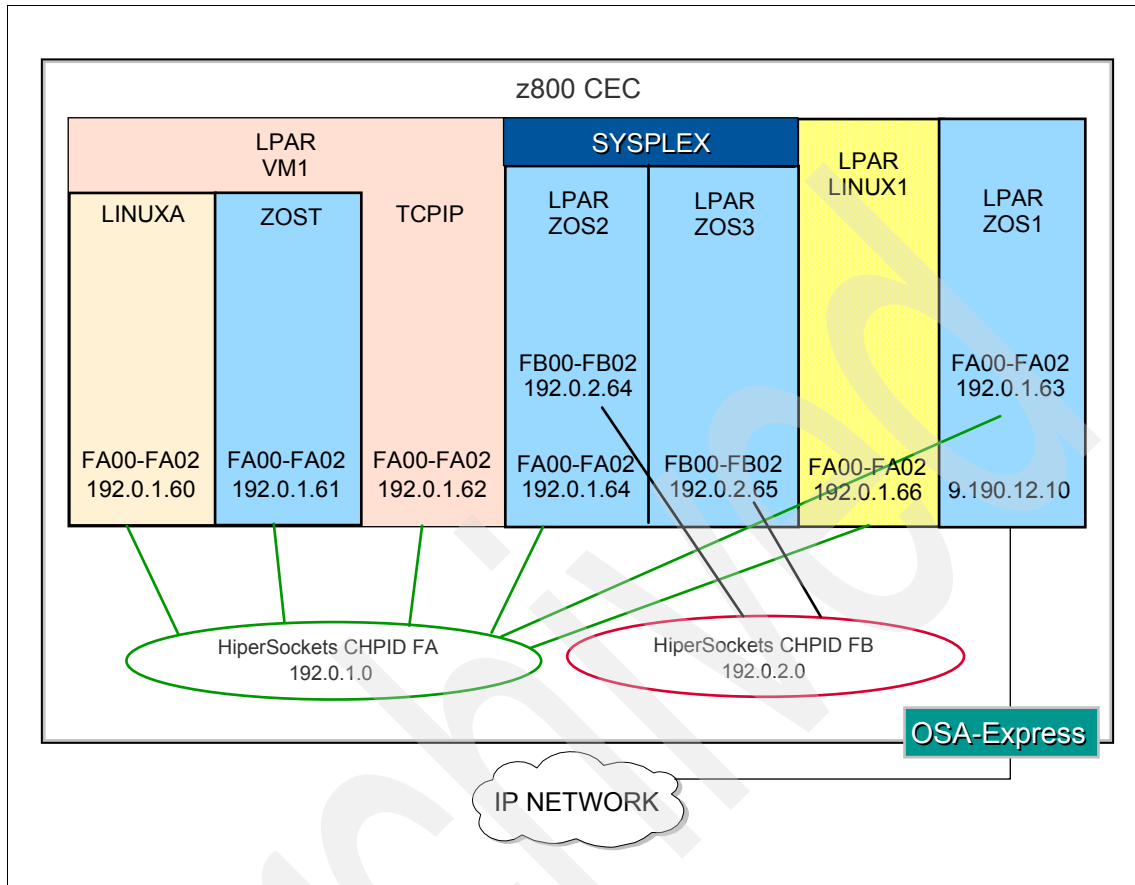


Figure 4-13 HiperSockets example environment

z/VM considerations

Each HiperSockets connection requires three I/O devices. One device is used for read control, one device is used for write control, and one device is used for data exchange. The device number for the control write device must be the device number for the read control device plus 1. The device number for the data exchange device can be any number.

We can use the VM DEDICATE command to reserve these devices for a particular Linux guest.

The syntax of the DEDICATE statement is:

```
DEDICATE virtual_address real_address
```

Figure 4-14 illustrates the DEDICATE statement usage in the directory entry for a Linux guest.

```
DEDICATE FA00 FA00
DEDICATE FA01 FA01
DEDICATE FA02 FA02
```

Figure 4-14 From a user directory of a HiperSockets interface dedicated to a Linux guest

Using HiperSockets with Linux

From a Linux on zSeries perspective, the HiperSockets interface looks a lot like an OSA-Express (QDIO mode) interface. Linux uses the qdio and qeth modules to exploit HiperSockets.

Linux 2.4 kernel

An example /etc/chandev.conf configuration for a HiperSockets interface follows:

```
noauto;qeth-1,0xfa00,0xfa01,0xfa02;add_parms,0x10,0xfa00,0xfa02
```

Table 4-4 provides a description of the parameters.

Table 4-4 qeth device driver parameters

Parameter	Description
noauto	Stops auto-detection of channel devices.
noauto;qeth-1,0xfa00,0xfa01,0xfa02,0,0	
qeth-1	The device interface number. A value of "-1" indicates that the next available device number will be automatically allocated. Even though we used "qeth" as the device interface type, the actual interface name will start with "hsi". So, for example, you would do an "ifconfig hsi0" to display the interface after the device drivers have been loaded.
0xfa00	The read subchannel address.
0xfa01	The write subchannel address.
0xfa02	The data subchannel address.
0	The number of kilobytes to be allocated for read and write buffers. 0 specifies the default value (8192 KB in QDIO mode).
0	The relative port number of the CHPID. HiperSockets devices use only port 0.
add_parms,0x10,0xfa00,0xfa02	

Parameter	Description
add_parms	Used to pass additional parameters to the driver.
0x10	Identifies the device as an OSA-Express CHPID in QDIO mode.
0xfa00,0xfa02	The desired device address range.

Note: We do not need to use a PORTNAME when using HiperSockets.

Linux 2.6 kernel

The device driver creates files in the /sysfs filesystem when it is loaded. The driver and the devices controlled by it are configured by writing values into these files. An example of a command to activate a HiperSockets device follows:

```
echo 0.0.fa00,0.0.fa01,0.0.fa02 > /sys/bus/ccwgroup/drivers/qeth/group
```

The device is defined as part of the qeth group. The three device numbers are specified in device bus-ID form with 0.0 preceding each device number. The device numbers are the read, write, and data channels of the device. The qeth device driver uses the device bus-ID of the read subchannel and creates a directory for the device:

```
/sys/devices/qeth/0.0.fa00
```

The directory contains several files that control attributes of the device, with one file per attribute. The device driver also creates other directories that are symbolic links to the device directory. In this example, the additional directories created are:

```
/sys/bus/ccwgroup/drivers/qeth/0.0.fa00
/sys/bus/ccwgroup/devices/0.0.fa00
```

The device driver automatically senses that the device is a HiperSockets device and sets all the device attributes to their default values. If any of the attributes need to be changed, the new value is written to the appropriate file. HiperSockets device usually do not need any attributes changed.

See the document *Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281* for a listing and description of all the attributes of a HiperSockets device and the qeth driver.

Once the desired attributes are set, the device is brought online by writing a 1 to the online file:

```
echo 1 >/sys/devices/qeth/0.0.fa00/online
```

Setting a device online associates it with an interface name which is available in the `if_name` file and can be retrieved by reading the file:

```
cat /sys/devices/qeth/0.0.fa00/if_name
hsi0
```

This is the name used to activate the device.

Linux configuration files for HiperSockets

The Linux distributions discussed in this book include configuration files and scripts to automatically define and start network connections at boot time. Because HiperSockets uses the same qeth device driver as OSA adapters, the configuration files are very similar to the files described in “Linux configuration files for OSA” on page 74. There are two differences to note for HiperSockets when reading that section:

1. The network interface name is `hsiX`, not `ethX`, where `X` is the interface number starting with 0 for the first interface.
2. A port name cannot be specified.

HiperSockets Network Concentrator

Traffic between HiperSockets and OSA-Express can be bridged transparently using the HiperSockets Network Concentrator. See Figure 4-15 for an example. This function allows network traffic to flow between the HiperSockets LAN and the physical LAN on the same subnet without requiring intervening network routing overhead, thus increasing performance and simplifying the network configuration. This is achieved by configuring a connector Linux system that has HiperSockets and OSA-Express connections defined. The HiperSockets Network Concentrator registers itself with HiperSockets as a special network entity to receive data packets destined for an IP address on the external LAN using an OSA-Express port. The HiperSockets Network Concentrator also registers IP addresses to the OSA-Express on behalf of the TCP/IP stacks using HiperSockets, hence providing inbound and outbound connectivity and making all systems appear to be on the same physical network.

HiperSockets Network Concentrator support is performed using the next-hop-IP-address in the Queued Direct Input/Output (QDIO) header, instead of using a Media Access Control (MAC) address. Therefore, VLANs in a switched Ethernet fabric are not supported by this support. TCP/IP stacks using only HiperSockets to communicate among each other with no external network connection see no difference, and the HiperSockets support and the networking characteristics are unchanged.

To exploit HiperSockets Network Concentrator, refer to the instructions detailed in the appropriate level of the Linux on zSeries device drivers manual for your kernel level. The documentation can be found at:

<http://www.ibm.com/developerworks/linux/linux390/>

The name of the documents for each kernel level at the time of the writing of this book are:

- ▶ Linux 2.4 kernel

Linux on zSeries, Device Drivers and Installation Commands, March 23, 2005, SC33-8282

- ▶ Linux 2.6 kernel

Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005, SC33-8281

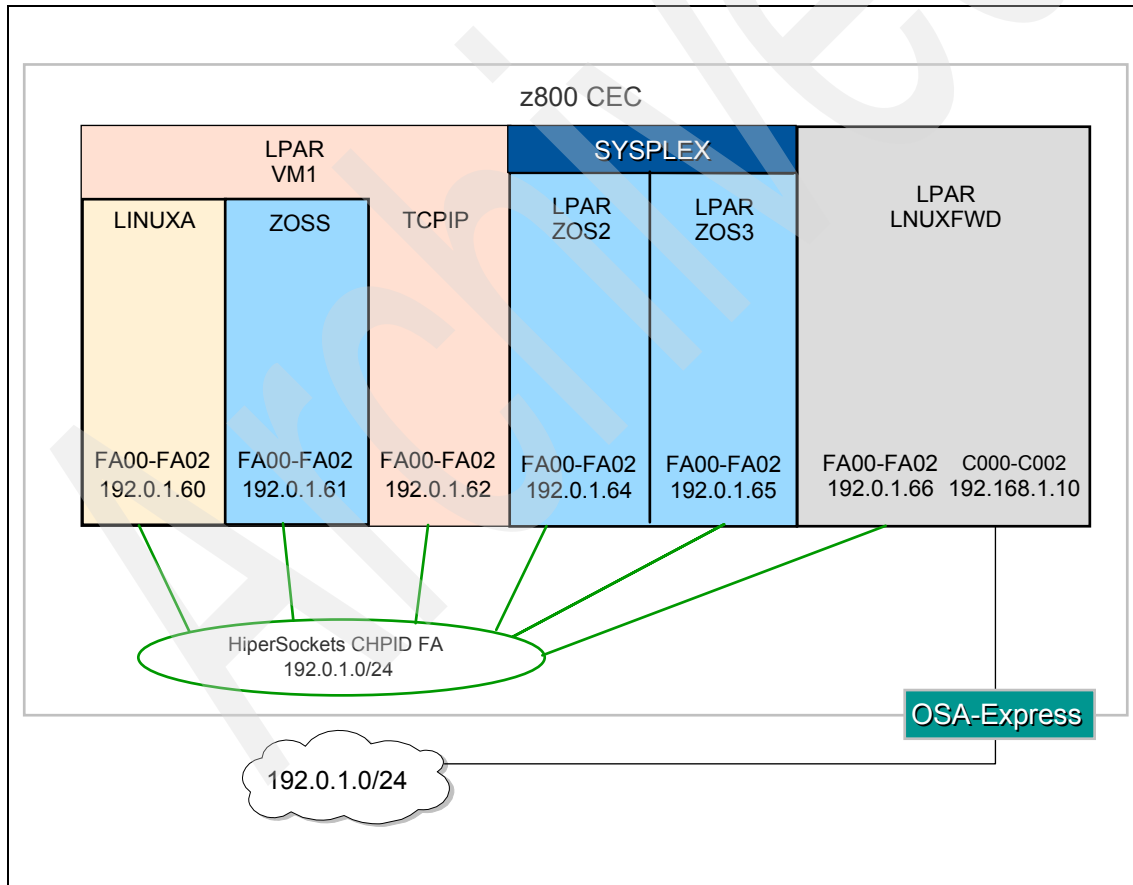


Figure 4-15 HiperSockets Network Concentrator example

Recommendations

HiperSockets is an excellent choice if you need to communicate across servers running in multiple LPARs in a single processor.

These servers can communicate at memory speeds, bypassing all the network overhead and delays. With HiperSockets, a maximum frame size can be defined according to the traffic characteristics for each of the HiperSockets LANs. By contrast, LANs such as Ethernet and token ring have a maximum frame size determined by their architecture.

Because there is no server-to-server traffic outside the zSeries CEC, a much higher level of network availability, security, simplicity, performance, and cost effectiveness is achieved as compared with servers communicating across a LAN. For example, because HiperSockets has no external components, it provides a very secure connection. For security purposes, servers can be connected to different HiperSockets LANs. All security features, such as firewall filtering, are available for HiperSockets interfaces as they are with other TCP/IP network interfaces. HiperSockets looks like any other TCP/IP interface. Therefore, HiperSockets is transparent to applications and supported operating systems.

If you need to communicate across different LPARs within a zSeries processor, we recommend that you use HiperSockets.

4.4 Virtualization technology

In a z/VM environment, the Linux operating system runs inside a virtual machine (*VM Guest*). The Linux server can still use physical interfaces, such as an OSA-Express card, but VM also provides virtual interfaces that the Linux guest can use. These interfaces are a VM control program simulation of a physical interface. They can be broadly split into two categories, point-to-point connectivity and VM simulated LAN technology.

4.5 Point-to-point connectivity

Prior to z/VM V4.2, the virtual connectivity options for connecting one or more virtual machines were limited to virtual channel-to-channel adapters (CTCA) and the Inter-User Communications Vehicle (IUCV) facility. These virtual interfaces are classified as point-to-point connections.

Using point-to-point connectivity, the z/VM TCP/IP stack or a Linux guest has to act as a router between the external network (using a physical interface) and the

Linux guests (which are using virtual interfaces.) This means that additional routing definitions are required in the network so that other machines know that, in order to communicate with the Linux guests, they must go through the z/VM TCP/IP or Linux router. Although the bandwidth of point-to-point connections is considerable and thus affords the rapid movement of large amounts of data between guests, these interfaces have a number of drawbacks.

Using CTCA links as an example, in order for a Linux guest to communicate with the external network, you must define CTCA device pairs on both the Linux and VM TCP/IP side. If you also have a requirement for individual Linux guests to communicate with each other, you also need to configure additional CTCA devices for those links (see the connections between Linux1 and Linux 2 in Figure 4-16 on page 92). CTCA devices on both sides of the connection then need to be coupled together. Static routing statements must be defined in both the Linux and VM TCP/IP stacks. Finally, if one side of the point-to-point connection goes down, it is often difficult to subsequently reconnect the two guests. Frequently, one of the Linux guest machines has to be rebooted in order to reestablish the connection.

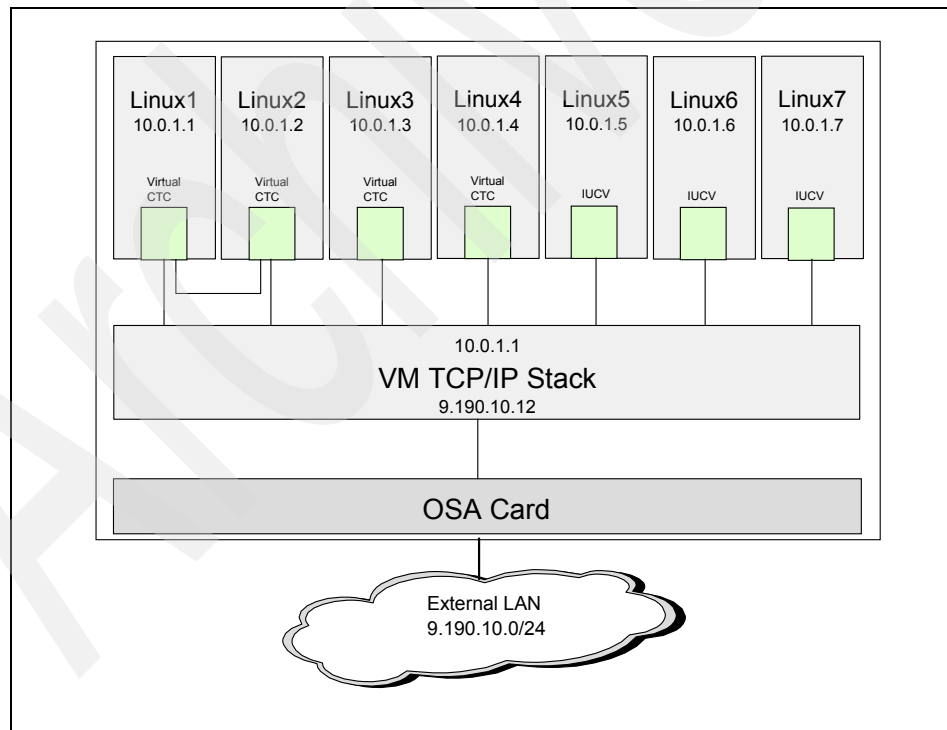


Figure 4-16 VM point-to-point connections

Recommendations

Given the complexity of managing a point-to-point network within a z/VM system, we do not recommend it as a solution. When it was the only topology we had available to communicate among guests, obviously, it was acceptable. We now recommend that you use one of the z/VM LAN technologies, Guest LAN or VSWITCH.

4.5.1 Guest LAN

From z/VM V4.2 and later, the z/VM control program (CP) has been enhanced to provide a feature known as *Guest LAN*. This feature enables you to create multiple Virtual LAN segments within a z/VM environment. As can be seen from Figure 4-17 on page 94, Guest LANs do not have a physical connection to the external network. Instead, they must use a router (z/VM TCP/IP or Linux) in the same fashion as was required for point-to-point topology. The Linux router, z/VM TCP/IP stack, must have an external interface, such as an OSA-Express card, and an interface connected to the Guest LAN.

Note: Although the structures and simulated devices related to the Guest LAN under z/VM are virtual, we use the term *Guest LAN* and not *Virtual LAN*, because the term Virtual LAN (VLAN) has a different meaning in the networking world.

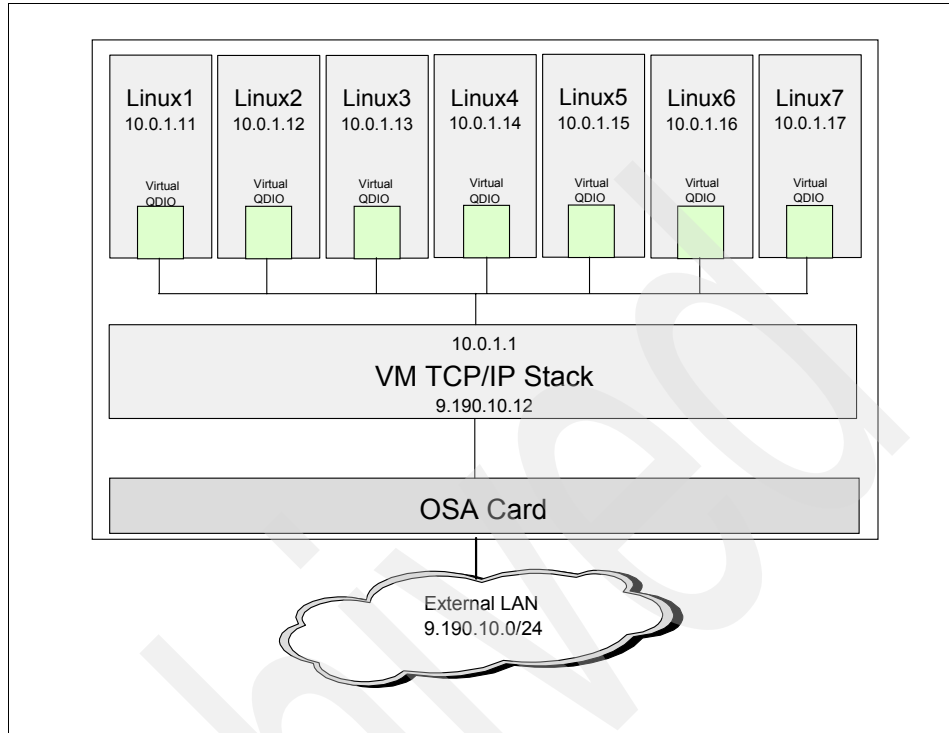


Figure 4-17 z/VM Guest LAN

There is no architectural limit on the number of Guest LAN segments that can be created. The actual limit is governed by available machine resources, but this is more of a theoretical limit than a practical one. The number of Guest LANs that can be created can be considered unlimited for all practical purposes.

In contrast to point-to-point connections, to connect to a Guest LAN, individual guest machines create a virtual Network Interface Card (NIC). They can connect this NIC to the Guest LAN and communicate with other guests using standard TCP/IP protocols. Defining links to individual guests and coding static routes for each link are no longer required, because the guests exist in the same Guest LAN segment.

Note: A z/VM Guest LAN is contained solely within a z/VM LPAR. A Linux guest connected to that Guest LAN cannot directly communicate (using its Guest LAN interface) with hosts outside of the Guest LAN.

Guest LANs versus Dedicated Hardware

When do you choose Guest LANs versus dedicated hardware connections? Here are some guidelines to consider:

- ▶ Dedicated hardware connections can be best for some environments:
 - When intense network activity is expected.
 - When direct external connectivity is required.
- ▶ Guest LANs may be best for other environments:
 - When network hardware is limited.
 - When multiple nodes are guests in the same z/VM host image.
 - When network activity must be isolated from primary network (for instance, test environments, student labs, or application server access to database servers.)
 - When system storage below the 2 GB line is constrained. Each dedicated interface might require up to 132 locked memory pages.
 - When most communication is among guests.

As z/VM Guest LAN is a virtualization technology, it is not limited to the use of zSeries hardware. Guest LAN is supported on all IBM mainframe hardware from IBM 9672 Generation 5 processors onward, including the Multiprise 3000.

When Guest LAN was first released with z/VM V4.2, the virtual NIC simulated a HiperSockets device, as introduced on the zSeries 900.

From z/VM V4.3, the Guest LAN could be defined to use either simulated HiperSockets (iQDIO, or internal QDIO) devices or simulated QDIO devices. The QDIO device that is simulated is the OSA-Express Ethernet adapter. All Guest LAN examples in this book use the QDIO Guest LAN and not the HiperSockets Guest LAN.

Note: Even though QDIO and HiperSockets Guest LANs are different, when it comes to defining the virtual Network Interface Card to the Linux guest, they both use the same qdio and qeth device drivers. However, QDIO Guest LANs use a device identifier of ethx, and HiperSockets Guest LANs use a device identifier of hsix.

QDIO versus iQDIO

When deciding whether to deploy QDIO Guest LAN or HiperSockets (iQDIO) Guest LAN, consider the following information:

- ▶ QDIO (OSA-Express simulation)
 - IPv4 and IPv6 support
 - Easy to migrate from QDIO Guest LAN to VSWITCH
 - Ethernet transport
 - Asynchronous
 - Can be used as an OSA-Express test network
- ▶ iQDIO (HiperSockets simulation)
 - IPv4 support
 - Supports multicast router connections
 - Deploy MTUs larger than 8 K
 - Synchronous
 - Can be used as a HiperSockets test network
 - Slightly smaller path length in CP than QDIO Guest LAN

4.6 Guest LAN configuration

To create a Guest LAN, use the following required steps:

1. Create a z/VM Guest LAN segment in the z/VM host system.
2. Create a virtual Network Interface Card (NIC) in each guest machine that will be connected to the Guest LAN.
3. Connect the virtual NIC in each guest machine to the Guest LAN.
4. After the Linux guest has been booted, configure the appropriate device drivers in that guest to connect to the Guest LAN.

Important: Although the syntax in the following sections is valid, always refer to the relevant level of z/VM reference manual for a complete description of all command syntax.

For the following examples, we used *z/VM Version 5 Release 1.0 CP Planning and Administration*, SC24-6083, and *z/VM Version 5 Release 1.0 CP Commands and Utility Reference*, SC24-6081.

4.6.1 Create a z/VM Guest LAN

z/VM Guest LANs can be created in one of two ways: either by a `DEFINE LAN` statement in the z/VM CP SYSTEM CONFIG file or by using the `DEFINE LAN CP` command. Why choose one over the other? Guest LANs created with the `DEFINE LAN` command are only valid for the life of a z/VM system. In other words, if that system is shut down and then IPLed, the Guest LAN is no longer be defined. We recommend that for production Guest LANs you make a permanent entry in the

CP SYSTEM CONFIG file. This is known as a *persistent* Guest LAN. For testing purposes, the **DEFINE LAN** command is perfectly valid because it provides the flexibility to dynamically create Guest LANs as required.

4.6.2 Persistent Guest LANs

In order to define a persistent Guest LAN, we need to add a DEFINE LAN statement to the CP SYSTEM CONFIG file. For a complete discussion of making changes to this file, refer to the relevant level of the z/VM reference manual.

The syntax of the DEFINE LAN statement is as follows:

```
DEFINE LAN lanname [ operands ]
```

Where the following is true:

lanname Is a 1-8 alphanumeric name of the z/VM Guest LAN.

operands Defines the characteristics of the z/VM Guest LAN.

Table 4-5 summarizes the operands accepted by the DEFINE LAN statement.

Table 4-5 Operands of the DEFINE LAN statement

Operand	Description
OWNERid <i>ownerid</i>	Establishes the owner of the LAN. The OWNERid is either a user id or SYSTEM. In our examples, the ownerid will be SYSTEM.
TYPE <i>lantype</i>	Specifies the type of LAN. Valid types are HIPERsockets for simulated HiperSockets adapters or QDIO for simulated QDIO adapters. HiperSockets is the default.
IP ETHernet	For QDIO Guest LANs, this indicates whether the transport for the LAN is Ethernet or IP. An Ethernet LAN operates at the Layer 2 level of the OSI model. An IP LAN operates at Layer 3 of the OSI model.
MAXCONN <i>maxconn</i>	Sets the maximum number of simultaneous adapter connections permitted. When MAXCONN is specified as INFinite, there is no limit on the number of connections. Any other value limits the number of simultaneous connections to a decimal value in the range of 1-1024.

Operand	Description
MFS <i>size</i>	Sets the Maximum Frame Size (MFS) for adapters on this network. When an adapter is connected to this LAN, it will adopt the network MFS. The MFS value determines the amount of storage to be allocated for internal structures and limits the effective Maximum Transfer Unit (MTU) size for the coupled adapters. The MFS operand is not valid for the QDIO Guest LAN; however, the effective MFS is 8 K for a QDIO adapter.
UNRESTRicted	Defines a LAN with no access control; therefore, any user can connect to the LAN. When neither UNRESTRicted nor RESTRicted are specified, UNRESTRicted is the default value.
RESTRicted	Defines a LAN with an access list to restrict connections. The LAN owner will use the SET LAN command to grant or revoke access to specific VM users (by user ID). The COUPLE command will only allow authorized users (those on the access list) to connect a simulated adapter to a RESTRICTED network.
ACCOUNTING <i>value</i>	Allows a Class B user to control whether accounting records are created for the LAN being defined.
GRANT <i>userlist</i>	Defines the list of users to be included in the Initial Access List of a RESTRICTED LAN. If the GRANT operand is omitted, the default is to GRANT the LAN owner.

As an example, to create a QDIO type Guest LAN named TSTLAN owned by SYSTEM, use:

```
DEFINE LAN TSTLAN OWNERID SYSTEM TYPE QDIO
```

4.6.3 The VMLAN statement

In addition to the DEFINE LAN statement, we can also add VMLAN statements to the CP SYSTEM CONFIG file to establish system-wide attributes for all z/VM Guest LANs that have been defined to the z/VM operating system. This includes Virtual Switches, which is discussed later.

Each VMLAN statement specifies a different system wide attribute and operands. Each attribute and operands are described separately.

VMLAN LIMIT [*operands*] See Table 4-6 on page 99.

VMLAN ACCOUNTing [*operands*] See Table 4-7 on page 99

VMLAN MACPREFIX *macprefix* See Table 4-8 on page 99

VMLAN MACIDRange [*operands*] See Table 4-9 on page 100

Table 4-6 Operands of the VMLAN LIMIT statement

Operands	Description
PERSistent INFinite <i>maxcount</i>	INFinite means that there will be an infinite number of PERSISTENT z/VM Guest LAN segments and Virtual Switches allowed on the system. INFinite is the default. Use the <i>maxcount</i> parameter to define a number of PERSISTENT Guest LANS (between 0 and 1024) permitted to run on the system.
TRANSient INFinite <i>maxcount</i>	INFinite means that there will be an infinite number of TRANSIENT z/VM Guest LAN segments allowed on the system. INFinite is the default. Use the <i>maxcount</i> parameter to define a number of TRANSIENT Guest LANS (between 0 and 1024) permitted to run on the system.

Table 4-7 Operands of the VMLAN ACCOUNTing statement

Operands	Description
SYSTEM ON OFF	Set the default accounting state for z/VM Guest LAN segments and Virtual Switches owned by the SYSTEM user ID. The default state of this attribute is OFF.
USER ON OFF	Set the default accounting state for z/VM Guest LAN segments owned by individual users. The default state of this attribute is OFF.

Table 4-8 Operands of the VMLAN MACPREFIX statement

Operands	Description
<i>macprefix</i>	Specifies the three-byte prefix (manufacturer ID) used when generating locally administered MAC addresses on the system. It must be six hexadecimal digits within the range of 020000 through 02FFFF (inclusive). In combination with the MAC ID used on the NICDEF directory statement, the MACPREFIX allows unique identification of virtual adapters within a network. If MACPREFIX is not specified, the default is 020000 02-00-00).

Table 4-9 Operands of the VMLAN MACIDRange statement

Operands	Description
SYSTEM xxxxxx-xxxxxx	The range of identifiers (up to six hexadecimal digits each) to be used by CP when generating the unique identifier part (last six hexadecimal digits) of a virtual adapter MAC address. If a SYSTEM MACIDRANGE is not specified, CP creates unique identifiers in any range (000001-FFFFFF). This operand is required.
USER xxxxxx-xxxxxx	USER xxxxxx-xxxxxx is the subset of the SYSTEM range of identifiers that are reserved for user definition of MACIDs in the NICDEF directory statement. When specified, CP does not assign MACIDs within this USER range during creation of virtual adapters defined dynamically (DEFINE NIC) or with the NICDEF (or SPECIAL) directory statement without the MACID operand. In these cases, CP generates a unique identifier for the adapter outside of the USER range. Any MACID values specified on a NICDEF directory statement must be within the USER range, or the virtual adapter is not defined during LOGON processing. If a USER MACIDRANGE is not specified, CP creates unique identifiers within the SYSTEM MACIDRANGE. This operand is optional.

4.6.4 Create a virtual Network Interface Card

You must create a virtual Network Interface Card (NIC) for each guest machine. After it is defined, this NIC can be connected to the Guest LAN. To the guest operating system, the NIC devices look like a range of OSA devices. The NIC can be defined permanently through a User Directory statement or temporarily (for the life of the Guest's session) through a CP command.

4.6.5 NIC definition in the user directory

To create a virtual Network Interface Card that will remain permanently defined to a VM guest machine (that is, across guest sessions and across IPLs of the z/VM operating system), use the NICDEF statement in the z/VM User Directory. The NICDEF statement defines virtual devices that are fully simulated by CP. The NIC automatically joins the Guest LAN when the z/VM user ID is logged on. The syntax of the NICDEF statement for Network Interface Cards is as follows:

```
NICDEF vdev [ operands ]
```

Where the following is true:

vdev Specifies the base virtual device address for the adapter.

operands Defines the characteristics of the virtual NIC.

Table 4-10 lists the operands accepted by the NICEF command.

Table 4-10 Operands for the NICDEF user directory statement

Operands	Description
TYPE HIPERs or TYPE QDIO	HIPERs indicates that a simulated HiperSockets adapter should be created. QDIO indicates that a simulated QDIO adapter should be created. If a LAN is identified in this statement or another with the same <i>vdev</i> , the NIC is automatically coupled to the specified <i>ownerid lanname</i> .
DEVices <i>devs</i>	The number (decimal) of virtual I/O devices to be created for a simulated NIC. If <i>devs</i> is omitted, the default number of devices is three.
LAN <i>ownerid lanname</i> or LAN SYSTEM <i>lanname</i>	Identifies a Guest LAN segment or Virtual Switch for an immediate connection to the NIC. If <i>ownerid</i> and <i>lanname</i> are omitted, the simulated adapter is left in the uncoupled state. When <i>ownerid</i> and <i>lanname</i> are specified, the adapter is automatically connected to the designated Guest LAN. Note that the <i>ownerid</i> can be specified as a name or using an asterisk (*) to represent the user ID of the current virtual machine. An <i>ownerid</i> of SYSTEM is used for a system owned Guest LAN or a Virtual Switch.
CHPID <i>xx</i>	A two-digit hexadecimal number that represents the CHPID number to be allocated in the virtual machine I/O configuration for this adapter. If CHPID is omitted, an available CHPID is automatically assigned to this adapter. This option is required when a HiperSockets adapter is being created for a z/OS guest, because z/OS configurations require a predictable CHPID number. During LOGON, CP attempts to use the specified CHPID number. If the specified CHPID number is already in use, this adapter is not defined. To correct this situation, you must eliminate the conflicting device or select a different CHPID.
MACID <i>xxxxxx</i>	Unique identifier (up to six hexadecimal digits) used as part of the adapter MAC address. During LOGON, your MACID (3 bytes) is appended to the system MACPREFIX (3 bytes) to form a unique MAC address for this adapter. If MACID is omitted from this definition, CP generates a unique identifier for this adapter. If the specified MACID is already in use, this adapter is not defined. To correct this situation, you must eliminate the conflicting device or select a different MACID.

Figure 4-18 shows an example CP User Directory entry for a Linux guest that connects to a QDIO Guest LAN.

```

USER LNX23 LNX23 128M 1G G
  INCLUDE IBMDFLT
  IPL CMSPARM AUTOOCR
  MACHINE XA
  CONSOLE 0009 3215
  NICDEF 0700 TYPE QDIO DEV 3 SYSTEM TSTLAN
  MDISK 0191 3390 3274 025 LEVW01 MR
  MDISK 0201 3390 3339 0200 LX3EA3 M
  MDISK 0202 3390 3539 3138 LX3EA3 M
  
```

Figure 4-18 User directory entry for a Linux guest: Connecting to a QDIO Guest LAN

4.6.6 NIC definition using CP commands

To create a virtual Network Interface Card that will only last for the life of a guest (that is, it will need to be redefined when the guest next logs on to the system), use the following command syntax:

```
DEFINE NIC vdev [ operands ]
```

Where the following is true:

- vdev** Specifies the base virtual device address for the adapter.
- operands** Define the characteristics of the virtual NIC.

Table 4-11 lists the operands accepted by the DEFINE NIC command.

Table 4-11 Operands for the DEFINE NIC command

Operands	Description
TYPE HIPERsockets	Defines this adapter as a simulated HiperSockets NIC. This adapter will function like the HiperSockets internal adapter. A HiperSockets NIC can function without a z/VM Guest LAN connection, or it can be coupled to a HiperSockets Guest LAN.
TYPE QDIO	Defines this adapter as a simulated QDIO NIC. This adapter will function like the OSA-Express (QDIO) adapter. A QDIO NIC is only functional when it is coupled to a QDIO Guest LAN or a Virtual Switch.

Operands	Description
DEVices <i>devs</i>	Determines the number of virtual devices associated with this adapter. For a simulated HiperSockets adapter, <i>devs</i> must be a decimal value between 3 and 3072 (inclusive). For a simulated QDIO adapter, <i>devs</i> must be a decimal value between 3 and 240 (inclusive). The DEFINE NIC command will create a range of virtual devices from <i>vdev</i> to <i>vdev</i> + <i>devs</i> -1 to represent this adapter in your virtual machine. The default value is 3.
CHPID <i>nn</i>	A two-digit hexadecimal number that represents the CHPID number the invoker wants to allocate for this simulated adapter. If the requested CHPID number is available, all of the virtual devices belonging to this adapter will share the same CHPID number. This option is only useful if you need to configure a virtual environment with predictable CHPID numbers for your simulated devices.

4.6.7 Connect the virtual NIC to the Guest LAN

Now that we have defined the virtual NIC, just as in a real network we need to connect that device to the LAN. If we had used the NICDEF User Directory statement to define our NIC, the guest machine would automatically connect to the LAN whenever it logged on. However, if we chose to use the DEFINE NIC command, we have an additional step to perform before the device is connected to the Guest LAN.

Use the COUPLE CP command to attach the virtual NIC to a compatible Guest LAN. The syntax of the COUPLE command for this scenario is:

```
COUPLE vdev TO [ operands ]
```

Where the following is true:

- vdev** Specifies the base virtual device address for the adapter.
- operands** Defines where to connect the NIC.

Table 4-12 lists the operands accepted by the COUPLE command for the purpose of connecting a virtual NIC to a Guest LAN.

Table 4-12 Operands for the COUPLE command

Operands	Description
<i>vdev</i>	The base address of the network adapter.

Operands	Description
<i>ownerid lanname</i>	The <i>ownerid</i> is the name of the owner of the Guest LAN (for example, SYSTEM). The <i>lanname</i> is the name of the Guest LAN or Virtual Switch.

Remember that a virtual NIC can only be coupled to a *compatible* Guest LAN. For example, a QDIO NIC cannot be coupled to a Guest LAN of type “HIPERsockets.”

Tip: If you choose to use the DEFINE NIC and COUPLE approach instead of the NICDEF User Directory statement, consider adding these two commands into your guest’s PROFILE EXEC file so that they are automatically executed whenever the guest logs on.

4.6.8 Example of building a z/VM Guest LAN

We now demonstrate how to build a z/VM Guest LAN in Figure 4-19.

```

DEFINE LAN TSTLAN OWNERID SYSTEM TYPE QDIO 1
LAN SYSTEM TSTLAN is created
Ready;

DEFINE NIC 0700 QDIO 2
NIC 0700 is created; devices 0700-0702 defined
Ready;

COUPLE 0700 TO SYSTEM TSTLAN 3
NIC 0700 is connected to LAN SYSTEM TSTLAN
Ready;

```

Figure 4-19 Steps to build a z/VM Guest LAN

To build a z/VM Guest LAN, follow these steps:

1. Define a QDIO Guest LAN owned by SYSTEM. This command was run from the MAINT user.
2. Define a Network Interface Card (NIC) of type QDIO. This command was run from Linux guest user LNX23.
3. Couple the NIC to the Guest LAN. This command was also run from LNX23.

Now that we have built a Guest LAN, we can use the CP QUERY LAN command to verify the status of the LAN, as shown in Figure 4-20 on page 105.

QUERY LAN TSTLAN ACTIVE

```
LAN SYSTEM TSTLAN      Type: QDIO      Active: 1      MAXCONN: INFINITE
  PERSISTENT UNRESTRICTED MFS: 8192    ACCOUNTING: OFF
Ready;
```

Figure 4-20 CP QUERY LAN command

In order to display information about the virtual NIC that we have defined, we can use the QUERY NIC CP command, as shown in Figure 4-21. If we use the DETAILS parameter of this command, we can get additional information about the IP addresses bound to this NIC and the amount of data that has been transmitted and received through this interface (TX packets/bytes and RX packets/bytes, respectively).

Notice that there is no IP addressing information and the number of bytes transmitted and received are both zero. Also, the port name value is set to UNASSIGNED. This tells us that the Linux guest has not started using this device for TCP/IP communications.

QUERY NIC

```
Adapter 0700 Type: QDIO      Name: UNASSIGNED Devices: 3
  Port 0 MAC: 02-00-00-00-00-06 LAN: SYSTEM TSTLAN      MFS: 8192
```

Q NIC DETAILS

```
Adapter 0700 Type: QDIO      Name: UNASSIGNED Devices: 3
  Port 0 MAC: 02-00-00-00-00-06 LAN: SYSTEM TSTLAN      MFS: 8192
RX Packets: 0      Discarded: 0      Errors: 0
TX Packets: 0      Discarded: 0      Errors: 0
RX Bytes: 0      TX Bytes: 0
Unassigned Devices:
  Device: 0700 Unit: 000 Role: Unassigned
  Device: 0701 Unit: 001 Role: Unassigned
  Device: 0702 Unit: 002 Role: Unassigned
```

Figure 4-21 CP QUERY NIC commands

Finally, we said that the virtual NIC simulates an OSA-Express QDIO device. This is confirmed by using the CP command QUERY VIRTUAL OSA from the guest machine, as shown in Figure 4-22 on page 106.

```
Q VIRTUAL OSA
OSA 0700 ON NIC 0700 UNIT 000 SUBCHANNEL = 0010
      0700 QDIO-ELIGIBLE      QIOASSIST NOT AVAILABLE
OSA 0701 ON NIC 0700 UNIT 001 SUBCHANNEL = 0011
      0701 QDIO-ELIGIBLE      QIOASSIST NOT AVAILABLE
OSA 0702 ON NIC 0700 UNIT 002 SUBCHANNEL = 0012
      0702 QDIO-ELIGIBLE      QIOASSIST NOT AVAILABLE
```

Figure 4-22 CP QUERY VIRTUAL OSA command

4.6.9 Undoing the definitions

Before moving on to describe how to connect a Linux guest to the Guest LAN, we review how to undo the previous definitions in an orderly fashion. This is only for completeness, and you should not follow these steps unless you no longer want to use the z/VM Guest LAN that you created.

Disconnect from a Guest LAN

Use the CP **UNCOUPLE** command to disconnect a virtual NIC from a Guest LAN segment. Figure 4-23 illustrates this command.

```
UNCOUPLE 700
NIC 0700 is disconnected from LAN SYSTEM TSTLAN
Ready;
```

Figure 4-23 UNCOUPLE command

Remove the virtual NIC from the guest machine

To remove a virtual NIC from a guest machine, use the CP **DETACH NIC** command. The command disconnects the virtual adapter from the Guest LAN (assuming the UNCOUPLE command has not been invoked) and removes each virtual device that has been created. Figure 4-24 illustrates the DETACH NIC command.

```
DETACH NIC 0700
NIC 0700 is destroyed; devices 0700-0702 detached
Ready;
```

Figure 4-24 DETACH NIC command

Remove the Guest LAN

To remove a Guest LAN from the system, use the **CP DETACH LAN** command. This command removes the LAN from the System LAN table, disconnects any virtual adapters that were using the LAN, and releases system resources associated with the LAN. Figure 4-25 illustrates the **DETACH LAN** command.

```
DETACH LAN TSTLAN OWNER SYSTEM
LAN SYSTEM TSTLAN is destroyed
Ready;
```

Figure 4-25 *DETACH LAN command*

4.6.10 Configuring Linux to connect to a Guest LAN

Now that we have created a z/VM Guest LAN and connected our guest's virtual NIC to that LAN, it can now be used by Linux. Because z/VM creates a virtual network adapter (NIC) that simulates a real OSA-Express or HiperSockets adapter, the configuration Linux for a virtual adapter is the same as it is for a real one. Please refer back to the previous sections of this book that described the configuration of the real adapters:

- ▶ For a type QDIO network adapter, see “Using OSA-Express with Linux” on page 71.
- ▶ For a type HIPERSOCKETS network adapter, see “Using HiperSockets with Linux” on page 87.

Recommendations

Linux guests connected to a z/VM Guest LAN must communicate with the physical network through a z/VM TCP/IP or Linux router. This adds both latency and increased CPU utilization to the environment. It also means that it is impossible to participate in an external VLAN. We discuss VLANs in 4.7, “Virtual Switch” on page 108. Additional subnetting is required in this environment, because the Linux guests must be on a separate subnet.

z/VM Guest LANs might, however, be appropriate in environments where physical network cards are limited, where there is a requirement for multiple Linux guests in the same z/VM LPAR to communicate with one another, and when the network activity of the Linux guests needs to be isolated from the physical network.

Given that VSWITCH can also fulfill these functions and does not have any of the drawbacks previously listed, it might be a more appropriate solution for your environment. We recommend that you use VSWITCH running in Layer 2 Switching mode rather than the standard z/VM Guest LAN.

4.7 Virtual Switch

The z/VM Virtual Switch (VSWITCH) introduced with z/VM V4.4 builds on the Guest LAN technology delivered in earlier z/VM releases. VSWITCH connects a Guest LAN to an external network using an OSA-Express port. Two additional OSA-Express ports can be specified as backups in the VSWITCH definition. The Linux guests connected to the VSWITCH are on the same subnet as the OSA-Express port or ports and other machines connected to that physical LAN segment.

The z/VM V4.4 implementation of VSWITCH operates at Layer 3 (network layer) of the OSI model. It only supports the transport of IP packets. In other words, it only can be used for TCP/IP applications. All destinations are identified as IP addresses, thus no MAC addresses are used because they are link layer independent. ARP processing is offloaded to the OSA-Express adapter. In this environment, all hosts share the same OSA-Express MAC address. In a method similar to the description in “Address Resolution Protocol and OSA-Express” on page 67, all traffic destined for the physical portion of the LAN segment is encapsulated into an Ethernet frame with the OSA-Express’s MAC as the source MAC address. For inbound packets, the OSA-Express strips the Ethernet frame and forwards the IP packet to the Virtual Switch for delivery to the guest by the destination IP address within the IP packet.

In z/VM V5.1, the VSWITCH implementation was extended to also have the ability to operate at Layer 2 (data link layer) of the OSI model. In Layer 2 mode, the VSWITCH:

- ▶ Uses the MAC destination address to send and receive Ethernet frames, even between the virtual adapters and adapters on the physical portions of the LAN segment.
- ▶ Transports Ethernet frames (not IP datagrams) to and from the operating system TCP/IP stack and the physical network.
- ▶ Does not offload ARP processing to the OSA-Express adapter; ARP processing performed by the operating system TCP/IP stack.
- ▶ Supports MAC level unicast, multicast, and broadcast.

Because the VSWITCH is essentially connected to the physical LAN, the requirement for an intermediate router between the physical and (internal) Guest LAN segments is removed. This reduces network latency. It also reduces overall CPU consumption, in some test cases by as much 30%. Removing the router also means that you no longer need specialized skills to configure and administer a VM-based or Linux-based router.

VLANs facilitate easy administration of logical groups of machines that can communicate as though they were on the same physical LAN. VSWITCH

provides support for IEEE 802.1Q VLANs. This means that Linux guests connected to a VSWITCH can participate in a VLAN environment.

Important: Avoid using the term bridging to describe a Virtual Switch. While this function looks as though it bridges between a Guest LAN and the Ethernet, *bridging* usually refers to the copying of an entire Layer 2 frame from one network to another. It also performs whatever frame translation is required on the way, such as in the case of translational bridging between Token Ring and Ethernet. VSWITCH handles only Ethernet packets, and does not qualify as a full Layer 2 network bridge.

z/VM Virtual Switch can also function in a disconnected mode, where either an OSA port is not associated, or the associated OSA does not flow traffic to the external network. It might seem that a VSWITCH without an OSA is just the same as a QDIO Guest LAN, but this is not the case; the VSWITCH provides layer 2 support and additional control over VLAN membership and handling of untagged frames. Refer to 4.8.5, “VLANs on z/VM Virtual Switch” on page 119 for more detail.

4.8 Introduction to VLANs

Starting with z/VM 4.4, VSWITCH and Guest LAN include the capability to support IEEE 802.1Q Virtual LANs within the simulated network. This section introduces the concept of Virtual LANs and some of the terminology involved.

4.8.1 What is a Virtual LAN

A virtual LAN allows a physical network to be divided administratively into separate logical networks. In effect, these logical networks operate as though they are physically independent of each other.

A VLAN-capable switch automatically manages the separation of traffic between VLANs for the devices attached to the switch. A VLAN ID can be assigned to each switch port and only packets that match the VLAN ID are sent to the device. See Figure 4-26 on page 110 for a diagram of a simple switch.

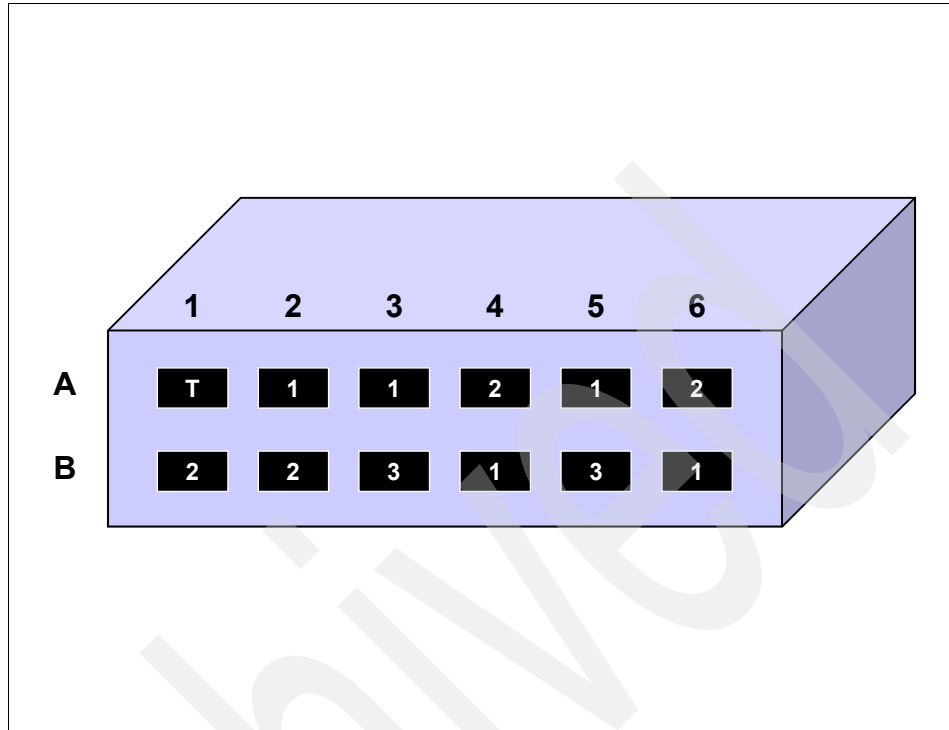


Figure 4-26 An Ethernet switch

In the figure, each port is identified by a column and row number, such as A1 for the port in the upper left and B6 for the port in the lower right. Each port in this switch except A1 is called an *access port*. These ports are defined to a single VLAN only and provide connections for non-VLAN aware devices. The number in each access port is the VLAN number assigned to that port. Only frames that match the VLAN ID of that port are sent to the device attached to the port, however, the VLAN information is removed from the frame before it is sent. Also, all packets sent from the device to the port are tagged by the switch with the same VLAN ID before being sent to their destination.

Notice that multiple access ports have the same VLAN number. All of these ports are on the same subnet and the devices attached to these ports can communicate with each other. For instance, ports B3 and B5 are in VLAN 3 and packets can be exchanged between the ports. But they cannot directly communicate with the devices attached to any of the other ports. It is as though ports B3 and B5 are a separate physical device with two ports.

Trunk ports

In Figure 4-26, port A1 is a special port called a *trunk port*, labeled as T. This port is different from an access port because packets from multiple VLANs flow in and out of it. It is used to connect switches together or to connect other VLAN aware devices. An example of two switches with connected trunk ports is shown in Figure 4-27.

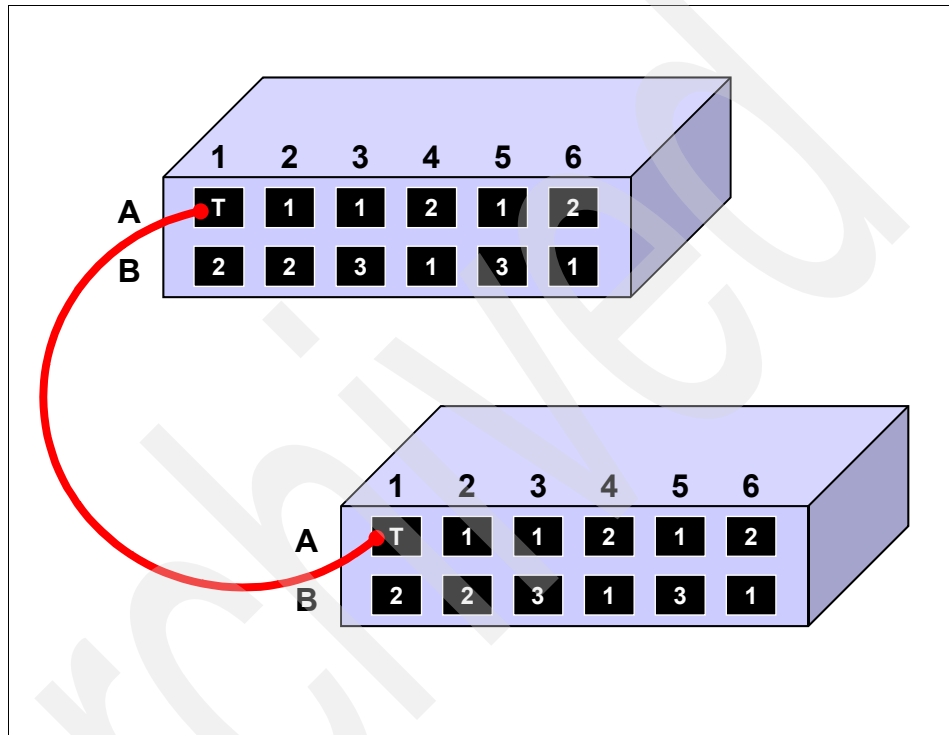


Figure 4-27 Ethernet switches connected via trunk ports

Notice that the access ports on both switches share the same VLAN IDs. The switches could be in locations separated by a great distance, yet the devices attached to them are in the same subnet, isolated from other devices. For example, VLAN 3 is assigned to two access ports on one switch and four access ports on the other. The connected trunk ports are what allow the VLAN three frames to flow from one switch to the other. This concept of trunking switches together can be extended beyond the two switches shown here.

A more typical example is shown in Figure 4-28 on page 112. This example shows how a router is used with VLANs.

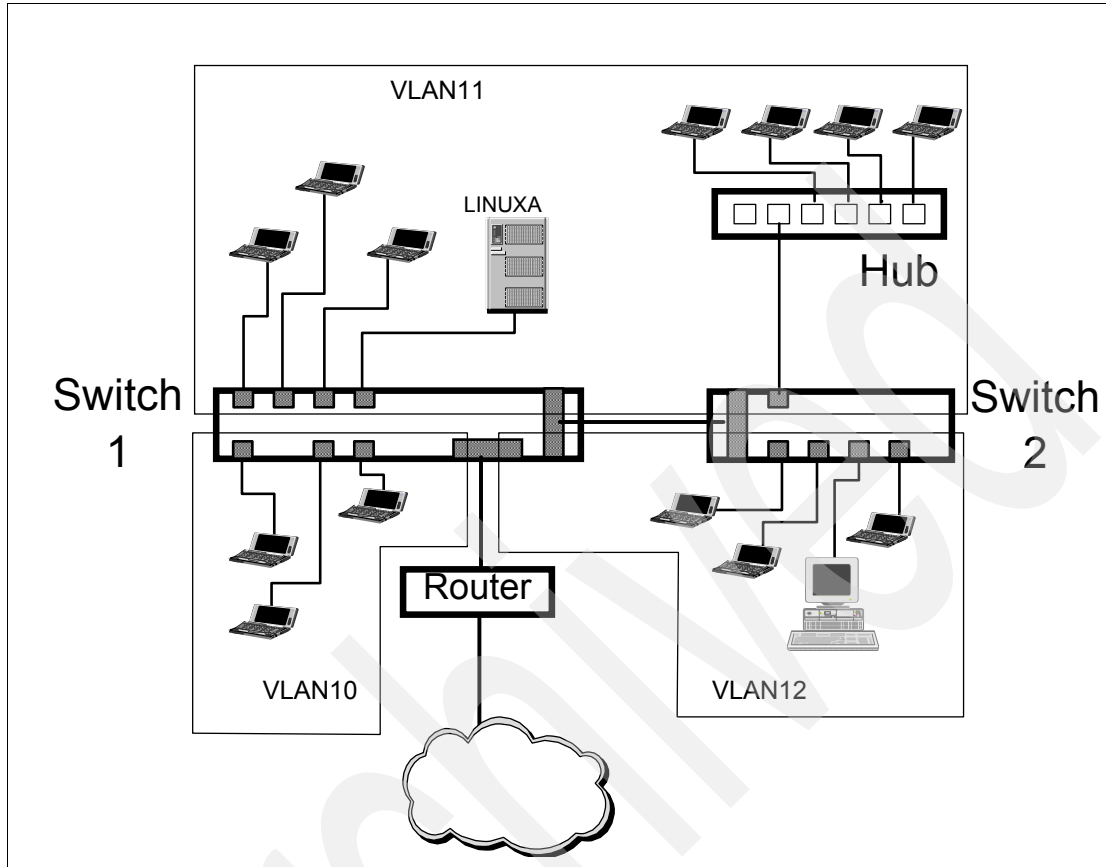


Figure 4-28 VLAN scenario with routing

We make the following observations about the network in this diagram:

- ▶ Switch ports are represented by dark squares. The larger squares are trunk ports, the small squares are access ports.
- ▶ Again we have two switches in physically separate locations, and connected by trunk ports.
- ▶ Switch 2 has a hub with several systems connected to it.
- ▶ Switch 1 has a trunk port connection to a router which, in turn, is connected to an external WAN, represented by the cloud.
- ▶ VLAN 11 is a network that exists across both locations. Data sent between VLAN 11 devices in the separate locations flows through the trunk port connection.

- ▶ The router is a VLAN-capable device, attached to one of the trunk ports. The correct definitions in the router provide a routing path between VLAN 10 and VLAN 12, and between either of these networks and the WAN. This is usually done by defining a virtual network device against the physical port on the router, linking that virtual interface to the VLAN, and enabling the interface for routing.
- ▶ VLAN 11 has no access to any other VLAN, or the WAN. Even though VLAN 11 shares access to trunk ports in both switches with VLAN 12, the VLAN architecture prevents traffic from flowing between the two networks.

If routing to VLAN 11 is required, the Switch 1 trunk port to the router could be included into VLAN 11 and a virtual interface for VLAN 11 defined in the router.
- ▶ The only machines in the entire network that are permitted to access the server LINUXA are those in VLAN 11. This is for the same reason that VLAN 11 cannot access the external network. There is no routing path between VLAN 11 and the other VLANs.

Attention: VLANs are different from Emulated LANs (ELANs). A VLAN uses the same frame format as the underlying medium, while an ELAN often uses one network technology to carry the frames of another. An example of this is Asynchronous Transfer Mode (ATM) ELAN, which allowed Ethernet and Token Ring traffic to be carried over an ATM backbone by emulating those frame formats over ATM cells.

Do not confuse VLANs and ELANS with the various simulated LAN technologies we use on zSeries and z/VM!

Later in this chapter, we present a working scenario of VLANs using a switch coupled to a z/VM Virtual Switch. See 4.10, “Layer 2 test scenario” on page 126.

4.8.2 VLAN standards

Several virtual LAN mechanisms exist; most of them are proprietary and operate on a single vendor’s equipment.

Port-based VLANs

Port-based VLANs are most often proprietary solutions that function within a single vendor’s switch hardware. They provide a method of dividing a single network device (the switch) into separate broadcast domains. How the switch accomplishes this task internally is platform-specific.

Importantly, end stations have no way to participate in multiple VLANs because the switch isolates the devices attached to it from the VLANs in the switch.

IEEE 802.1Q VLAN tagging

IEEE 802.1Q defines a standard virtual LAN mechanism that is being implemented across equipment from many different vendors. It uses a header, called the VLAN tag, added to packets transmitted over the network. The tag contains information that allows the network to manage the separation of the different virtual LANs.

Cisco Inter-Switch Link VLANs

Cisco Systems has a proprietary VLAN trunking mechanism called Inter-Switch Link (ISL). If your site uses Cisco networking equipment, you might be using ISL VLANs. ISL VLANs are not compatible with IEEE 802.1Q VLANs, but Cisco switches provide a function that allows mapping between ISL VLANs and 802.1Q VLANs.

Important: In this book we investigate the IEEE 802.1Q VLAN only, because this is the VLAN standard supported under Linux and z/VM. For the remainder of this book, when we refer to VLANs, we are specifically referring to IEEE 802.1Q VLANs (unless stated otherwise).

4.8.3 How IEEE 802.1Q VLANs work

VLAN adds additional information to the data packet. The extra data is referred to as the *VLAN tag*. The tag appears immediately after the Ethernet frame header, and before the data payload of the frame.

The format of the VLAN tag on Ethernet is shown in Figure 4-29 on page 115.

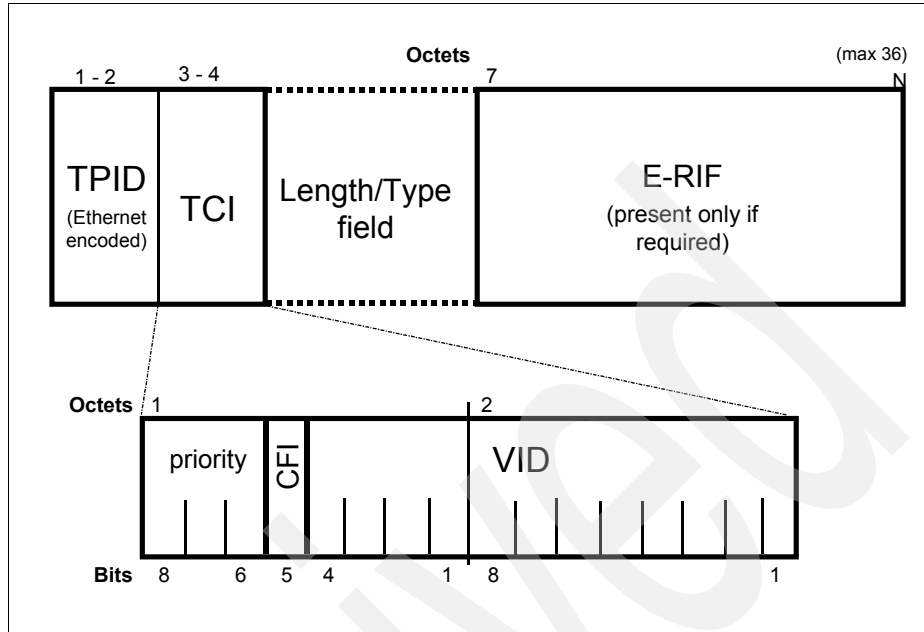


Figure 4-29 VLAN tag format for Ethernet

In most instances, only the Tag Protocol Identifier (TPID) and Tag Control Information (TCI) fields are used. This gives the impression that the VLAN header is only four bytes. In fact, the specification defines that additional information can be carried in the tag, including an Extended Routing Information Field (E-RIF) which would be used in source-routing environments (the Canonical Format Indicator (CFI) bit in the TCI indicates whether the E-RIF is present or not).

The three-bit priority field at the start of the TCI can be used by switch equipment to prioritize frames on different VLANs. On Linux, the `vconfig` command has parameters that allow the priority field to be set for the VLAN being defined.

Note: The VLAN priority is separate from IP priority mechanisms. VLAN priority is used to prioritize the frames of a VLAN relative to other VLANs, while IP prioritization operates within the IP layers of routers. Still other prioritization schemes can exist, like the traffic shaping facilities provided by Linux.

The VLAN tag is never included in a packet sent to a non-VLAN device. Part of the function of a VLAN-capable device is to add or remove VLAN tags as required, usually based on the learned capability of the peer device.

Untagged frames

A device does not have to support VLANs in order for it to join a network that uses them. A frame generated by a device that is not VLAN-capable is called an *untagged frame* when it arrives at the VLAN-capable switch.

The action taken by the switch in this case can vary. The switch can assign a default VLAN number to be assigned to any untagged frames that enter the switch, or it can tag the frames with a port-specific VLAN number, providing a function similar to a port-based VLAN.

Connecting to a VLAN

VLAN-capable devices attach to VLANs through the use of virtual network interfaces. Each VLAN has its own separate virtual interface. The diagram in Figure 4-30 on page 117 shows the layered relationship of the components involved in packet transmission.

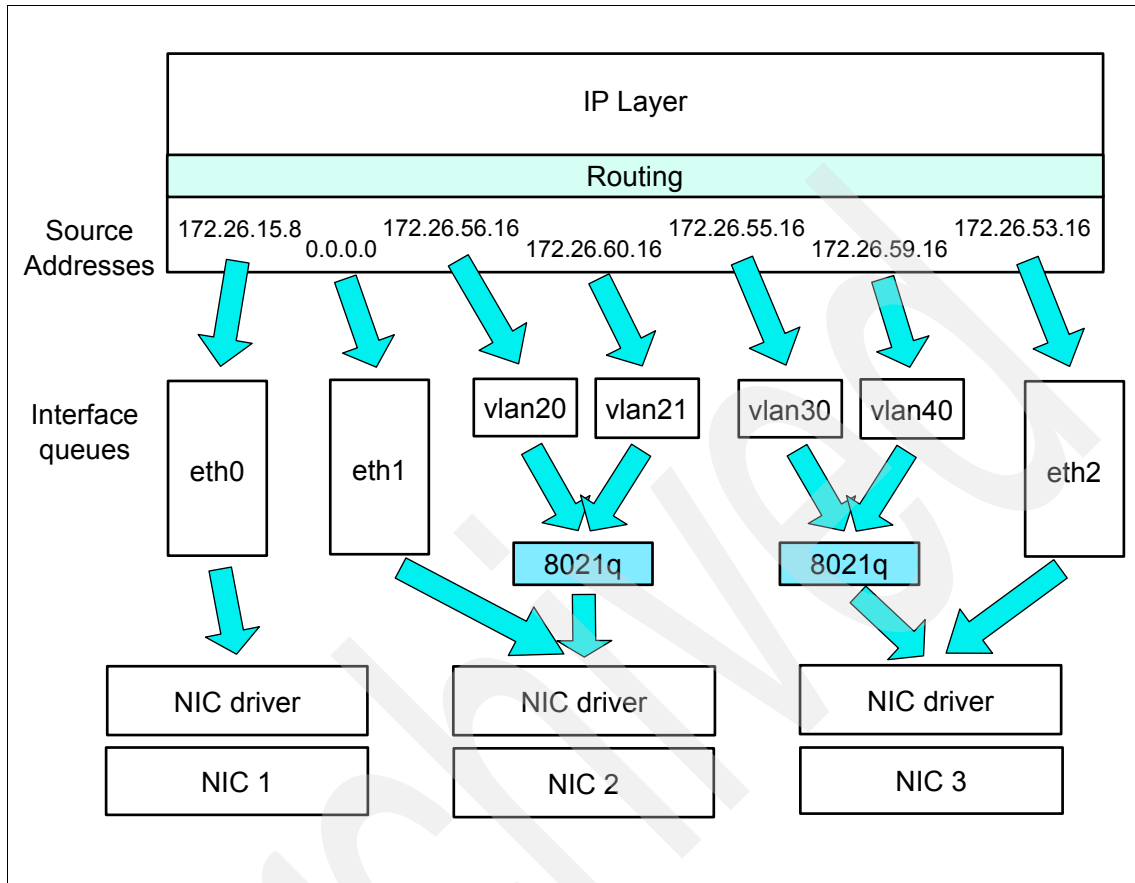


Figure 4-30 Layer diagram for VLAN-capable devices

Attention: This diagram is not meant to be a representation of the way that VLANs are implemented in any particular device. It is a conceptual overview only.

In Figure 4-30, *eth0* is a non-VLAN device. After a routing decision has selected *eth0* as the interface for the packet to be transmitted on, the IP layer, places the packet onto the queue for *eth0*. Because *eth0* is non-VLAN, when the packet gets to the front of the queue it is given to the NIC driver for encoding and transmission in the network with no other action. If the switch that this interface connects to is VLAN-aware, it must be configured to handle these frames. See “Untagged frames” on page 116 for more information.

The *eth1* interface supports connections to two VLANs, 20 and 21. The virtual interfaces for these VLANs are *vlan20* and *vlan21*, respectively. As for any other interface, packets can arrive at the queues for these interfaces as a result of the IP layer making a routing decision. For VLAN interfaces, however, the kernel VLAN code processes the packet by adding the VLAN tag and passing the tagged packet to the driver for the interface to which the VLAN belongs.

Note: In Linux, when VLAN interfaces are configured, the physical interface the VLAN is associated with must be provided as part of the configuration. This allows the VLAN support to send the packet to the correct interface queue.

The IP address associated with *eth1* is a *dummy* address (that is, an address which does not match a real network configuration). Figure 4-30 on page 117 shows *eth1* configured as 0.0.0.0, which is commonly used for this purpose because it can never be used as a source address. The practical effect of this is that the IP interface *eth1* will not generate any traffic, because the IP layer does not direct any packets to that interface for transmission, nor will the network send any packets to that address. This means that switch port does not need any configuration to support untagged frames. It can operate purely as a trunk port.

The *eth2* interface has an almost identical configuration to *eth1*, except for the address associated with it. Here, a valid IP address is configured on the *eth2* interface. When the IP layer selects 172.26.53.16 as a source address, the packet is not processed by VLAN, resulting in an untagged frame being transmitted. The switch that *eth2* is connected to must know what to do with such frames arriving when tagged frames are expected. The 802.1Q specification refers to this kind of port as a *hybrid port*, it can act both as a trunk port to handle tagged frames, and as an access port to handle untagged frames.

Some equipment, and also z/VM 5.1, define a default VLAN ID used for any interfaces that are not defined with an explicit VLAN ID specified. This avoids a problem with a device generating untagged frames when communicating with VLAN-aware switch. Linux, on the other hand, can generate untagged frames in the way described for *eth2* when connected to a trunk port, even if that port is defined with permissions for a specific list of VLAN IDs. Some equipment associates these frames with a native or internal VLAN used for control or management of the equipment. It is possible that these frames could cause problems or interference, so it is best to not allow them to be created.

Note: Using untagged frames in a VLAN environment can result in unexpected connectivity problems and potential security issues. To avoid this, always ensure that network traffic specifies a VLAN ID. You can do this by using a dummy address on the base interface, as illustrated with eth1 in Figure 4-30 on page 117. An example of what can happen with untagged frames in a VLAN environment is shown in 4.8.6, “VLAN isolation” on page 120.

4.8.4 VLAN support on z/VM Guest LAN

Starting with z/VM 4.4, a simulated QDIO or HiperSockets network can provide IEEE 802.1Q VLAN function to attached guests. The Guest LAN simulation works in exactly the same manner as in previous z/VM releases, except that it can pass VLAN tagged frames between guests that are VLAN-aware. When using VLANs over Guest LAN, the Guest LAN simulation does not interfere with the VLAN tagging of network frames. This means that a guest attached to a Guest LAN *must* be VLAN-aware in order to participate in the VLAN network.

Important: At this time, the HiperSockets microcode on zSeries processors does not support VLAN-tagged frames. VLAN support applies only to simulated HiperSockets on z/VM 4.4.

Another feature added to z/VM simulated HiperSockets with z/VM 4.4 is *broadcast capability*. This means that guests attached to a simulated HiperSockets can now use applications and protocols that use IP broadcast, in addition to the IP multicast support added in a previous release.

4.8.5 VLANs on z/VM Virtual Switch

z/VM Virtual Switch supports IEEE 802.1Q VLANs. This means that guests attached to a VSWITCH under z/VM 4.4 or later can participate in VLAN networking. When an OSA-Express port is attached to the VSWITCH, VLAN operations extend between the VSWITCH and the LAN via the OSA-Express.

A VSWITCH can be created as a VLAN-aware switch and it performs VLAN tag processing according to its configuration. The guest need not be VLAN-aware in order to communicate to a specific VLAN. In the case where the guest is coupled to a virtual access port on the VSWITCH, the VSWITCH adds the correct VLAN tag to frames as they leave the guest and remove it when the guest receives frames. If the configuration of the VSWITCH does not specify a VLAN ID for that guest, the frames are sent and received using the default VSWITCH VLAN ID.

Note: The guest does not require any configuration for VLAN support. Guests that do not support VLAN send and receive frames on the VLAN with which the CP has been configured to connect them.

VSWITCH can provide many options for controlling which guests receive frames on which VLANs. This is called *VLAN filtering* and is described in ““VSWITCH VLAN filtering” on page 120”.

If an OSA Express is associated with your VLAN-aware VSWITCH, this OSA Express port functions similar to a trunk port with access to all of the VLAN IDs appearing in the VSWITCH. When you configure the port on the LAN switch the OSA Express connects to, you can choose to make that port belong to any of the VLAN numbers configured in the VSWITCH, extending those VLANs out of the VSWITCH into the LAN. If the VLAN IDs already exist in the LAN, the guests on the VSWITCH are present on the same VLAN.

4.8.6 VLAN isolation

This section discusses VLAN isolation.

VSWITCH VLAN filtering

VSWITCH provides an additional feature that provides further isolation. Using either the CP **SET VSWITCH** command or an external security manager in z/VM 5.1 or later, you can control VLAN access by VM user ID in the VSWITCH. This is analogous to defining the VLAN membership of a trunk port on a LAN switch. You specify the VLAN IDs that a guest is allowed to see. Only frames tagged with those VLAN IDs are passed to the guest.

Attention: In z/VM 4.4, there appears to be one exception. Regardless of the VLAN IDs specified, untagged frames from within the VSWITCH are always passed to the guest if the destination IP address matches an address that the guest has configured. This could result in unauthorized access to different network resources. This exception has been fixed in z/VM 5.1 by removing the option **VLAN ANY** on the **DEFINE VSWITCH** command. Now a VLAN-aware VSWITCH defines a default VLAN ID applied to untagged frames.

We recommend that you do not use untagged frames in your environment. For VLAN-unaware guests, use the CP **SET VSWITCH** command to grant the guest access to one VLAN ID only. For VLAN capable systems, always configure the base interface with a dummy IP address as described in “Connecting to a VLAN” on page 116.

The rules that apply to packet delivery on a z/VM Virtual Switch are explained in the “Working with Virtual Networks” chapter of *z/VM Virtual Machine Operation*, SC24-5955.

4.9 VSWITCH configuration

Complete the following required steps to implement VSWITCH:

1. Configure one or more z/VM TCP/IP virtual machines to act as controllers for the VSWITCH.
2. Define a VSWITCH to act as a LAN segment for the virtual machines.
3. Create a simulated Network Interface Card (NIC) on each virtual machine.
4. Link the Linux system to the VSWITCH.

4.9.1 Transport mode: IP or Ethernet

A VSWITCH is defined in either IP or Ethernet data transport mode. Additionally, this mode applies to all connections made to it. Here are some things to consider when deciding which mode to deploy:

- ▶ Do your servers or applications need to have their own unique MAC addresses? Load Balancers are an example of servers with this requirement.
- ▶ Do you plan to deploy non-IP based applications in your network? SNA or NetBIOS applications are an example of this.
- ▶ Does your virtual LAN segment need to operate in a manner that closely resembles a physical LAN segment?

4.9.2 Configure controller service machines

The VSWITCH's connection to an OSA-Express interface is enabled by a controller virtual machine. A controller is a z/VM service machine running the VM TCP/IP stack. In order for a VSWITCH to provide connectivity to a physical LAN, at least one TCP/IP service machine must be configured to be a controller which manages the VSWITCH connection. At least one TCP/IP service machine must be configured to be a controller.

Recommendation: We recommend that you define at least two new TCP/IP service machines dedicated to this purpose for backup and isolation reasons. Do not add this function to your primary TCP/IP service machine. In our example, we define two virtual machines to act as VSWITCH controllers.

Figure 4-31 shows the lines that we added to the VM User Directory for our controller machines.

```
USER VSWCTL1 TCPIP 32M 128M ABCG
  INCLUDE TPCMSU
  IUCV *VSWITCH MSGLIMIT 65535
  OPTION QUICKDSP SVMSTAT MAXCONN 1024 DIAG98 APPLMON
  SHARE RELATIVE 3000
  LINK 5VMTCP10 0491 0491 RR
  LINK 5VMTCP10 0492 0492 RR
  LINK TCPMAINT 0591 0591 RR
  LINK TCPMAINT 0592 0592 RR
  LINK TCPMAINT 0198 0198 RR
  MDISK 0191 3390 66 1 LX4USR MR

USER VSWCTL2 TCPIP 32M 128M ABCG
  INCLUDE TPCMSU
  IUCV *VSWITCH MSGLIMIT 65535
  OPTION QUICKDSP SVMSTAT MAXCONN 1024 DIAG98 APPLMON
  SHARE RELATIVE 3000
  LINK 5VMTCP10 0491 0491 RR
  LINK 5VMTCP10 0492 0492 RR
  LINK TCPMAINT 0591 0591 RR
  LINK TCPMAINT 0592 0592 RR
  LINK TCPMAINT 0198 0198 RR
  MDISK 0191 3390 67 1 LX4USR MR
```

Figure 4-31 VSWITCH controller user definitions

Note that the user entries include a IUCV *VSWITCH statement, which is required for the machines to be considered as VSWITCH controllers.

Note: Unlike previous networking configurations that used the z/VM TCP/IP stack, there is no requirement to configure manually IP addresses or devices when using VSWITCH. The only settings required in the stack's PROFILE TCP/IP file are listed in Figure 4-33 on page 123.

AUTOLOG changes

For the controller service machines to start automatically when VM is IPLed, we added two lines to our AUTOLOG1 PROFILE EXEC, as shown in Figure 4-32 on page 123. Other commands required for VM system setup can also be placed in this file or the system configuration file, including defining your VSWITCH devices. This is described later.


```
'CP XAUTOLOG VSWCTL1'  
'CP XAUTOLOG VSWCTL2'
```

Figure 4-32 PROFILE EXEC for AUTOLOG1

Profile for TCP/IP and PROFILE EXEC

On z/VM 5.1, create files named VSWCTL1 TCPIP and VSWCTL2 TCPIP on the TCPMAINT 198 disk with the contents shown in Figure 4-33. On earlier VM releases, you have to name this file PROFILE TCPIP and place it on the 191 disk of each VSWITCH controller machine.

```
; Simple profile for a VSWITCH controller machine  
OBEY  
OPERATOR TCPMAINT MAINT  
ENDOBAY  
VSWITCH CONTROLLER ON
```

Figure 4-33 VSWCTL1 TCPIP file

Create a PROFILE EXEC on the 191 disk of each controller service machine. This can either be copied from the PROFILE EXEC of the existing TCP/IP service machine or copy the file TCPROFIL EXEC file on TCPMAINT 591 to the 191 disk as PROFILE EXEC.

The SYSTEM DTCPARMS file

Modify the SYSTEM DTCPARMS file on the TCPMAINT 198 disk and add the lines shown in Figure 4-34. This designates that the controller machines will run the TCP/IP stack code.

```
:nick.VSWCTL1 :type.server :class.stack  
:nick.VSWCTL2 :type.server :class.stack
```

Figure 4-34 Lines in SYSTEM DTCPARMS

When you have made all the configuration changes, start the controller machines using the XAUTOLOG command:

```
XAUTOLOG VSWCTL1  
XAUTOLOG VSWCTL2
```

You can verify that the controller machine have initialized correctly using the **QUERY CONTROLLER** command. You should see output similar to the following:

```

Q CONTROLLER
Controller VSWCTL1 Available: YES VDEV Range: * Level 510
Capability: IP ETHERNET VLAN_ARP
Controller VSWCTL2 Available: YES VDEV Range: * Level 510
Capability: IP ETHERNET VLAN_ARP

```

If you only see one controller or the message: “HCPSWQ2835E CONTROLLER * does not exist,” then you must log on to the controller VMs to see what error has prevented their start up.

Important: The controller virtual machines (VSWCTL1 and VSWCTL2) are involved only in device initialization. After the VMs are initialized, data transfers occur directly between the OSA device and the Linux guest without passing through the controller stack.

4.9.3 Defining a VSWITCH

A VSWITCH is created using the **CP DEFINE VSWITCH** command from a user ID with Class B privileges. Or, a **DEFINE VSWITCH** statement can be added to the **SYSTEM CONFIG** file and the switch will be created during system start up.

The syntax of the **DEFINE VSWITCH** statement is as follows:

```
DEFINE VSWITCH switchname [ operands ]
```

Where the following is true:

switchname Is the name of the Virtual Switch.

operands Define the attributes of the Virtual Switch.

Table 4-13 summarizes the common operands accepted by the DEFINE VSWITCH statement and command.

Table 4-13 Common operands of the DEFINE VSWITCH statement

Operands	Description
RDEV <i>rdev-list</i>	A real device address to be used to connect the Virtual Switch to a QDIO OSA-Express device. You can specify a maximum of three real device numbers. Each real device address represents a trio of devices. For example, specifying RDEV 111 222 333 means that the first devices, 111-113, are used to provide the connection to the real hardware LAN segment. If there is a problem with the connection, devices 222-224 are used next to provide the connection, and if those devices fail to connect, devices 333-335 are used. This feature provides dynamic recovery for OSA-Express device failures.
CONnect	Indicates that the device identified by the RDEV keyword must be activated, and traffic must flow through the device to the real LAN segment.
CONTRoller * or CONTRoller <i>userid</i>	Identifies the z/VM user ID that controls the OSA-Express device connected at the device address identified by rdev. CONTROLLER * means that CP selects from any of the eligible z/VM TCP/IP stacks. If you specify multiple real devices on the RDEV keyword, specify CONTROLLER *, or allow it to default. The controller functions are then spread across multiple z/VM TCP/IP stacks, providing more flexibility in case of a failure.
IP or ETHernet	Indicates whether the transport for the Virtual Switch is ETHERNET or IP. An ETHERNET Virtual Switch operates at the Layer 2 level of the OSI model, and an IP Virtual Switch operates at Layer 3.
VLAN <i>defvid</i>	Defines the default VLAN id associated with untagged frames on this switch. The default is VLAN UNAWARE, which indicates that the virtual switch will ignore VLAN tags.
PORTtype <i>type</i>	Defines the default port type for guests attached to this virtual switch. The <i>type</i> can be ACCESS or TRUNK. This operand is not valid if VLAN UNAWARE is specified.
PORTname <i>portname</i>	A 1-to-8 character name that identifies the OSA-Express adapter. You can specify a maximum of three port names. Multiple port names are used when different port names are needed for the multiple rdevs specified on the RDEV operand. ^a

- a. See “OSA port names” on page 74 for more information about port names.

Linking a Linux guest to the VSWITCH is exactly the same as linking a guest to a Guest LAN. See 4.6.4, “Create a virtual Network Interface Card” on page 100 for the description of that process.

4.9.4 VSWITCH failover support

Failover support for Virtual Switches provides recovery for controller failures and OSA-Express port failures. A redundant system would use two or three virtual controllers and two or three separate OSA cards. If one controller fails, another controller takes over. A port on each card is plugged into an access port on the same subnet, which can be in separate physical switches if the switches are linked together, or trunk ports with identical configurations. If one OSA port fails, traffic is switched to use another OSA port.

On a z/VM 5.1 system, when a VSWITCH is defined with multiple OSA devices and multiple virtual controllers are available, z/VM pairs backup controllers to the backup OSA devices, attaches them to the controllers, and performs as much initialization as possible for the backup ports. This reduces the elapsed time to recover from an error.

In the event of an OSA or controller failure, data transfer to the network is suspended and the data transfer queues are moved to the backup device. Output traffic is queued while this occurs. After the backup device is initialized, the queued traffic is sent. Some input traffic might be lost during the period when no OSA port is responding to the real LAN segment.

4.10 Layer 2 test scenario

To demonstrate the Layer 2 function available in the z/VM 5.1 Virtual Switch, we created a test scenario with several virtual and real Linux servers in a simulated web hosting environment. We wanted to demonstrate VLAN separation, Layer 2 (MAC) addressing, multiple interfaces, failover, and the use of external devices working with the virtual servers. This can be a complicated environment, but we try to keep it as simple as possible for this demonstration. Real world environments can be even more complicated in order to follow security and processing needs, but we hope that our demonstration gets you started.

4.10.1 Requirements

To use Layer 2 LAN Switching, you need the following prerequisites:

- ▶ zSeries 990 or 890 server at Driver Level 55K

- ▶ OSA-Express or OSA-Express2 Ethernet adapter
The code level must be at least 6.23 (available in October 2004). We used code level 6.27 in our testing and we recommend that this level or later is used.
- ▶ z/VM V5.1 with APAR VM63538
In our testing, we used z/VM 5.1 at service level 0501 with APAR VM63665. Enter QUERY VMLAN on your VM system to see the service level of the CP networking support
- ▶ Linux qeth device driver that supports Layer 2. Layer 2 support for the June 2003 stream (kernel 2.4.21) was released on developerWorks in October 2004. Layer 2 support for the April 2004 stream (kernel 2.6.5) was released on developerWorks in March, 2005. Check with your distributor to see if Layer 2 support is available in your Linux distribution. In our testing, we used SUSE SLES 8 running with kernel 2.4.21-281 and the qeth driver supplied with the kernel.

Planning for unique MAC addresses

MAC addresses used by z/VM virtual adapters must be unique within your physical network. To ensure uniqueness, each z/VM system should have a unique MAC address prefix. The address prefix is six hexadecimal digits in the range of 020000 to 02FFFF, inclusive. This range falls within the allowable range of addresses classified as Locally Administered.

The administrator of the VM system should work with the LAN administrator to decide what MAC addresses are locally generated and associated with each VM system and its guests. They assign MAC addresses using a combination of the VMLAN statement in SYSTEM CONFIG and the NICDEF statement in the User Directory.

The VMLAN statement (4.6.3, “The VMLAN statement” on page 98) contains a MACPREFIX parameter that enables the administrator to specify a three-byte manufacturer ID prefix for all MAC addresses in this z/VM system. The VMLAN parameter MACIDRANGE controls the range of identifiers that can be used by CP when generating the unique identifier component of a virtual NIC’s MAC address.

In the User Directory, a NICDEF statement is added to each guest connected to the VSWITCH. The optional MACID parameter of NICDEF enables the administrator to specify a unique identifier that appends to the MACPREFIX to form a unique MAC address for that guest. If MACID is omitted, CP generates a unique identifier based on the range specified in the VMLAN MACIDRANGE parameter. If you specify your own MACID value in the NICDEF, and the MACID

is already in use, the adapter is not be created, and you need to remove the conflicting device or select a different MACID.

These locally generated MAC addresses are visible across the physical portion of the LAN segment. In our test, we specified the default MACPREFIX of 020000 on the VMLAN statement. This is not a good idea in your own datacenter, but it was sufficient for our tests.

4.10.2 Test overview

Our sample test environment has two sample customers who could be just two departments of the same company with a Web application available on the Internet. One customer requires two load-balanced Web servers, and the other has just one. Also, one customer requires that their Web server is behind a firewall; the other does not. We attempt to convince this customer at a later time that a firewall should be a requirement! Our example Web hosting company stores their Web pages on a common file server. Our company also requires a separate administrative network that is connected to each server and is used for administration of the Linux servers, back ups, software installation, and so forth. This administrative network is also protected by firewalls to prevent one customer from accessing the other's servers.

We want to build this environment using a combination of virtual Linux servers and real servers. The decision of when to use a virtual versus a real server depends on the usage of the server, its utilization, software and hardware support, and so on. Our configuration gives us the freedom of moving a server function from a real to a virtual server or vice versa, due to the flexibility of our networking options.

Figure 4-35 on page 129 is a logical diagram of our network.

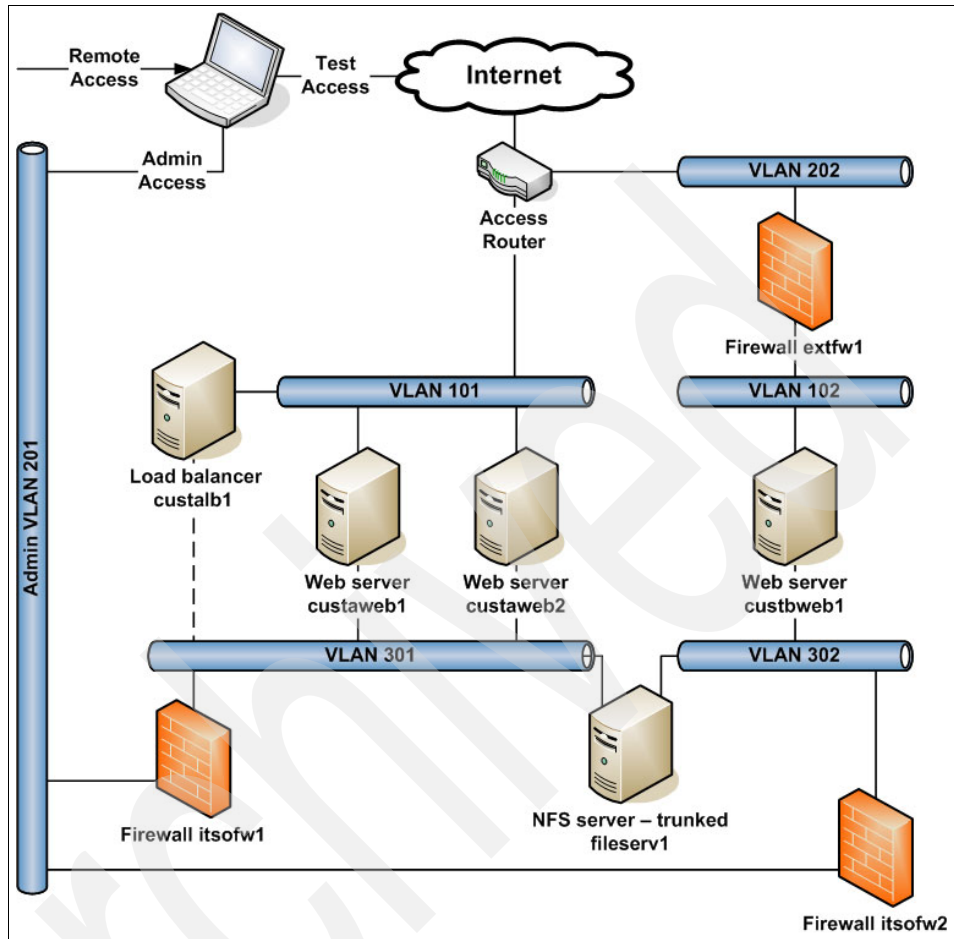


Figure 4-35 Example test network - logical view

The physical implementation is shown in Figure 4-36.

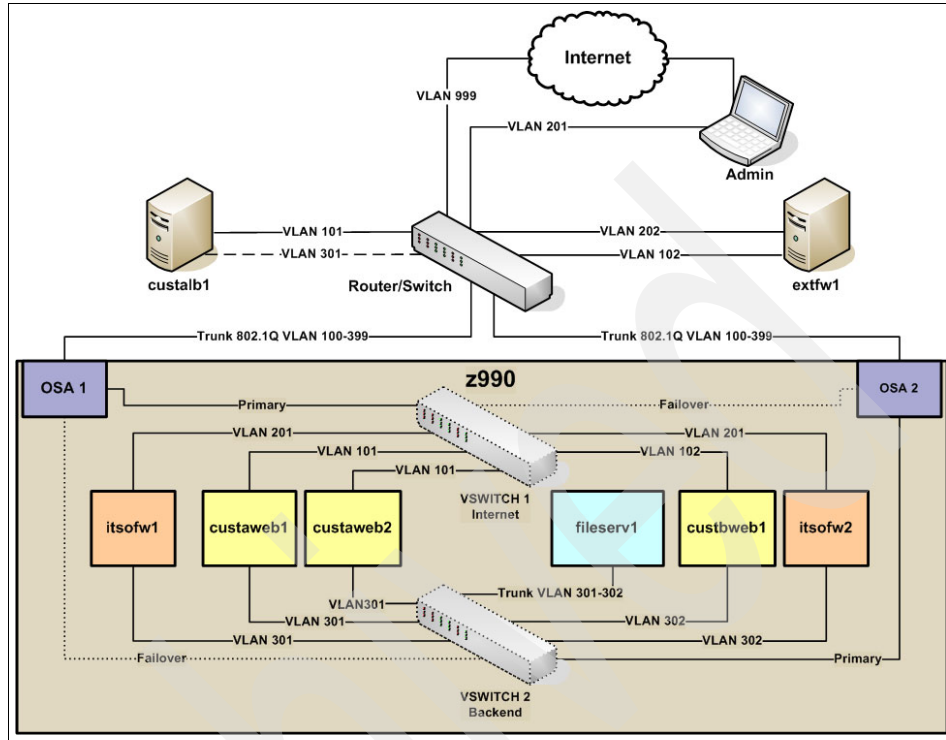


Figure 4-36 Example test network: physical view

There are many virtual LANS (VLANs) defined. Table 4-14 defines the purpose of each one. A numbering scheme was used for the VLAN IDs to make the management and usage of each one more clear. VLAN IDs in the range 101-199 are Internet accessible, 201-299 are only accessible from inside or private networks, and 301-399 are used between firewalls and administrative interfaces of the servers.

Table 4-14 Example network VLAN definitions

VLAN id	Base IP and mask	Usage
Internet facing networks		
VLAN 101	192.168.1.0/24	"Customer A" Internet addresses
VLAN 102	192.168.2.0/24	"Customer B" Internet addresses
Interconnection networks		
VLAN 201	10.1.251.0/24	Administrative access

VLAN id	Base IP and mask	Usage
VLAN 202	10.1.252.0/24	Intermediate access
Administrative access networks		
VLAN 301	10.1.1.0/24	“Customer A” back end network
VLAN 302	10.1.2.0/24	“Customer B” back end network

IP addresses and routing

We used the Internet standard private IP ranges for our entire test network, with the assumption that addresses starting with 192.168 represent Internet facing addresses and addresses starting with 10.1 represent internal transport or administrative addresses. Table 4-15 documents the IP address assignments. Static routes are used to direct the administrative access through the proper interface. See Table 4-16 on page 132 for a list of the routing assignments. All IP addresses ending in 250 are the assigned to the external access router which is not shown in the table.

Table 4-15 Example network IP addresses

IP name	Interface	IP address	VLAN	Usage
custalb1	eth0	192.168.1.1	101	External load balancer
	eth0:0	192.168.1.2	101	
custaweb1	eth0	192.168.1.3	101	Customer A web server 1
	eth1	10.1.1.1	301	
custaweb2	eth0	192.168.1.4	101	Customer A web server 2
	eth1	10.1.1.2	301	
extfw1	eth0	10.1.252.1	202	Customer B external firewall
	eth1	192.168.2.5	102	
custbweb1	eth0	192.168.2.1	102	Customer B web server
	eth1	10.1.2.1	301	
itsofw1	eth0	10.1.251.1	201	Customer A administrative firewall
	eth1	10.1.1.5	301	
itsofw2	eth0	10.1.251.2	201	Customer A administrative firewall
	eth1	10.1.2.5	302	
fileserv1	vlan301	10.1.1.10	301	File server with trunk access
	vlan302	10.1.2.10	302	

Table 4-16 Routing table

IP name	Network	Netmask	Gateway	Interface
custalb1	Default	n/a	192.168.1.250 ^a	eth0
custaweb1	10.1.0.0	255.255.0.0	10.1.1.5	eth1
	Default	n/a	192.168.1.250 ^a	eth0
custaweb2	10.1.0.0	255.255.0.0	10.1.1.5	eth1
	Default	n/a	192.168.1.250 ^a	eth0
extfw1	Default	n/a	10.1.252.250 ^a	eth0
custbweb1	10.1.0.0	255.255.0.0	10.1.2.5	eth1
	Default	n/a	192.168.2.250 ^a	eth0
itsofw1	Default	n/a	10.1.251.250 ^a	eth0
itsofw2	Default	n/a	10.1.251.250 ^a	eth0
fileserv1	Default	n/a	10.1.1.5	vlan301

a. This address is defined on the access router outside of the test network.

4.10.3 External devices

Our test environment used a Cisco Catalyst 6509 switch that has VLAN and routing support.

The servers External Load balancer A and External FW (firewall) are devices external to z/VM. They could be either commercially available, off-the-shelf devices dedicated to this function or they could be general purpose servers programmed to perform this function. In our case, they were identical desktop Personal Computers loaded with SUSE Linux SLES 9 and programmed to perform different roles. Their configuration is described later.

4.10.4 z/VM system configuration

Our test network required at a minimum one connection from an OSA-Express port to a trunk port on the physical switch. See the description of this type of port in “Trunk ports” on page 111. We also required back up network connections, so we used two OSA ports connected to two switch trunk ports. The majority of our virtual servers connect to two different subnets, two different VLANs, so two virtual switches are required to implement the required VLAN access restrictions. Because we have two OSA ports and two Virtual Switches, it is logical to assign one OSA port as the primary interface to one switch and the other OSA port as the primary to the other. This way both ports are utilized, but at the same time both are available for backup. If a single OSA port cannot handle all of the

network traffic for your environment during a failure of one of the ports, then additional OSA ports are needed for backup.

The device addresses for our OSA ports are 2D40 and 2E40, with 2D43 and 2E43 as the address of the backup devices. As described in “Hardware configuration” on page 70, each OSA port actually uses three device addresses.

Defining Virtual Switches

One or more VSWITCH controller machines are required to be operating to use a Virtual Switch, so make sure the tasks described in 4.9.2, “Configure controller service machines” on page 121 have been completed. We added statements to our SYSTEM CONFIG file to define our Virtual Switches during system start up. See “Defining a VSWITCH” on page 124 for the syntax of the DEFINE command. Here are the lines from our SYSTEM CONFIG file:

```
Define Vswitch VSWTCHN1 Rdev 2D40 2E43 Ethernet VLAN 1 Porttype ACCESS
Define Vswitch VSWTCHN2 Rdev 2E40 2D43 Ethernet VLAN 1 Porttype ACCESS
```

Notice that we have specified Ethernet for Layer 2 mode. The default VLAN ID is 1, which matches the native VLAN ID of our switch. And the default connection is through an access port. A VSWITCH can also be defined dynamically with a CP **DEFINE** command. In our example, the syntax of the command is the same as the SYSTEM CONFIG statement.

Another line that should be added to the SYSTEM CONFIG file is a VMLAN statement. This is to ensure that the virtual MAC addresses created in the z/VM system are unique in your network. See “Planning for unique MAC addresses” on page 127 for further information. In our test system we added this line:

```
VMLAN MACPREFIX 020000
```

This is also the default value if no VMLAN MACPREFIX statement is in the SYSTEM CONFIG file. We recommend that you do not use this default value in your systems, but assign a unique number to each one. In our example, our system is the only one on the isolated network.

Defining virtual Linux servers

Our example network has six virtual Linux servers in it. The VM user ID, host name, VLAN access, and usage of each server is shown in Table 4-17

Table 4-17 VM user id list

VM User id	Host name	VLANs	Usage
LNXNFW1	itsofw1	201 301	Admin firewall A

VM User id	Host name	VLANs	Usage
LNXNFW2	itsofw2	201 302	Admin firewall B
LNXNWA1	custaweb1	101 301	Customer A web server 1
LNXNWA2	custaweb2	101 301	Customer A web server 2
LNXNWB1	custbweb1	102 302	Customer B web server 1
LXNDFS1	fileserv1	301 302	Common file server

The column of VLAN numbers is the VLAN IDs that each host is connected to, as also shown in Figure 4-35 on page 129.

Each user ID except LXNDFS1 is connected to both Virtual Switches. The directory entry for each user has the following lines:

```
NICDEF 0600 TYPE QDIO LAN SYSTEM VSWTCHN1
NICDEF 0610 TYPE QDIO LAN SYSTEM VSWTCHN2
```

User id LXNDFS1 only connects to VSWTCHN2, using this line in its directory:

```
NICDEF 0600 TYPE QDIO LAN SYSTEM VSWTCHN2
```

Defining VLAN permissions

Only VM user IDs that have been granted permission are allowed to connect a virtual NIC to a Virtual Switch. Included in this permission is the type of access (ACCESS or TRUNK) and the permitted VLAN id (ACCESS port) or ids (TRUNK port.) Each user ID and Virtual Switch requires a separate command or definition. There are three ways to define this access: SYSTEM CONFIG file statements, CP commands, or RACF/VM permissions.

SYSTEM CONFIG file

The following lines in Example 4-1 are added after the Define Vswitch lines:

Example 4-1 VLAN permissions in the SYSTEM CONFIG file

```
Modify Vswitch VSWTCHN1 Grant LNXNFW1 Porttype ACCESS VLAN 201
Modify Vswitch VSWTCHN2 Grant LNXNFW1 Porttype ACCESS VLAN 301
Modify Vswitch VSWTCHN1 Grant LNXNFW2 Porttype ACCESS VLAN 201
Modify Vswitch VSWTCHN2 Grant LNXNFW2 Porttype ACCESS VLAN 302
Modify Vswitch VSWTCHN1 Grant LNXNWA1 Porttype ACCESS VLAN 101
Modify Vswitch VSWTCHN2 Grant LNXNWA1 Porttype ACCESS VLAN 301
```

```
Modify Vswitch VSWTCHN1 Grant LNXNWA2 Porttype ACCESS VLAN 101
Modify Vswitch VSWTCHN2 Grant LNXNWA2 Porttype ACCESS VLAN 301
Modify Vswitch VSWTCHN1 Grant LNXNWB1 Porttype ACCESS VLAN 102
Modify Vswitch VSWTCHN2 Grant LNXNWB1 Porttype ACCESS VLAN 302
Modify Vswitch VSWTCHN2 Grant LNXNFS1 Porttype TRUNK VLAN 301 302
```

These permissions are automatically granted during every VM system IPL.

CP commands

These commands can be executed as part of the system start up process, if the SYSTEM CONFIG file statements are not used, or used to dynamically set up the test environment.

Example 4-2 VLAN permissions as CP commands

```
CP SET VSWITCH VSWTCHN1 GRANT LNXNFW1 PORTTYPE ACCESS VLAN 201
CP SET VSWITCH VSWTCHN2 GRANT LNXNFW1 PORTTYPE ACCESS VLAN 301
CP SET VSWITCH VSWTCHN1 GRANT LNXNFW2 PORTTYPE ACCESS VLAN 201
CP SET VSWITCH VSWTCHN2 GRANT LNXNFW2 PORTTYPE ACCESS VLAN 302
CP SET VSWITCH VSWTCHN1 GRANT LNXNWA1 PORTTYPE ACCESS VLAN 101
CP SET VSWITCH VSWTCHN2 GRANT LNXNWA1 PORTTYPE ACCESS VLAN 301
CP SET VSWITCH VSWTCHN1 GRANT LNXNWA2 PORTTYPE ACCESS VLAN 101
CP SET VSWITCH VSWTCHN2 GRANT LNXNWA2 PORTTYPE ACCESS VLAN 301
CP SET VSWITCH VSWTCHN1 GRANT LNXNWB1 PORTTYPE ACCESS VLAN 102
CP SET VSWITCH VSWTCHN2 GRANT LNXNWB1 PORTTYPE ACCESS VLAN 302
CP SET VSWITCH VSWTCHN2 GRANT LNXNFS1 PORTTYPE TRUNK VLAN 301 302
```

RACF/VM permissions

We recommend that your VM systems use an external security manager, especially for production systems. Starting with z/VM 5.1, RACF/VM can control access to a VSWITCH. A complete description of how to set up RACF on your system is outside the scope of this book. However, if you have it set up, see Figure 4-37 on page 136 for the commands to enable VSWITCH access in our environment.

```

RACF
RDEFINE VMLAN SYSTEM.VSWTCHN1 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN2 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN1.0101 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN1.0102 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN1.0201 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN2.0301 UACC(NONE)
RDEFINE VMLAN SYSTEM.VSWTCHN2.0302 UACC(NONE)
PERMIT SYSTEM.VSWTCHN1 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW1 LNXNFW2 LNXNWA1 LNXNWA2
LNXNWB1)
PERMIT SYSTEM.VSWTCHN1.0201 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW1)
PERMIT SYSTEM.VSWTCHN1.0201 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW2)
PERMIT SYSTEM.VSWTCHN1.0101 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWA1)
PERMIT SYSTEM.VSWTCHN1.0101 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWA2)
PERMIT SYSTEM.VSWTCHN1.0102 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWB1)
PERMIT SYSTEM.VSWTCHN2 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW1 LNXNFW2 LNXNWA1 LNXNWA2
LNXNWB1 LNXNFS1)
PERMIT SYSTEM.VSWTCHN2.0301 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW1)
PERMIT SYSTEM.VSWTCHN2.0302 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFW2)
PERMIT SYSTEM.VSWTCHN2.0301 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWA1)
PERMIT SYSTEM.VSWTCHN2.0301 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWA2)
PERMIT SYSTEM.VSWTCHN2.0302 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNWB1)
PERMIT SYSTEM.VSWTCHN2.0301 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFS1)
PERMIT SYSTEM.VSWTCHN2.0302 CLASS(VMLAN) ACCESS(UPDATE) ID(LNXNFS1)
END

```

Figure 4-37 RACF/VM commands for a Virtual Switch

More information about how to configure RACF/VM to protect Guest LANs and Virtual Switches can be found in the publication *RACF V1R10 Security Administrator's Guide*, SC28-1340.

4.10.5 Configuring Linux

We installed SUSE Linux SLES 8 from the installation CDs, and also service pack 4 using the installation instructions found on the CDs. We selected the “Minimum System” and added these extra packages on some servers:

Web server	Packages apache, expat, and mm
File server	Packages nfs-utils and vlan

Also, the latest available kernel from SUSE Linux at the time of this writing was obtained and installed. This is package k_deflt-2.4.21-281.s390.rpm.

Note: While this installation is for testing layer 2 support, we used a layer 3 (VSWITCH type IP) network connection for the initial install. We did not investigate if layer 2 support is available in the SUSE Linux installation image.

The Linux qeth driver is configured for Layer 2 support. The lines in `/etc/chandev.conf` we added for our network are shown in Figure 4-38. See “Using OSA-Express with Linux” on page 71 for an explanation of these parameters. The parameter for Layer 2 support has been highlighted in the example.

```
noauto;qeth0,0x0600,0x0601,0x0602;add_parms,0x10,0x0600,0x0602,1ayer2
qeth1,0x0610,0x0611,0x0612;add_parms,0x10,0x0610,0x0612,1ayer2
```

Figure 4-38 Configuration file `/etc/chandev.conf`

The Linux network device configuration can be done with YaST or by creating or editing the files in `/etc/sysconfig/network`.

Note: Our test network uses only Linux servers with the 2.4 kernel, because Layer 2 support in the 2.6 kernel was not available in a Linux distribution at the time this book was written. The Layer 2 support for 2.6 kernels is documented in the book *Linux on zSeries, Device Drivers, Features, and Commands, March 23, 2005*, SC33-8281. The instructions for defining a qeth device for the 2.6 kernel has been described in “Linux 2.6 kernel” on page 73. In addition to those instructions and before the devices are placed online, enable layer 2 mode with these commands:

```
echo 1 >/sys/devices/qeth/0.0.0600/1ayer2
echo 1 >/sys/devices/qeth/0.0.0610/1ayer2
```

Then bring the device online as instructed with the command:

```
echo 1 >/sys/devices/qeth/0.0.0600/on1ine
echo 1 >/sys/devices/qeth/0.0.0610/on1ine
```

4.10.6 Setting up the file server

The file server demonstrates a trunk port connection to a Virtual Switch. We described trunk port connections to switches in “Trunk ports” on page 111. A trunk port connection from a Linux server to a VSWITCH is also possible connected to up to 4 VLANs. Linux must be configured in this case with a VLAN aware connection. On SUSE Linux, the VLAN package is required. We set up the connection manually to test it, then modified the configuration files so that it would start automatically, as described in the following paragraphs.

Manual network configuration

VLAN support is usually built as a module called 8021q.o. Use the **insmod** or **modprobe** command to load the module before attempting any operations to configure or use VLANs. The following shows how to load the module using the **modprobe** command:

```
modprobe 8021q
```

You do not need to load a module if your kernel includes the VLAN support in the kernel, rather than separately in a module.

When the module is loaded, you can see the following messages in your system log or dmesg output:

```
802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>  
All bugs added by David S. Miller <davem@redhat.com>
```

The physical interface used to connect to a VLAN must be activated prior to configuring any VLAN interfaces. For our system, the physical interface is eth0. We are only using this connection for VLAN aware traffic, so we configure a dummy IP address on it. You can use the address 0.0.0.0 as a valid dummy address (in fact, the IP configuration scripts in SLES 8 seemed to recognize what we were doing when we used 0.0.0.0, and our configuration was very easy). The command is:

```
ifconfig eth0 0.0.0.0 mtu 1492 up
```

Configure a VLAN interface

We created our VLAN virtual interface using the **vconfig** command. Apart from creating and deleting virtual interfaces, the **vconfig** command allows you to control the operation of the VLAN interfaces. For instance, you can specify the priority of packets on your VLANs, which changes the relative priority of packets for different VLANs.

The **vconfig** command has settings for the default method in which the virtual devices are named. SLES 8 specifies the format VLAN_PLUS_VID_NO_PAD in its scripts which creates interface names as "vlanxxx", where xxx is the vlan number with no leading zeros. The following commands create virtual interfaces for VLANs 301 and 302 for the eth0 interface:

```
vconfig set_name_type vlan_plus_vid_no_pad  
vconfig add eth0 301  
vconfig add eth0 302
```

We can examine the output from the **ifconfig -a** command or display the file `/proc/vlan/config` to see if the command worked. See the contents of this file in Figure 4-39 on page 139.


```
cat /proc/net/vlan/config
VLAN Dev name | VLAN ID
Name-Type: VLAN_NAME_TYPE_PLUS_VID_NO_PAD
vlan301      | 301 | eth0
vlan302      | 302 | eth0
```

Figure 4-39 Display the VLAN device configuration

Bring up the VLAN interface

The VLAN interface added using **vconfig** supports the usual Linux interface configuration commands such as **ip** and **ifconfig**. We configured the virtual interface in the same way as any other network interface on your system. The following command configures the `vlan301` virtual interface on `eth0` with the IP address `10.1.1.10`:

```
ifconfig vlan301 10.1.1.10 netmask 255.255.255.0 up
```

If the `iproute2` utility is installed on your system, you can use the **ip** command to configure the interface instead. You would use these **ip** commands to get the same result as the **ifconfig** command above:

```
ip addr add 10.1.1.10/24 dev vlan301
ip link set vlan301 up
```

Tip: The `iproute2` utility by Alexey Kuznetsov, which provides the **ip** command, allows a great degree of flexibility in configuring the Linux IP stack. Many distributors now include it by default, and have rewritten their network configuration scripts using `iproute2` instead of the traditional IP configuration commands such as **route** and **ifconfig**. In addition, `iproute2` provides the facilities to control Linux advanced routing features like multiple route tables, traffic shaping, and policy routing.

Automatic network configuration

VLAN support is a fairly recent addition to Linux. Configuring VLANs has been a manual process because the network configuration scripts provided with most distributions were not VLAN-aware. SUSE Linux has set up their network scripts, however, so as long as you can use the right names for your VLAN interfaces, the system will set up your VLAN interfaces automatically.

Restriction: In addition to the naming issue, the configuration requires that the `vconfig per_kernel` option be set to `on`. This means that VLAN numbers have scope across the kernel, rather than just per interface. If you need the ability to have the same VLAN numbers on different real interfaces representing different VLANs, you will not be able to use SUSE's configuration method and will have to create your own.

VLAN numbers should be allocated universally, so it is unlikely that this restriction will cause a problem for you.

To configure a VLAN on SLES8, create a file in the `/etc/sysconfig/network` directory called `ifcfg-vlanX`, replacing `X` with the VLAN number you are configuring.

Important: The name of the `ifcfg` file you create is critical. If the file is named incorrectly, (more accurately, if the name you pass to the `ifup` script, which has to match the last part of the `ifcfg` file name, is not correct), the SUSE Linux network configuration processing will not call the correct script to configure the interface.

In our test configuration, we need three configuration files. One is for the physical interface (`eth0`) and the other two are for the VLAN configuration. Each configuration file is shown in an example. File `ifcfg-eth0` for the physical interface is shown in Figure 4-40, and for the VLANs, file `ifcfg-vlan301` is shown in Figure 4-41 on page 141, and file `ifcfg-vlan302` is shown in Figure 4-42 on page 141.

```
BOOTPROTO="static"  
STARTMODE="onboot"  
IPADDR="0.0.0.0"  
MTU="1492"
```

Figure 4-40 File `ifcfg-eth0`

```
ETHERDEVICE="eth0"
BOOTPROTO="static"
STARTMODE="onboot"
IPADDR="10.1.1.10"
NETMASK="255.255.255.0"
NETWORK="10.1.1.0"
BROADCAST="10.1.1.255"
```

Figure 4-41 File `ifcfg-vlan301`

```
ETHERDEVICE="eth0"
BOOTPROTO="static"
STARTMODE="onboot"
IPADDR="10.1.2.10"
NETMASK="255.255.255.0"
NETWORK="10.1.2.0"
BROADCAST="10.1.2.255"
```

Figure 4-42 File `ifcfg-vlan302`

The `ETHERDEVICE` field is an additional variable for VLAN configurations. This is used to instruct `vconfig` the physical interface on which to configure the VLAN. The `STARTMODE` field works the same as for physical interfaces, allowing you to add your VLAN interfaces to your network configuration and have them activated automatically at boot-time.

When the files have been created, we can start the interfaces by issuing `ifup` commands with the interface names as the parameters. The following commands start all the interfaces:

```
ifup eth0
ifup vlan301
ifup vlan302
```

4.10.7 NFS server configuration

The purpose of this server is to be a file server for the Web servers and to demonstrate a trunk port connection to a virtual switch. In our example, we used NFS (network file system) to host the Web content and allow remote access by the Web servers. The directories shown in Figure 4-43 on page 142 were created to copy the default directory structure of an apache web server for each of our customers, A and B. The document root, base directory, for the Web server of customer A is `/www/custa`, and the document root for customer B is `/www/custb`.

```
/www
/www/custa
/www/custa/htdocs
/www/custa/cgi-bin
/www/custa/icons
/www/custa/icons/small
/www/custb
/www/custb/htdocs
/www/custb/cgi-bin
/www/custb/icons
/www/custb/icons/small
```

Figure 4-43 File server directory structure

We created the file `/etc/exports` with the contents shown in Figure 4-44 to make the document root directories available to the respective Web servers.

```
# See the exports(5) manpage for a description of the syntax of this file.
# This file contains a list of all directories that are to be exported to
# other computers via NFS (Network File System).
# This file used by rpc.nfsd and rpc.mountd. See their manpages for details
# on how make changes in this file effective.
/www/custa 10.1.1.0/255.255.255.0(ro,sync,root_squash)
/www/custb 10.1.2.0/255.255.255.0(ro,sync,root_squash)
```

Figure 4-44 File `/etc/exports`

The NFS server was started manually for testing, then configured to start automatically when Linux started. The following commands in Example 4-3 were used:

Example 4-3 Starting the NFS server

```
/etc/init.d/portmap start
/etc/init.d/nfslock start
/etc/init.d/nfsserver start
chkconfig portmap on
chkconfig nfslock on
chkconfig nfsserver on
```

4.10.8 Setting up the Web servers

The Web servers run apache to serve Web pages to the Internet. The Web content is stored on the file server and exported to the Web server using NFS. The apache configuration of each server is identical, except for the directory it

accesses on the file server. The servers for Customer A receive Web connections from a load balancer, so their network configuration is different from the Customer B server. The changes required for this set up are described as part of the Load Balancer configuration, in 4.10.10, “Setting up the load balancer” on page 145. In this section, we only show the set up of the Customer B Web server.

Because the Web content is stored on the file server, the Web server must access the file server using an NFS client. The default document root (the directory where it serves files from) for the apache Web server software in SLES 8 is /srv/www, so we mount the NFS file system at that directory. To automatically mount the file system every time Linux starts, we add a line to the /etc/fstab file. We added the following line to our fstab:

```
10.1.2.10:/www/custb /srv/www nfs ro,rsize=8192,wsiz=8192,soft 0 0
```

The nfs mount options are recommended in the man page for nfs (from the **man nfs** command.) The final steps are to configure the system to start the required services automatically, manually start the services, and mount the file system. We used the following commands:

```
chkconfig portmap on
chkconfig nfs on
/etc/init.d/portmap start
mount -a
```

The -a option on mount instructs the mount command to mount any unmounted file systems. Entering the command **mount** shows it mounted successfully, as highlighted in Figure 4-45.

```
# mount
/dev/dasdb1 on / type ext3 (rw)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/dasdc1 on /usr type ext3 (rw)
shmfs on /dev/shm type shm (rw)
10.1.2.10:/www/custb on /srv/www type nfs (ro,rsiz=8192,wsiz=8192,soft,addr=10.1.2.10)
```

Figure 4-45 Mounted filesystems

All that is left to do is start apache and also configure it to start automatically:

```
chkconfig apache on
/etc/init.d/apache start
```

4.10.9 Setting up the firewalls

Our test uses Linux servers as firewall devices. Linux has a built in packet filtering capability in the kernel that is configured using the `iptables` command. A Linux based packet filtering firewall was sufficient for our test network, but it might not be sufficient for all installations. Special purpose firewall devices are available that implement more sophisticated firewalls. This includes specialized virtual firewalls that can run on zSeries under z/VM. One goal of our test was to show that both physical and virtual firewalls can be used together in a single implementation.

A *firewall* is really both a packet filter and a router because it forwards packets from one subnet to another. After the basic network configuration is up and working, our first task is to set up the servers to forward packets correctly, and then set up the packet filter. SLES 8 and SLES 9 will enable forwarding during the boot process if the variable `IP_FORWARD` is set to `yes` in the file `/etc/sysconfig/sysctl`. After this variable is changed, we entered the following command to make it active.

```
/etc/init.d/boot.ipconfig start
```

This command is automatically run as part of the boot process, but we can run it manually to change the kernel setting for forwarding.

In order to quickly create a test firewall, we used the `SuSEfirewall2` package at level 3.1 supplied with SUSE Linux. Only a few of the parameters are changed in the configuration file to create a working firewall. The configuration file is named `/etc/sysconfig/SuSEfirewall2`. This file contains many comments to explain the parameters and give examples. Table 4-18 lists which parameters we changed and why we made the changes.

Table 4-18 *SuSEfirewall2* parameters

Parameter name and value	Comments
<code>FW_DEV_EXT="eth0"</code>	This is the network interface that connects to the untrusted network, in other words, the Internet.
<code>FW_DEV_DMZ="eth1"</code>	This is the interface that connects to a partially trusted network. This network contains our servers and limited access is allowed from the untrusted network.
<code>FW_ROUTE="yes"</code>	Configure the kernel to forward packets.
<code>FW_MASQ_DEV=""</code>	Set this variable to null because we are not using masquerading.

Parameter name and value	Comments
FW_TRUSTED_NETS="10.1.0.0/16"	Define our trusted network.
FW_FORWARD="0/0,192.168.2.0/24,tcp,80"	Forward http requests from anywhere to our web servers.

To configure the firewall at boot time, enter this command:

```
chkconfig SuSEfirewall12 on
```

Start the firewall with the following command:

```
/etc/init.d/SuSEfirewall12 start
```

The firewall logs both successful and blocked connections on the console and in the `/var/log/messages` file. Check this file to see if the firewall is working correctly.

4.10.10 Setting up the load balancer

We set up the load balancer (also called a *Director*) using the instructions found at the Linux Virtual Server (LVS) project web site at:

<http://www.linuxvirtualserver.org/>

LVS ties a group of server machines together using various network and scheduling techniques, so that they present a single server image to clients requesting a service. All traffic for the server is initially sent to a single load balancing and scheduling machine, which it then sends to back-end servers for processing.

An LVS installation implements one of three network techniques:

- ▶ Network address translation (LVS-NAT)

Packets arriving at the LVS Director (load balancer) are address-translated to the IP address of one of the back-end servers for processing by the application. When the response is generated, the director reverses the address translation so that the load balancer appears to have processed the request. Both the incoming and outgoing packets are processed by the director.

- ▶ IP tunneling (LVS-TUN)

Packets arriving at the LVS Director are “tunnelled” using IP encapsulation to a back-end server, which processes the request. The back-end server responds back to the client directly. A description of implementing this method on a zSeries Linux server can be found in the IBM Redbook *Linux on IBM @server zSeries and S/390: Distributions*, SG24-6264.

► Direct routing (LVS-DR)

Packets arriving at the LVS Director are redirected to one of the back-end boxes by changing only the MAC address on the incoming packet and forwarding it to the back-end server. The back-end server responds back to the client directly. This method requires that the director and the back-end servers are on the same LAN segment.

For our network, we set up an LVS-DR director. Prior to the Layer 2 support in the OSA-Express and the Virtual Switch, it was not possible to use the Direct Routing method for load balancing on a zSeries server. SUSE Linux includes the support necessary for load balancing in their kernel and in their distribution. The additional packages required on the director are `ipvsadm` and `mon`. The Web servers do not require any additional packages; the support is already included in the kernel.

LVS direct routing

The director requires an extra IP address assigned to it which is called the *virtual IP* (VIP.) The back-end server that actually processes a request is called a *real server*. The IP address of a real server is called the *real IP* (RIP.) A director works by accepting a connection to its VIP for a defined service, such as port 80 for a Web server, and forwarding that request to a real server to be processed. If there are multiple real servers, then the director must use some kind of selection algorithm to select which server to send the request to. The real server must be configured to accept a request sent to the VIP and after processing it sends the response directly back to the client.

LVS-DR implements this by accepting the connection request, selecting a real server, changing the destination MAC address of the Ethernet frame to one of the real servers, and then sending that frame with no other modifications to that server. This is a very efficient operation, because very little processing of the received frame is done by the director. The real server accepts the request as though it was sent directly to it and processes it as normal.

In order for a real server to accept the request forwarded from the director, it must have the VIP configured as an alias IP on an interface. But, we do not want the real server to respond to ARP requests for the VIP; the director should do that. So we create the VIP as an alias on the loopback interface and instruct the Linux kernel to hide the VIP by never responding to an ARP request.

LVS Scheduling algorithms

As we write this book, the LVS project has defined 10 scheduling algorithms for load balancing, most of which are variations on three primary selections:

- ▶ Round-Robin Scheduling

This algorithm directs the connection to a different server in a round-robin manner. All servers are considered equal. A variation on this algorithm assigns a scheduling weight to each server.

- ▶ Least-Connection Scheduling

This algorithm directs the connection to the server with the least number of connections. It must keep track of active connections for each real server. There are variations of this algorithm that assign weights to each server or does processing based on the network delays to each server.

- ▶ Hashing

The server is selected based on a hash table lookup based on the source or destination IP address.

In our example, we use round-robin scheduling.

Setting up the Web servers

Connection requests from users are addressed to the VIP on the director which in turn are redirected to the real servers, our Web servers. Therefore, the Web servers must have an interface configured with the VIP address so that they will accept packets addressed to the VIP. But we cannot just assign the VIP as an alias IP on the primary interface, because the Web server would attempt to respond to ARP requests that must only be responded to by the director server. Therefore, we must configure Linux to hide the VIP from ARP requests. This is done by using the loopback interface and setting some parameters in the kernel.

The alias IP is put on the loopback interface so that we do not prevent the primary interface from responding to ARP requests. We created an interface named `lo:1` and configured it with a file named `/etc/sysconfig/network/ifcfg-lo:1` with this content:

```
BOOTPROTO="static"  
STARTMODE="onboot"  
IPADDR="192.168.1.2"  
NETMASK="255.255.255.255"  
POST_UP_SCRIPT=hidearp
```

Note: The documentation for the `ifcfg` configuration file variables is found in the `ifcfg` man page. Enter `man 8 ifcfg` to read it.

Two lines have been emphasized. The `IPADDR` is the VIP of the director. The `POST_UP_SCRIPT` is a script name found in the directory `/etc/sysconfig/network/scripts` that is called after the interface is activated. In the

script, we set kernel parameters to hide the loopback interface from ARP requests. Here is the contents of this script:

```
echo 1 >/proc/sys/net/ipv4/conf/all/hidden
echo 1 >/proc/sys/net/ipv4/conf/lo/hidden
```

After creating this script, be sure to make it executable.

To test the setup, enter this command:

```
ifup lo:1
```

See Figure 4-46 for a listing of the commands issued on our system to verify the set up.

```
# ifconfig lo:1
lo:1      Link encap:Local Loopback
          inet addr:192.168.1.2  Mask:255.255.255.255
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
# cat /proc/sys/net/ipv4/conf/all/hidden
1
# cat /proc/sys/net/ipv4/conf/lo/hidden
1
```

Figure 4-46 Verification of the web server virtual interface

Setting up the Director

On this Linux server we set up the VIP as an alias on the primary interface. We do want this server to respond to ARP requests for the VIP address. In SUSE Linux SLES 9, both the primary and alias IP address can be configured in one file. Example 4-4 is the contents of our `/etc/sysconfig/network/ifcfg-eth0` file:

Example 4-4 The `/etc/sysconfig/network/ifcfg-eth0` file

```
BOOTPROTO='static'
BROADCAST='192.168.1.255'
IPADDR='192.168.1.1'
MTU='1492'
NETMASK='255.255.255.0'
NETWORK='192.168.1.0'
REMOTE_IPADDR=''
STARTMODE='onboot'
IPADDR_0='192.168.1.2'
NETMASK_0='255.255.255.0'
```

The variable `IPADDR` defines the address of the primary interface. The variable `IPADDR_0` defines the IP address of the VIP address.

The **ipvsadm** command is the command line interface to the kernel based load balancing code. SUSE Linux supplies this module and scripts to save and restore the configuration. We manually entered our configuration the first time and then saved it so that it was set up every time Linux started.

Table 4-19 explains the options of the **ipvsadm** command used in our configuration.

Table 4-19 Options for the ipvsadm command

Option	Comment
-A	Add a new service
-a	Add a real server to the configuration
-s	Specify the scheduler to use. We used "rr" for Round Robin
-t	Specify a real server and port as host:port
-g	Specify the method as LVS-DR (also called gatewaying)
-w	Specify the weight of a real server. We did not use a weighted scheduler, so this is ignored in our configuration.

The commands we entered are in Example 4-5:

Example 4-5 Using the ipvsadm command

```
ipvsadm -A -t 192.168.1.2:80 -s rr
ipvsadm -a -t 192.168.1.2:80 -r 192.168.1.3:80 -g -w 1
ipvsadm -a -t 192.168.1.2:80 -r 192.168.1.4:80 -g -w 1
/etc/init.d/ipvsadm save
chkconfig ipvsadm on
```

The last two commands save our configuration as file `/etc/ipvsadm.rules` (Example 4-6) and configure the rules to be automatically loaded when Linux starts. We checked our settings by entering the **ipvsadm** command without any arguments:

Example 4-6 Saving the /etc/ipvsadm.rules

```
# ipvsadm
IP Virtual Server version 1.2.0 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  192.168.1.2:http  rr
  -> 192.168.1.4:http             Route   1      0          0
  -> 192.168.1.3:http             Route   1      0          0
```

4.10.11 Testing

The first step we did in our testing was to verify our setup of z/VM, starting with the VSWITCH controller virtual machines. Here is our query in Example 4-7:

Example 4-7 Verifying z/VM setup

```
q controller
Controller VSWCTL1 Available: YES VDEV Range: * Level 510
Capability: IP ETHERNET VLAN_ARP
SYSTEM VSWTCHN1 Primary Controller: * VDEV: 2D40
SYSTEM VSWTCHN2 Backup Controller: * VDEV: 2D43
Controller VSWCTL2 Available: YES VDEV Range: * Level 510
Capability: IP ETHERNET VLAN_ARP
SYSTEM VSWTCHN2 Primary Controller: * VDEV: 2E40
SYSTEM VSWTCHN1 Backup Controller: * VDEV: 2E43
```

If your Virtual Switches have not been defined yet, your output will be different. Next, we queried our Virtual Switches, showing the definition of each switch and its access list (Example 4-8):

Example 4-8 Querying the Virtual Switches

```
q vswitch accesslist
VSWITCH SYSTEM VSWTCHN1 Type: VSWITCH Connected: 6 Maxconn: INFINITE
PERSISTENT RESTRICTED ETHERNET Accounting: OFF
VLAN Aware Default VLAN: 0001 Default Porttype: Access
State: Ready
QueueStorage: 8
Portname: UNASSIGNED RDEV: 2D40 Controller: VSWCTL1 VDEV: 2D40
Portname: UNASSIGNED RDEV: 2E43 Controller: VSWCTL2 VDEV: 2E43 BACKUP
Authorized userids:
LNXF1W1 VLAN: 0201 Porttype: Access
LNXF1W2 VLAN: 0201 Porttype: Access
LNXNWA1 VLAN: 0101 Porttype: Access
LNXNWA2 VLAN: 0101 Porttype: Access
LNXNWB1 VLAN: 0102 Porttype: Access
SYSTEM VLAN: 0001 Porttype: Access
VSWITCH SYSTEM VSWTCHN2 Type: VSWITCH Connected: 6 Maxconn: INFINITE
PERSISTENT RESTRICTED ETHERNET Accounting: OFF
VLAN Aware Default VLAN: 0001 Default Porttype: Access
State: Ready
QueueStorage: 8
Portname: UNASSIGNED RDEV: 2E40 Controller: VSWCTL2 VDEV: 2E40
Portname: UNASSIGNED RDEV: 2D43 Controller: VSWCTL1 VDEV: 2D43 BACKUP
Authorized userids:
LNXF1S1 VLAN: 0301 0302 Porttype: Trunk
LNXF1W1 VLAN: 0301 Porttype: Access
LNXF1W2 VLAN: 0302 Porttype: Access
```

LNXNWA1	VLAN: 0301	Porttype: Access
LNXNWA2	VLAN: 0301	Porttype: Access
LNXNWB1	VLAN: 0302	Porttype: Access
SYSTEM	VLAN: 0001	Porttype: Access

Alternative commands are **QUERY VSWITCH** and **QUERY VSWITCH DETAILS**. The **DETAILS** operand displays information about every connection to the **VSWITCH**. The command with no options just displays a summary of the switch definition. If your system is using **RACF/VM** to control access to the **VSWITCH**, the **ACCESSLIST** operand will not show any users in the access list. You have to use the **RACF RLIST** command to display the access authorizations.

All the Linux servers are brought up and the connections between them verified using the **ping** command. We found errors with mistyped IP addresses and incorrect VLAN assignments that caused failures. After all the connections were verified, we checked the routing through the firewalls to see if packets could make the complete round trip. Then the firewall code was started and checks were done to see if it was passing through the correct packets and blocking the others. This verified that the basic configuration was correct.

At this point, we were ready to create some simulated network traffic and see how it reacted to errors. A simple script was created to use the **w3m** command to fetch Web pages in a loop to create simulated traffic. The sample script is shown in Figure 4-47. It retrieves the Web page, prefixes each output line with a time stamp, and sends the line to standard out. The output of this script can be logged to check for any gaps when the Web page could not be retrieved. We also opened ssh sessions to the servers with the administrative IP addresses and ran commands such as **vmstat 1** to generate regular output. And finally, we started ping to various IP addresses.

```
#!/bin/bash
# Fetch a web page over and over
while [ 1 = 1 ]; do
    w3m -dump $1 2>&1 | awk '{print strftime("%T"), $0}'
    sleep 1s
done
```

Figure 4-47 Fetch web pages in a loop

With all these tests running, we introduced errors. Here is a list of ideas we had for errors:

1. Stop each Customer A Web server to force the load balancer to fail over.
2. Unplug the network cable from each OSA port.
3. Do a forced logoff of each **VSWITCH** controller user id.
4. Detach OSA device from the **VSWITCH** controller user id.

5. Deactivate each port on the Cisco switch.
6. Reboot Linux servers and check that all connections come back up.

In each case, the system recovered, usually with a brief outage. Some connections in some of the tests did not drop, but the connection might have briefly experienced a slow response. Whether or not there was an outage depended on the connection (an established connection or an attempt to create a new one), the direction of the connection (ingoing or outgoing), and how severe the failure was (for example, the reboot of Linux took longer to recover.) We were impressed at how well the network connections recovered during our testing.

4.10.12 Recommendations

VSWITCH is a very powerful virtualization solution. It removes the requirement for an intermediate router between an external LAN and the internal Guest LAN. This can save significant amounts of CPU cycles if your environment is moderately to heavily used. VSWITCH supports the IEEE 802.1Q VLAN standards, which are important to many organizations. The failover capabilities of VSWITCH are very impressive. With only a minimal amount of effort, it is possible to configure a highly available networking environment.

VSWITCH provides significant benefits over the older point-to-point technologies and has more function than the base Guest LAN implementation. For communications within a single z/VM environment, VSWITCH is a very good solution.

The Layer 2 Switching support available as part of z/VM V5.1 VSWITCH, is a very powerful solution that builds on earlier z/VM LAN technology. Running as a Layer 2 Switch means that Linux guests can exist and operate in the network in exactly the same way as physical machines. Advanced networks can be created and supported with interactions of Linux servers both inside and outside of your zSeries server. Non-IP protocols, such as SNA, IPX™, NetBIOS, and DECnet, are now supported, as is IPv6, which means that there are more opportunities to consolidate machines to a z/VM environment.

We recommend that you run VSWITCH running in Layer 2 mode if you have a requirement to run multiple Linux systems in a z/VM environment. This option is the current state of the art network virtualization technology for z/VM.

4.11 Summary

Networking options for Linux on zSeries have matured significantly since Linux first became available on the mainframe in December 1999. From a hardware perspective over that period, we started with OSA-2 cards, many running at just

10 Mbps. Since then, we have seen the introduction of OSA-Express 2 10 Gigabit Ethernet Long Reach, OSA-Express Gigabit Ethernet, 1000BASE-T, and HiperSockets.

We have seen perhaps even greater advances in networking technology in the virtualized world of z/VM. Originally, apart from a direct connection to a physical interface, the only methods of connecting Linux guests to each other or to an external network were through IUCV or virtual channel to channel adapters. These point-to-point technologies were cumbersome, prone to error, and alien concepts to Linux administrators used to an Ethernet world.

With the introduction of z/VM Guest LAN technology, and in particular the latest Layer 2 Virtual Switch solution, z/VM now has a network virtualization capability that is unparalleled in the industry.

We recommend that you use OSA-Express2, OSA-Express Gigabit Ethernet, or 1000BASE-T for your machine's physical network interfaces. For internal communications, it really depends on what systems are being connected together. For LPAR-to-LPAR communications, we recommend HiperSockets. When multiple Linux guests within a z/VM environment need to communicate with each other, a HiperSockets Guest LAN is one of the best choices. When multiple Linux guests need to communicate within a z/VM LPAR and also with the external network, we recommend that you use a Virtual Switch running in Layer 2 Switching mode.

Archived

FCP-attached SCSI disks

In this chapter, we discuss how to configure and use SCSI disks accessed using Fibre Channel Protocol (FCP) in the Linux 2.6 kernel on zSeries. Topics include:

- ▶ FCP configuration using sysfs
- ▶ SCSI device configuration files
- ▶ Persistence SCSI disk naming with udev
- ▶ The FCP LUN Access Control feature
- ▶ Multipathing for SCSI disks

Note: For an introduction to FCP on Linux for zSeries, see *Linux on zSeries and S/390: Fibre Channel Protocol Implementation Guide*, SG24-6344.

5.1 FCP configuration using sysfs

Configuration of FCP-attached devices has changed significantly in the Linux 2.6 kernel. The `/proc/scsi/zfcplib/map` interface used in the Linux 2.4 kernel is no longer available or required. Instead, the Linux 2.6 kernel uses `sysfs` to map FCP device information to the SCSI subsystem. The SCSI device Worldwide Port Name (WWPN) and Logical Unit Number (LUN) values exist as attributes of the FCP device in `sysfs`.

Note: For details on FCP device mapping, see *Linux on zSeries and S/390: Fibre Channel Protocol Implementation Guide*, SG24-6344.

Figure 5-1 illustrates the `sysfs` directory structure for an FCP device at address B002.

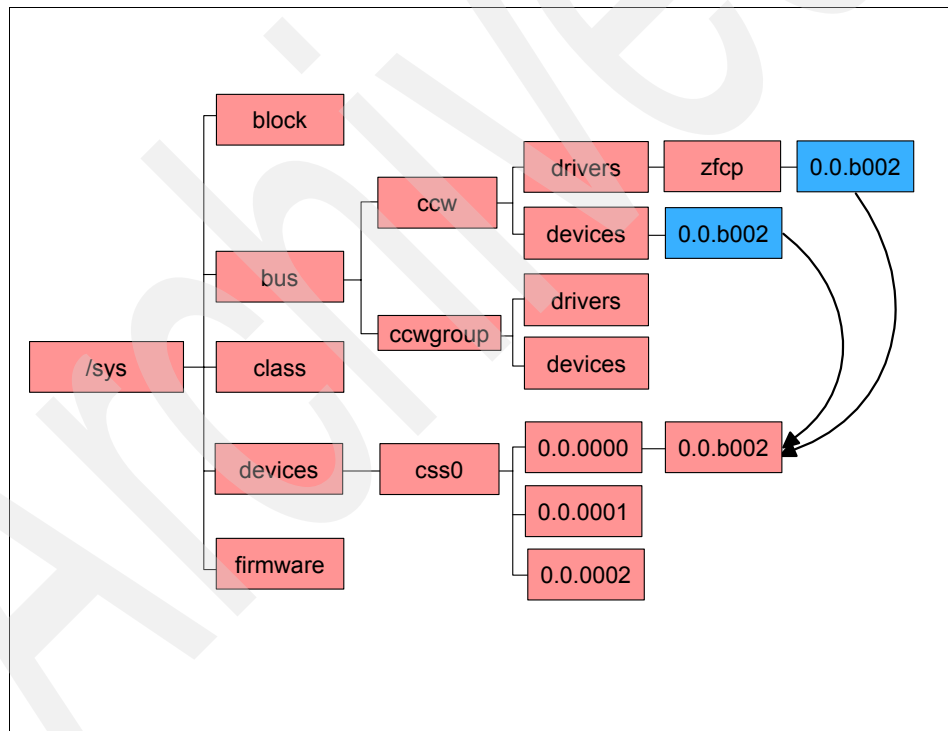


Figure 5-1 Portions of `sysfs` relevant to the `zfcplib` driver

Dynamically adding and removing SCSI disks is simplified in the Linux 2.6 kernel, in contrast to the Linux 2.4 kernel, using the `sysfs` filesystem. The following examples illustrate the procedure:

Note: The zfcplib device driver must be loaded *before* FCP device entries appear in sysfs. To load the driver, issue the **modprobe** command:

```
modprobe zfcplib
```

Entries in /sys directory allow us to configure a SCSI device. In these examples, we assume the current directory is /sys/bus/ccw/drivers/zfcplib.

1. Set the virtual FCP adapter online. To bring FCP device B002 online:

```
# echo 1 > 0.0.b002/online
```

To bring the device offline:

```
# echo 0 > 0.0.b002/online
```

As an alternative, use the **chccwdev** command:

```
# chccwdev -e 0.0.b002
Setting device 0.0.b002 online
Done
# chccwdev -d 0.0.b002
Setting device 0.0.b002 offline
Done
```

2. Add a WWPN. To add WWPN 0x5005076300cd9589 to the FCP device:

```
# echo 0x5005076300cd9589 > 0.0.b002/port_add
```

The WWPN appears as a subdirectory for the device (the 0.0.b002/0x5005076300cd9589 subdirectory).

3. Add a Logical Unit (LUN) to the WWPN . To add LUN 0x520b000000000000 to the WWPN:

```
# echo 0x520b000000000000 > 0.0.b002/0x5005076300cd9589/unit_add
```

The LUN is now available to Linux. As this is the first SCSI disk, it maps to the sda device. This can be checked using:

```
# cat /sys/block/sda/device/fcp_lun
0x520b000000000000
```

When added, the LUN is mapped to the SCSI subsystem in sysfs. A subdirectory of the form *host:bus:target:lun* is added to the /sys/bus/scsi/devices directory:

```
# cat /sys/bus/scsi/devices/0:0:1:2/fcp_lun
0x5209000000000000
```

5.1.1 Querying FCP device information

Information for an FCP device is available in sysfs. For example, attributes for the device B002 device can be read from files found in the `/sys/bus/ccw/drivers/zfc/0.0.b002` directory.

Note: A complete list of all attributes for an FCP device can be found in *Linux on zSeries Device Drivers, Features, and Commands March 23, 2005, SC33-8281*.

Both RHEL4 and SLES9 still maintain the `/proc/scsi/scsi` file interface (Example 5-1):

Example 5-1 File interface in RHEL4 and SLES9

```
# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 01 Lun: 00
  Vendor: IBM      Model: 2105800      Rev: 1.62
  Type:   Direct-Access      ANSI SCSI revision: 03
Host: scsi0 Channel: 00 Id: 01 Lun: 01
  Vendor: IBM      Model: 2105800      Rev: 1.62
  Type:   Direct-Access      ANSI SCSI revision: 03
```

SLES9 provides several additional commands to query SCSI devices:

- ▶ The `lsscsi` command uses sysfs to report information about SCSI devices. The `-g` option lists the generic SCSI device name for each SCSI disk (Example 5-2):

Example 5-2 Using the sscsi command

```
lsscsi -g
[0:0:1:0] disk IBM 2105800 1.62 /dev/sda /dev/sg0
[0:0:1:1] disk IBM 2105800 1.62 /dev/sdb /dev/sg1
[0:0:1:2] disk IBM 2105800 1.62 /dev/sdc /dev/sg2
[0:0:1:3] disk IBM 2105800 1.62 /dev/sdd /dev/sg3
[0:0:1:4] disk IBM 2105800 1.62 /dev/sde /dev/sg4
```

Note: The `sg` device driver must be loaded to obtain generic SCSI device names. If not loaded, the reported generic name appears as “-”.

- ▶ The `systemd` command uses sysfs to report devices by bus, class, and topology. The `-v` option reports all device attributes; the `-b` option reports device information for a specific bus (Example 5-3 on page 159):

Example 5-3 Using the systool command

```
# systool -vb scsi
Bus = "scsi"

Device = "0:0:1:0"
Device path = "/sys/devices/css0/0.0.0013/0.0.b002/host0/0:0:1:0"
delete = <store method only>
detach_state = "0"
device_blocked = "0"
fcplun = "0x5206000000000000"
hba_id = "0.0.b002"
model = "2105800"
online = "1"
queue_depth = "32"
rescan = <store method only>
rev = "1.62"
scsi_level = "4"
timeout = "30"
type = "0"
vendor = "IBM"
wwpn = "0x5005076300cd9589"
```

- ▶ The **udevinfo** command reports device information from udev (Example 5-4):

Example 5-4 Using the udevinfo command

```
# udevinfo -q all -n /dev/sda1
P: /block/sda/sda1
N: sda1
M: 0660
S: disk/by-path/ccw-0.0.b002-zfcplun-0x5005076300cd9589:0x5206000000000000p1
disk/by-id/1IBM_2105_20622513p1
O: root
G: disk
```

The `-q all` option specifies all sysfs device attributes are to be reported; the `-n /dev/sda1` option displays disk and partition information for the `/dev/sda1` device.

5.2 SCSI device configuration files

SCSI device configurations defined using sysfs are lost at the next system IPL. To define devices that persist across IPLs, the Linux 2.6 kernel uses configuration files executed early in system boot. FCP device configuration files differ between SLES9 and RHEL4.

5.2.1 FCP device configuration in SLES9

FCP device configuration files for SLES9 are found in the `/etc/sysconfig/hardware` directory. A unique configuration file exists for each device to be configured at system boot. FCP configuration file names take the form:

```
hwcfg-zfcplib-bus-ccw-0.0.xxxx
```

The `xxxx` value is the device number of the FCP subchannel.

Note: A skeletal FCP configuration is provided as the `/etc/sysconfig/hardware/skel/hwcfg-zfcplib` file.

A sample configuration file for an FCP device at address B002 is shown in Figure 5-2.

```
#!/bin/sh
#
# hwcfg-zfcplib-bus-ccw-0.0.b002
#
# Configuration for the zfcplib adapter at CCW ID 0.0.b002
#
STARTMODE="auto"
MODULE="zfcplib"
MODULE_OPTIONS=""
MODULE_UNLOAD="yes"

# Scripts to be called for the various events.
# If called manually the event is set to 'up'.
SCRIPTUP="hwup-ccw"
SCRIPTUP_ccw="hwup-ccw"
SCRIPTUP_scsi_host="hwup-zfcplib"
SCRIPTDOWN="hwdown-scsi"
SCRIPTDOWN_scsi="hwdown-zfcplib"
ZFCPLIB_LUNS="0x5005076300cd9589:0x5206000000000000 0x5005076300cd9589:0x5207000000000000"
```

Figure 5-2 The `/etc/sysconfig/hardware/hwcfg-zfcplib-bus-ccw-0.0.b002` configuration file

SCSI device WWPN and LUN values are defined using the `ZFCPLIB_LUNS` parameter (in the form `WWPN:LUN`). Multiple SCSI devices can be defined separated by a space.

Note: New WWPN and LUN values can be added to the configuration file at any time. The new SCSI device will be available automatically after reboot, provided the system is *not* booted from the new device. To boot from a new SCSI device, it is necessary to install a boot loader (using the **zip1** command) on the disk. If the root filesystem resides on the disk, a new initial ramdisk must also be created (using the **mkinitrd** command). For details on how to create a new initial ramdisk and install a boot loader, consult *Cloning FCP-attached SCSI SLES9 Linux*, REDP3871.

Manually configuring an FCP device

To manually configure a new FCP device:

1. Copy a sample FCP device configuration file to the `/etc/sysconfig/hardware` directory. Use an existing configuration file, or the provided FCP skeletal file. Name the new file appropriately (for instance, `hwcfg-zfcp-bus-ccw-0.0.1234` for an FCP device at address 1234).
2. Provide the specific WWPN and LUN values as `ZFCP_LUN` parameter.
3. Execute the **hwup** command, providing the name of the new configuration file as a parameter:

```
# hwup zfcp-bus-ccw-0.0.1234
```

This configures each SCSI device defined in the configuration file.

Note: The **chccwdev** command (used to bring the device online and offline) automatically looks for the appropriate device configuration file located in the `/etc/sysconfig/hardware` directory. If the system is booted from SCSI disk, the disk cannot be taken offline using the **chccwdev** command. The following message appears on the system console:

```
scsi0 (1:0): rejecting I/O to device being removed
```

Configuring an FCP device using YaST

The version of YaST provided with SLES9 supports FCP-attached SCSI devices. With YaST, you can add, remove, and configure SCSI disks. The add SCSI disk dialog is shown in Figure 5-3 on page 162.

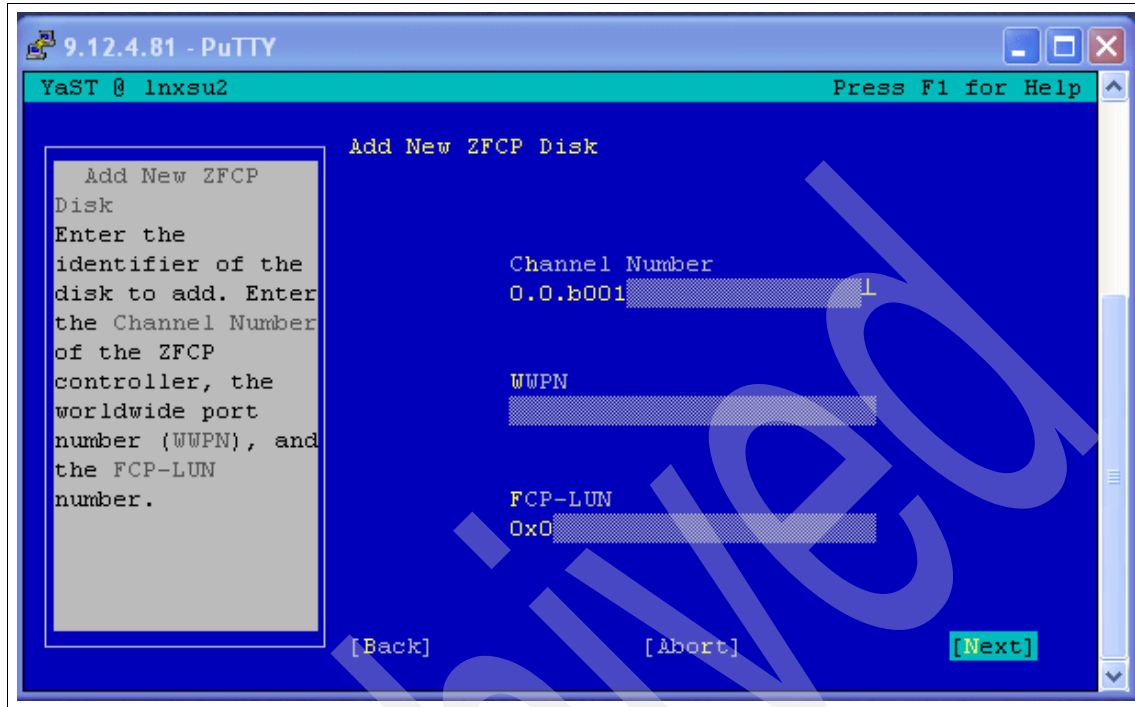


Figure 5-3 YaST dialog to add SCSI disk

The appropriate hardware configuration file for the FCP device is automatically created in the `/etc/sys/hardware` directory. For more details on using YaST to manage SCSI disks, consult *Linux on zSeries and S/390: Fibre Channel Protocol Implementation Guide*, SG24-6344.

5.2.2 FCP device configuration in RHEL4

As with RHEL3, RHEL4 maps WWPNs and LUNs using the `/etc/zfc.conf` file to. However, the file format has improved since RHEL3; entries can now span multiple lines, as shown in Figure 5-4 on page 163.


```
# cat /etc/zfcp.conf
0.0.b020 0x01 0x5005076300cd9589 0x00 0x5213000000000000
0.0.b020 0x01 0x5005076300cd9589 0x01 0x5212000000000000
# cat /etc/modprobe.conf
alias eth0 qeth
options dasd_mod dasd=201,202
alias scsi_hostadapter zfcp
```

Figure 5-4 FCP configuration files for RHEL4

During system boot, RHEL4 executes the `/sbin/zfcpconf.sh` script, which, in turn, loads the `zfcp` device driver and executes the `/etc/zfcp.conf` file.

Note: With RHEL4, the `alias scsi_hostadapter zfcp` statement *must* be included in the `/etc/modprobe.conf` file.

Device mapping is specified in the general format used by RHEL3. However, the FCP device number is now specified in the form:

```
0.0.devno
```

The leading zeros indicate the host and bus numbers (always zero in both case); the *devno* portion is the actual FCP device address. For details on how to map FCP devices, see *Linux on zSeries and S/390: Fibre Channel Protocol Implementation Guide*, SG24-6344.

To automatically configure a new SCSI device under RHEL3, a new initial ramdisk must be created. The boot loader must also be reinstalled.

Creating a new initial ramdisk

The FCP mapping specified in the `/etc/zfcp.conf` file determines how LUNs are assigned to SCSI devices (the first LUN entry is assigned to `/dev/sda`, the second to `/dev/sdb`, and so on). To create a new initial ramdisk using a new SCSI device mapping, issue the commands shown in Figure 5-5 on page 164.

```

# cd /boot
# mv initrd-2.6.9-6.37.EL.img initrd-2.6.9-6.37.EL.img.old
# mkinitrd -v --with=scsi_mod --with=zfcplib --with=sd_mod initrd-2.6.9-6.37.EL \
2.6.9-6.37.EL
.
.
Loading module scsi_mod
Loading module sd_mod
Loading module scsi_transport_fc
Loading module zfcplib
.
.
.

```

Figure 5-5 Creating an initial ramdisk

Note: The initial ramdisk is created in the /boot directory. We first save the original initial ramdisk (rather than overwrite it).

Reinstall the boot loader

After a new initial ramdisk is created, we reinstall the boot loader (pointing to the new initial ramdisk) using the `zipl` command. Rather than provide command line parameters, we use the `/etc/zipl.conf` configuration file shown in Figure 5-6).

```

[defaultboot]
default=linux
target=/boot/
[linux]
image=/boot/vmlinuz-2.6.9-6.37.EL
ramdisk=/boot/initrd-2.6.9-6.37.EL.img
parameters="root=LABEL="

```

Figure 5-6 The `/etc/zipl.conf` file

5.3 Persistence SCSI disk naming with udev

At system boot in the Linux 2.4 kernel, SCSI disks are mapped to device node names according to the order in which the LUNs are mapped. The mapping order is typically determined by the address of the FCP device and the accessed LUN. For instance, the first LUN mapped to the first FCP device is assigned to device node name `/dev/sda`, the second to `/dev/sdb`, and so on.

This can lead to a unpredictable situation when new SCSI disks are added to the system. For example, consider a Linux server with SCSI disks accessed using an FCP device at address 0x0010 and mapped to device node /dev/sda. Adding additional SCSI disks accessed by an FCP device at address 0x000A causes this device to be mapped to /dev/sda on system reboot (the device at address 0x0010 is then mapped to /dev/sdb).

In the Linux 2.6 kernel, udev provides a predictable mechanism to map SCSI disks to their respective device nodes. This improves system reliability; after system reboot (or in the event of channel path failure and recovery), SCSI device node assignments remain unchanged. Both RHEL4 and SLES9 support persistent udev device naming rules.

5.3.1 Persistent SCSI disk naming in RHEL4

With RHEL4, udev device naming rules are specified in the files found in the /etc/udev/rules.d directory. Files are read in lexical order and determine the udev device naming conventions. By default, two rule files are provided (10-udev.rules and 50-udev.rules).

Several options are available to define persistent device naming for SCSI disks. To illustrate, we show an example that uses symbolic links composed of the SCSI disk WWPN and LUN. Add the following lines to the /etc/udev/rules.d/50-udev.rules file:

```
KERNEL="sd*[a-z]", SYMLINK="scsi/%s{fcplun}-%s{wwpn}/disk"  
KERNEL="sd*[0-9]", SYMLINK="scsi/%s{fcplun}-%s{wwpn}/part%n"
```

These lines use regular expressions to create symbolic links in the /dev directory. The symbolic links are created using the WWPN (the %s{wwpn} expression) and LUN (the %s{fcplun} expression) as part of the name. Because the WWPN and LUN uniquely identify a SCSI device, the symbolic link deterministically identifies the SCSI device.

The rules create a unique subdirectory in /dev/scsi for each SCSI disk detected by a hotplug event. As shown Figure 5-7 on page 166, the symbolic link name is uniquely identified by the device LUN and WWPN.

```
# ls -lah /dev/scsi
total 0
drwxr-xr-x  5 root root  100 Apr 15 18:48 .
drwxr-xr-x 10 root root 2.9K Apr 15 18:48 ..
drwxr-xr-x  2 root root   80 Apr 15 18:40 0x5211000000000000-0x5005076300cd9589
drwxr-xr-x  2 root root   80 Apr 15 18:48 0x5212000000000000-0x5005076300cd9589
drwxr-xr-x  2 root root   80 Apr 15 18:48 0x5213000000000000-0x5005076300cd9589
```

Figure 5-7 Directories created by udev to map SCSI disks

As shown in Figure 5-8, each subdirectory in turn contains symbolic links to the familiar device node name (/dev/sda for instance).

```
# ls -lah /dev/scsi/0x5211000000000000-0x5005076300cd9589
total 0
drwxr-xr-x  2 root root   80 Apr 15 18:40 .
drwxr-xr-x  5 root root  100 Apr 15 18:48 ..
lrwxrwxrwx  1 root root   9 Apr 15 18:40 disk -> ../../sda
lrwxrwxrwx  1 root root  10 Apr 15 18:40 part1 -> ../../sda1
lrwxrwxrwx  1 root root  10 Apr 15 18:40 part2 -> ../../sda2
```

Figure 5-8 Symbolic links created by udev to uniquely identify SCSI disks

The disk symbolic link points to the SCSI device (/dev/sda). The part1 symbolic link points to the first partition on the SCSI disk (/dev/sda1); part2 points to the second partition (/dev/sda2).

Using the generated udev names, we can identify each device *regardless of the order in which the device is brought online*. For instance, to modify the partition table on the SCSI disk, use:

```
# fdisk /dev/scsi/0x5211000000000000-0x5005076300cd9589/disk
```

In this example, udev name corresponds to /dev/sda because it was the first SCSI disk brought online. If the device is brought offline then brought back online, it could be assigned to another device name (/dev/sdd for instance). In this case however, the disk (and all its partitions) can still be referred to by its udev name (composed from the WWPN and LUN).

Selecting the correct attributes to ensure unique device identification is critical to writing udev rules. Use output of the `udevinfo` command to identify these attributes.

```
BUS="scsi*", SYSFS{fcplun}="0x5213000000000000",
SYSFS{wwpn}="0x5005076300cd9589", SYMLINK="sda_simlink"
```

Note: We recommend changing the device SYMLINK name (*never* changing the KERNEL name) when writing custom udev rules.

5.3.2 Persistent SCSI disk naming in SLES9

SLES9 provides two start-up scripts (`/etc/rc.d/boot.scsidev` and `/etc/init.d/boot.udev`) to establish persistent device naming.

Note: Although SLES9 supports the `/etc/rc.d/boot.scsidev` startup script, the `/etc/init.d/boot.udev` is the preferred mechanism to create persistent device names.

To execute `/etc/init.d/boot.udev` during at system boot process, enable the script using the `insserv` command:

```
# insserv boot.udev
```

When enabled, udev creates two persistent symbolic links for each detected SCSI disk (this also occurs automatically each time a device is added or removed from a running system). Each symbolic is composed from the SCSI device WWPN and LUN. Symbolic links are added to the `/dev/disk/by-path` directory, and have the following format:

```
ccw-0.0.b008-zfcp-0x5005076300cd9589:0x5206000000000000  
ccw-0.0.b008-zfcp-0x5005076300cd9589:0x5206000000000000p1
```

The `/dev/disk/by-id` directory contains symbolic links composed from the device vendor identification (gathered from the device itself). These have the form:

```
1IBM_2105_20722513  
1IBM_2105_20722513p1
```

Note: Because vendor device registration is voluntarily, there is no guarantee the vendor identification is unique. Therefore, we recommend using the symbolic links in the `/dev/disk/by-path` directory. In addition the, the symbolic link in `/dev/disk/by-path` use meaningful names based on device number, WWPN, and LUN (allowing easy identification of the physical device).

Writing custom udev rules

It is possible to define a custom set of udev naming rules (similar to the method used in RHEL4). In SLES9, udev rules are applied to the `/etc/udev/udev.rules` file.

Using the method described in 5.3.1, “Persistent SCSI disk naming in RHEL4” on page 165, we add the following lines to the `/etc/udev/udev.rule` file:

```
KERNEL="sd*[a-z]", SYMLINK="scsi/%s{fcp_lun}-%s{wwpn}/disk"  
KERNEL="sd*[0-9]", SYMLINK="scsi/%s{fcp_lun}-%s{wwpn}/part%n"
```

Note: The SYMLINK parameter uses /dev/scsi directory because the /etc/rc.d/boot.scsidev startup script uses the /dev/scsi directory by default.

Figure 5-9 shows the entries in the /dev/scsi directory created by our custom udev naming rules.

```
# ls -lah /dev/scsia  
total 0  
drwxr-xr-x 2 root root 80 Apr 19 08:34 .  
drwxr-xr-x 5 root root 100 Apr 19 08:30 ..  
drwxr-xr-x 2 root root 4096 Apr 19 08:34 0x5206000000000000-0x5005b076300cd9589  
drwxr-xr-x 2 root root 4096 Apr 19 08:34 0x5207000000000000-0x5005b076300cd9589  
drwxr-xr-x 2 root root 4096 Apr 19 08:34 0x5208000000000000-0x5005b076300cd9589  
drwxr-xr-x 2 root root 4096 Apr 19 08:21 0x520a000000000000-0x5005b076300cd9589
```

Figure 5-9 The /dev/scsia directory created by udev to map SCSI disks

5.4 The FCP LUN Access Control feature

Mechanisms exist to control access to devices in a SAN fabric:

- ▶ *Zoning* is defined at the SAN switch, zoning determines whether connectivity between two WWPNs is permitted.
- ▶ *LUN masking* is defined at the storage controller, LUN masking determines if a WWPN can access a LUN.

When multiple Linux servers on zSeries share an FCP channel, by default each server has connectivity to all devices in the FCP fabric defined to the channel.

Figure 5-10 on page 169 illustrates several Linux servers sharing a single channel.

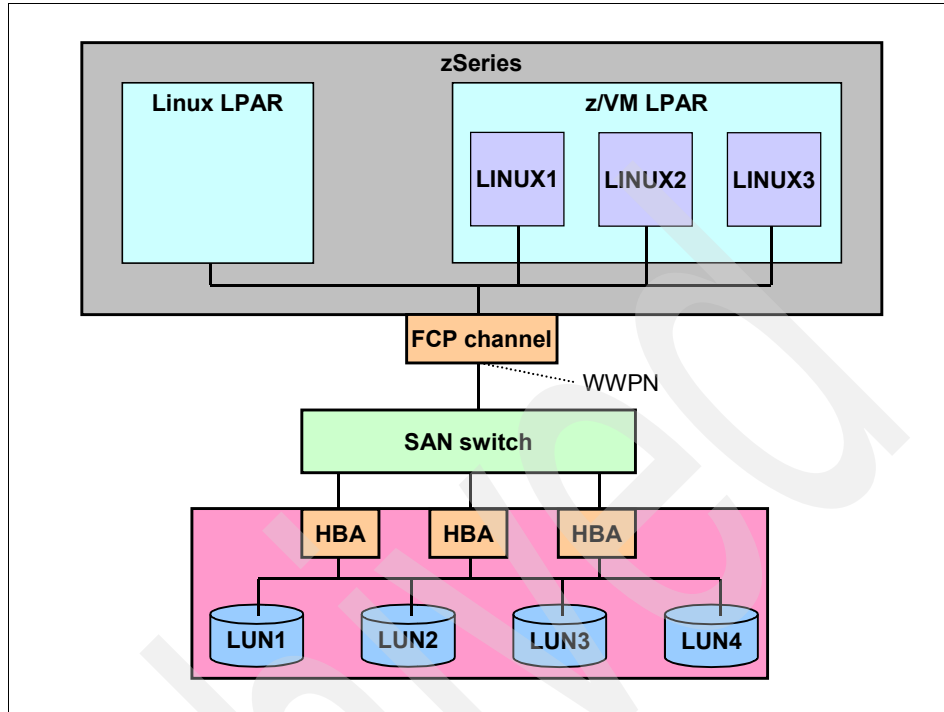


Figure 5-10 The effect of channel sharing in a SAN environment

In the figure, not the following:

- ▶ Three Linux guests running under z/VM share the same FCP channel.
- ▶ Using Multiple Image Facility (MIF), the FCP channel is also shared by a Linux server running in an LPAR.

In this case, zoning and LUN masking cannot be used to control connectivity. Each Linux server is granted to the LUNs on the storage server. A single WWPN is advertised to the SAN fabric (namely, the WWPN of the shared FCP channel). All four Linux servers are either granted access to the four LUNs, or they are all denied access.

FCP LUN Access Control provides a mechanism to define individual access rights to storage controller WWPNs and LUNs for each operating system image sharing an FCP channel. An access control table (ACT) defines access rights to the shared FCP channel. Individual operating systems sharing the FCP channel are granted or denied access to specific LUNs based on rules defined in the ACT.

5.4.1 FCP LUN Access Control operation

FCP LUN Access Control is available as licensed internal code (LIC) on System z9 and zSeries servers (at a minimum, LIC driver level 55 is required). Operation of the LUN Access Control feature is illustrated in Figure 5-11.

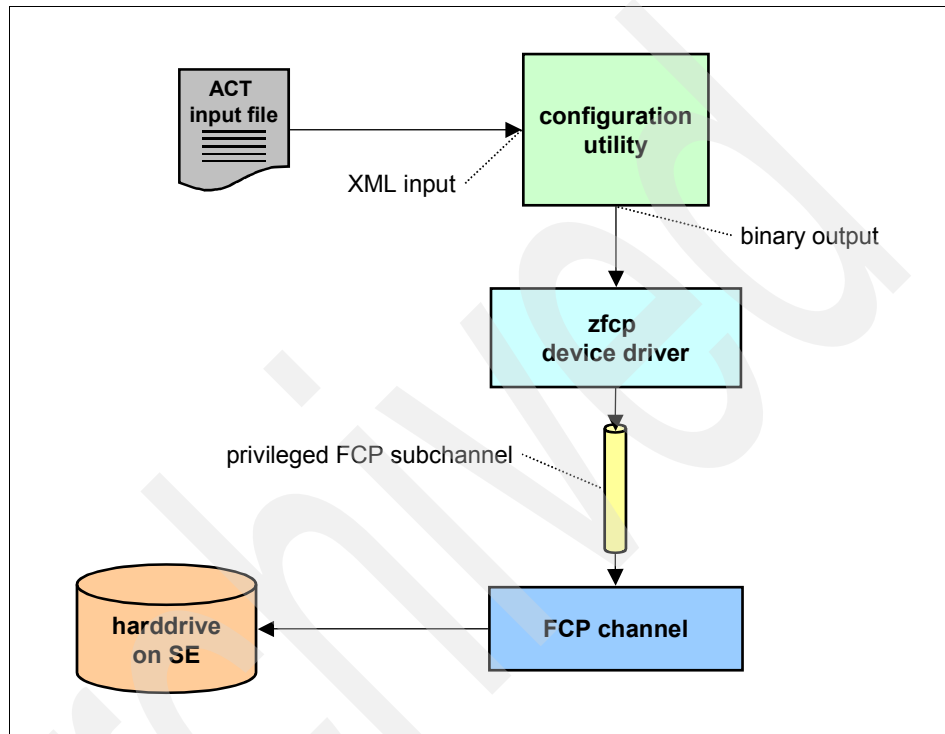


Figure 5-11 Operation of FCP LUN Access Control feature

In the figure:

- ▶ The ACT input file is an XML document. This describes the operating systems, WWPNs, and LUNs accessed over the FCP channel. Rules in the ACT grant or deny operating system access to specific WWPNs and LUNs.
- ▶ The configuration utility compiles the ACT input file to generate a binary ACT. Using the zfc device driver, the binary ACT is sent to FCP channel over a privileged FCP subchannel. With access control enabled, the FCP adapter intercepts outgoing I/O requests. Only those requests allowed by the ACT are sent by FCP channel to the FCP fabric. The configuration utility is provided as a separate RPM package.

- ▶ The FCP adapter saves the ACT on the primary and alternate Support Elements (SEs).

When a power on reset (POR) is performed, the ACT is read from the SE and restored to the FICON Express card. This ensures access control remains in effect across PORs, or in the event the card is replaced. If the primary SE is unavailable, the ACT is restored from the alternate SE.

5.4.2 Installing the FCP LUN Access Control feature

To install the LUN Access Control feature:

1. Download the configuration utility from Resource Link.

The utility is available as an RPM package for both 31-bit (the `actcli-1.1.0.s390.rpm` file) and 64-bit (the `actcli-1.1.0.s390x.rpm` file) distributions. The configuration utility can be obtained from IBM Resource Link. The RPM package contains:

- The `actcli` command (the interface to the configuration utility)
- The FCP LUN Access Control user's guide (*Configuration Utility for FCP LUN Access Control User's Guide*, SC33-8280)
- ACT template files in XML format

2. Install the prerequisite software.

For the Linux 2.6 kernel, the distribution should be based on the IBM developerWorks Linux on zSeries April 2004 stream at:

http://www.ibm.com/developerworks/linux/linux390/april2004_documentation.html

For the Linux 2.4 kernel, the distribution should be based on the June 2003 stream with all features from the 2004-01-30 code drop available at:

http://www.ibm.com/developerworks/linux/linux390/june2003_recommended.html

The `libxml2` package is also required (version 2.5.7 or higher). This is available as an RPM package from:

<ftp://xmlsoft.org>

If you are running Linux as a z/VM guest, z/VM 4.4 and higher are supported. z/VM 4.4 and z/VM 5.1 systems must include APAR VM63328.

3. Define access to at least one privileged FCP unit address.

The Linux for zSeries instance where the configuration utility runs must have access to at least one of the unit addresses `0xFC` or `0xFD` on the FCP channel defined in the IOCDS. If running Linux under z/VM, be sure to attach

the device to the guest. The Linux device node used to access the FCP channel is created by `actcli_init` command and is named `/dev/zfcp_cfdc`.

Access rights are granted using the FCP privileged devices (the `0xFC` and `0xFD` unit addresses). Ensure these unit addresses are only attached to trusted users. Access to the control port may require re-initialization of the device:

```
# chccwdev -d 0.0.0xb0fc
# chccwdev -e 0.0.0xb0f
```

4. Install the configuration utility.

Use the `rpm` command to install the configuration utility. The `actcli` executable installs in the `/usr/bin` directory; XML templates install in the `/opt/act` directory.

5.4.3 Using the configuration utility

The configuration utility uses the `actcli` command to access the FCP channel. The following examples show some typical scenarios:

- ▶ To syntax check an ACL source file named `config01_acl.xml`:

```
# actcli -v -D /dev/zfcp_cfdc -N config01_acl.xml 0.0.b0fc
```

If no errors are detected, a binary ACT is generated downloaded to the FCP adapter at device `0.0.b0fc`.

- ▶ To reset an FCP adapter accessed from device `0.0.b0fc`:

```
# actcli -v -D /dev/zfcp_cfdc -P allow 0.0.b0fc
```

The adapter is reset to default values (no access control restrictions).

- ▶ To retrieve the current ACT from the FCP adapter at address `0.0.b0fc` and store it in file `current_config.xml`:

```
# actcli -v -D /dev/zfcp_cfdc -L 0.0.b0fc > current_config.xml
```

The current ACT is retrieved from FCP adapter, converted to XML, and saved as a file.

Note: Version 1.1.2 of the `actcli` command adds the string “Operation executed successfully.” (which is intended as a console message) to the beginning of the file. Do not forget to delete this line if you intent to modify the XML file.

- ▶ To block all outgoing requests from the FCP adapter from all subchannels:

```
# actcli -v -D /dev/zfcp_cfdc -P deny 0.0.b0fc
```

The FCP adapter continues to accept configuration commands over the privileged unit addresses (0xFC and 0xFD).

5.4.4 A sample scenario

We create a simple access control file for three LUNs and two Linux z/VM guests. All are connected to the FCP fabric over one FCP channel to a single storage controller WWPN. The ACT is downloaded to the adapter using the command:

```
# actcli -v --device /dev/zfcp_cfdc -N b0fc_acl.xml 0.0.b0fc
```

If the ACT input file denies access to an open FCP connection, the new ACT configuration is not loaded to the adapter. For example, in Figure 5-12 the b0fc_default.xml ACT input file blocks access from the LNXSU1 Linux guest to LUN 0x5212000000000000. A warning message is issued and the operation terminates.

```
# actcli -v --device /dev/zfcp_cfdc -N b0fc_default.xml 0.0.b0fc
actcli: opening communication to FCP adapter via /dev/zfcp_cfdc.
Sending request to adapter B0FC ...
Adapter completed request ...
actcli: FSF Status: Access conflict(s) detected. 1 open connection(s) exist.
4 of the affected connections are listed:
LUN: 5212000000000000 configuring rule in XML in line 33
VM guest ID: LNXSU1 2nd level VM guest ID: devno: B005 cssID: 1
partition: 19
actcli: New Access Control Table not activated.
Need -F,--force to break open connections.
```

Figure 5-12 Configuration with conflicting access rights

Using the -F option can force the ACT to be loaded as shown in Figure 5-13 on page 174.

```

# actcli -v -F --device /dev/zfcp_cfdc -N b0fc_acl.xml 0.0.b0fc
actcli: ACT Identifier: CHPID90_EX1
=====
Warning:
You are about to force a change of the FCP LUN Access Control configuration.
Be sure that you understand the consequences this will have for
your installation.
Production systems might unintentionally be rendered inoperable
or go down because of denied accesses!
Sensitive data might be open for access that is not intended.

Are you sure you want to download the new configuration? (y/n) [n]: y
actcli: opening communication to FCP adapter via /dev/zfcp_cfdc.
Sending request to adapter B0FC ...
Adapter completed request ...
actcli: FSF Status: Downloaded ACT activated successfully
actcli: but not saved on alternate SE.
Retry the latest --new-table operation.

```

Figure 5-13 Forcing an FCP ACL configuration change

We decided to configure a simple access control table to show the restricted access. We used three LUNs: 0x5210000000000000, 0x5211000000000000, 0x5212000000000000 accessible with the physical FCP channels 90 and 91. These are attached to the storage controller WWPNs 0x5005076300cd9589 and 0x5005076300c19589. Two Linux images (LNXSU2 and LNXRH2), running in z/VM guests on the z990, providing the two CHPIDs 90 and 91.

```

LNXSU2 had exclusive read/write access on LUN 5210000000000000
LNXRH2 had exclusive read/write access on LUN 5211000000000000
LNXSU2 and LNXRH2 had read only access on LUN 5212000000000000

```

With our access control table activated, trying to access LUN 0x5211000000000000 in LNXSU2 results in (no matter if the LUN is already accessed by LNXRH2 or not), which proves the exclusive read/write right on LUN 0x5211000000000000 by LNXRH2:

```

zfcp: Access denied, cannot open unit 0x5211000000000000 on remote port
0x5005076300cd9589 on adapter 0.0.b006
zfcp: unit erp failed on unit 0x5211000000000000 on port 0x5005076300cd9589
on adapter 0.0.b006

```

This is an access denied message, generated by the access control feature. Don't confuse it with:

```

zfcp: FCP-LUN 0x5211000000000000 at the remote port with WWPN
0x5005076300cd9589 connected to the adapter 0.0.b006 is already in use in
LPAR9, CSS1

```

```
zfcpl: unit erp failed on unit 0x5211000000000000 on port 0x5005076300cd9589
on adapter 0.0.b006
```

Remember, without the access control feature it was possible to access any LUN in read/write mode (assuming a running physical connection, correct masking, zoning and binding definitions) as long as this LUN was not accessed by another linux images using the same FCP channel. Using different physical FCP channels, the same LUN could even be read/write accessed by several Linux images.

Important: If Linux was booted from SCSI, be careful when modifying the access control list. Ensure, that boot disk grants the correct access rights.

5.5 Multipathing for SCSI disks

In this section, we look at using multipathing with SCSI disks on Linux for zSeries. In general, there are two reasons establishing a multiple paths to a device:

- ▶ *High availability* provides several physical paths to a device offers high availability in a failure scenario; if one path fails, other paths to the device are still available.
- ▶ *Performance* is achieved by using multiple paths simultaneously to write to or read from a device can significantly increase I/O performance.

Redundant paths defined to Linux appear as separate Linux devices as shown in Figure 5-14 on page 176.

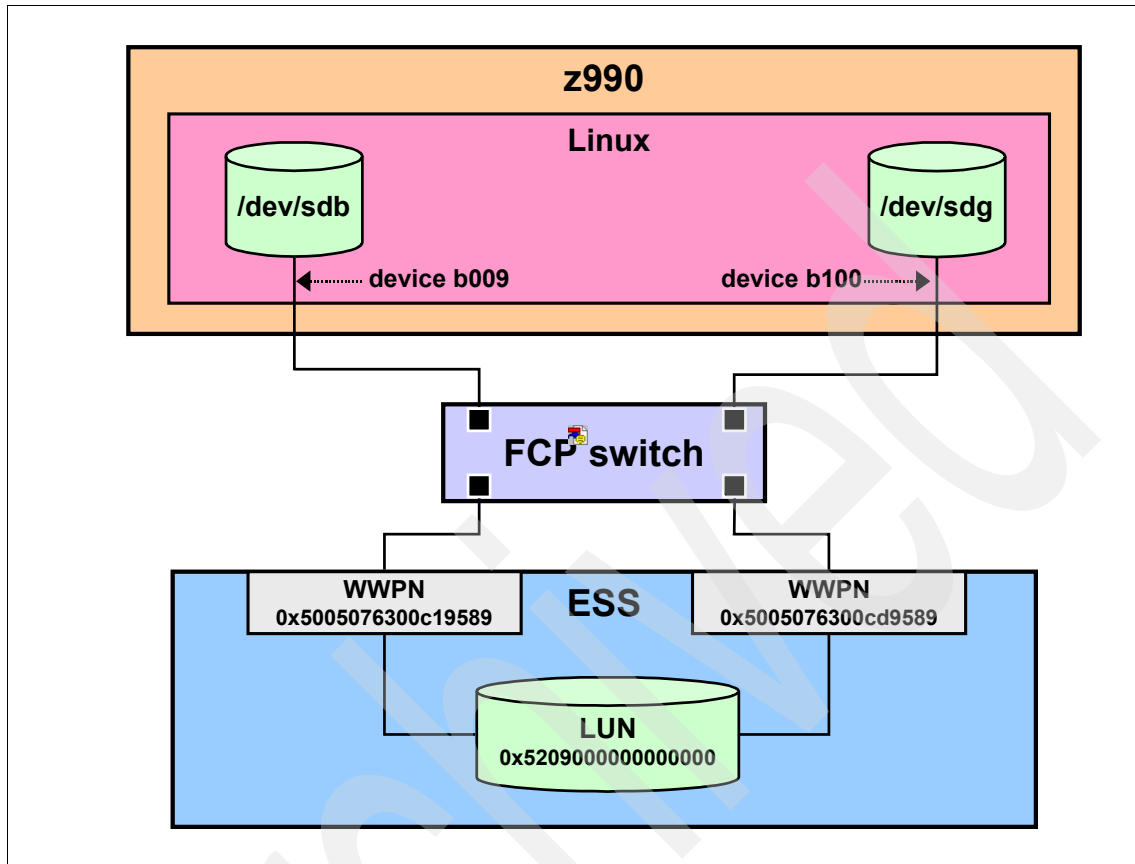


Figure 5-14 A multipath scenario

In the figure, two paths to a single physical LUN (0x5209000000000000). Linux accesses each path using a separate device:

- ▶ Device /dev/sdb accesses the LUN using WWPN 0x5005076300cd9589 over FCP device b009.
- ▶ Device /dev/sdg accesses the LUN using WWPN 0x5005076300c19589 over FCP device b100.

5.5.1 Multipathing using EVMS on SLES9

In this section, we illustrate using multipathing using EVMS on SLES9.

Figure 5-15 on page 177 shows the subchannel and udev mapping for the disks.

```

# lscss | grep "B"
0.0.B006 0.0.000F 1732/03 1731/03 yes 80 80 FF 90000000 00000000
0.0.B009 0.0.0011 1732/03 1731/03 yes 80 80 FF 90000000 00000000
0.0.B100 0.0.0013 1732/03 1731/03 yes 80 80 FF 91000000 00000000
# udevinfo -q all -n /dev/sdb
P: /block/sdb
N: sdb
M: 0660
S: disk/by-path/ccw-0.0.b009-zfcp-0x5005076300cd9589:0x5209000000000000
O: root
G: disk
# udevinfo -q all -n /dev/sdg
P: /block/sdg
N: sdg
M: 0660
S: disk/by-path/ccw-0.0.b100-zfcp-0x5005076300c19589:0x5209000000000000
disk/by-id/1IBM_2105_20922513
O: root
G: disk

```

Figure 5-15 Subchannel and udev information about multipath disks

From this information, we see:

- ▶ Device /dev/sdb is connected using device b009. It is attached to CHPID 90, and accesses the LUN using WWPN 0x5005076300cd9589.
- ▶ Device /dev/sdg is connected using device b100. It is attached to CHPID 91, and accesses the LUN using WWPN 0x5005076300c19589.

After formatting and partitioning the SCSI disks, we start the ncurses interface to EVMS using the `evmsn` command. Use the **Tab** bar to switch between the different available views. We first ensure the two devices are in the available object list, then:

1. Assign a DOS Segment Manager to each multipath disk.

From the EVMS main menu, navigate to the Add Segment Manager dialog in Figure 5-16 on page 178 using **Actions** → **Add** → **Segment Manager** → **Storage Object**.

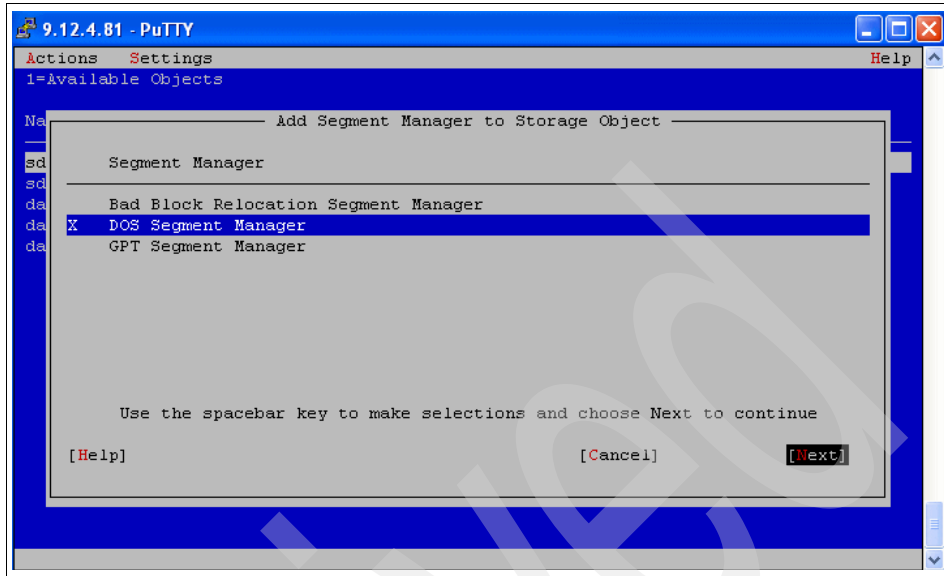


Figure 5-16 Add DOS Segment Manager selection

Select the DOS Segment Manager and click **Next** to assign the segment manager to a specific disk as shown in Figure 5-17.

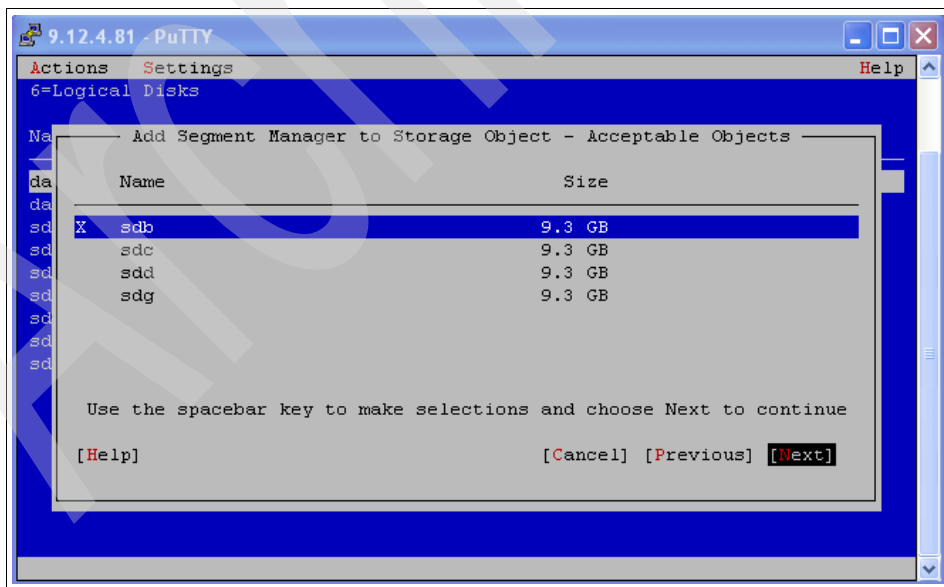


Figure 5-17 Add segment manager to SCSI disk

Use the spacebar to select the desired disk and click **Next** to assign a segment manager to the disk in Figure 5-18.

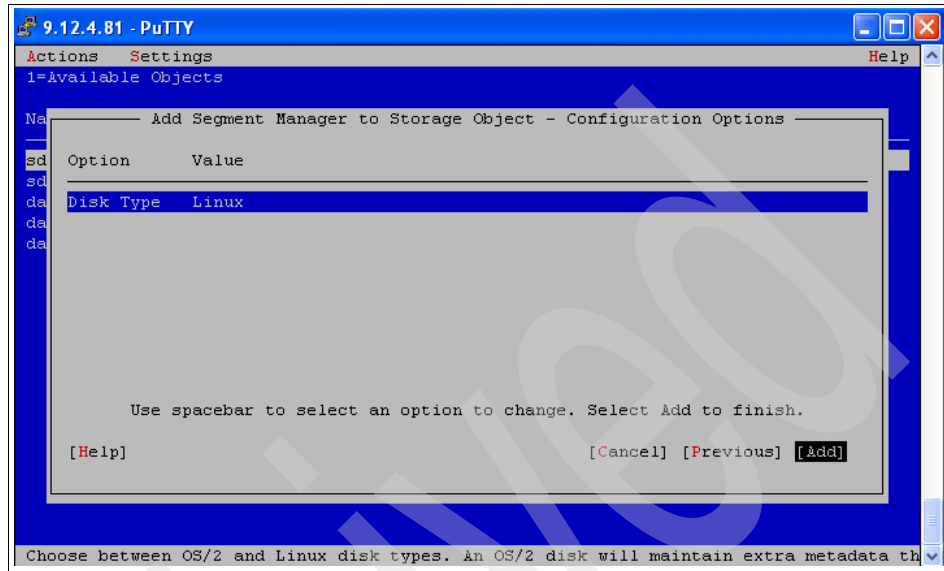


Figure 5-18 Assign the segment manager to a Linux disk

Repeat these steps for each multipath disk.

2. Create EVMS segments on the disks.

Next, create a EVMS segment on each SCSI disk. From the Disk Segments view, select the SCSI disk and press **Enter** to open the disk details dialog box in Figure 5-19 on page 180.

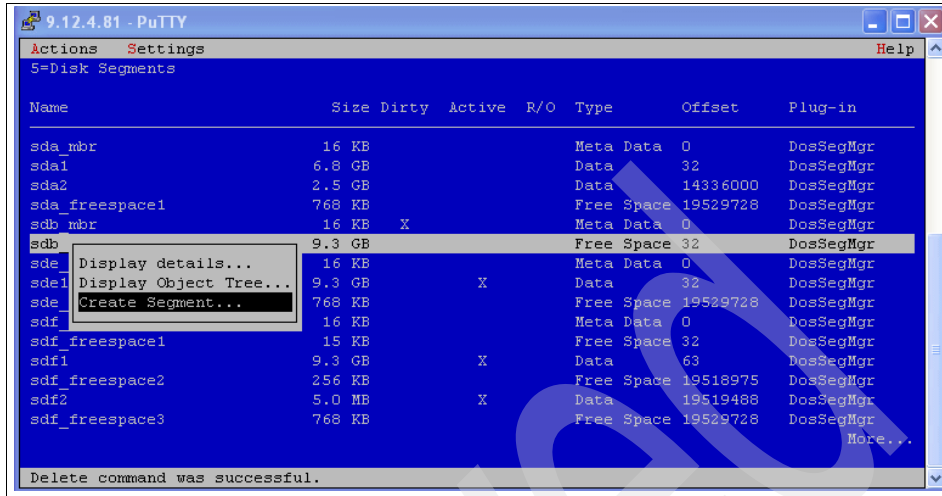


Figure 5-19 The disk details dialog

Select **Create Segment**, and press **OK** on the next screen to confirm the operation. Create an EVMS segment on each multipath disk.

Important: When segments are created on each SCSI disk, exit and restart the EVMS nurses interface. When restarted, EVMS will see the new segments created on the devices.

3. Create a multipath storage region.

When EVMS is restarted, navigate **Actions** → **Create** → **Region** to the Create Storage Region dialog shown in Figure 5-20 on page 181.

Click **Create** to complete the operation.

4. Create a Compatibility Volume.

Next, create a compatibility volume on the multipath region by navigating **Actions** → **Create** → **Compatibility Volume** to the panel in Figure 5-22.

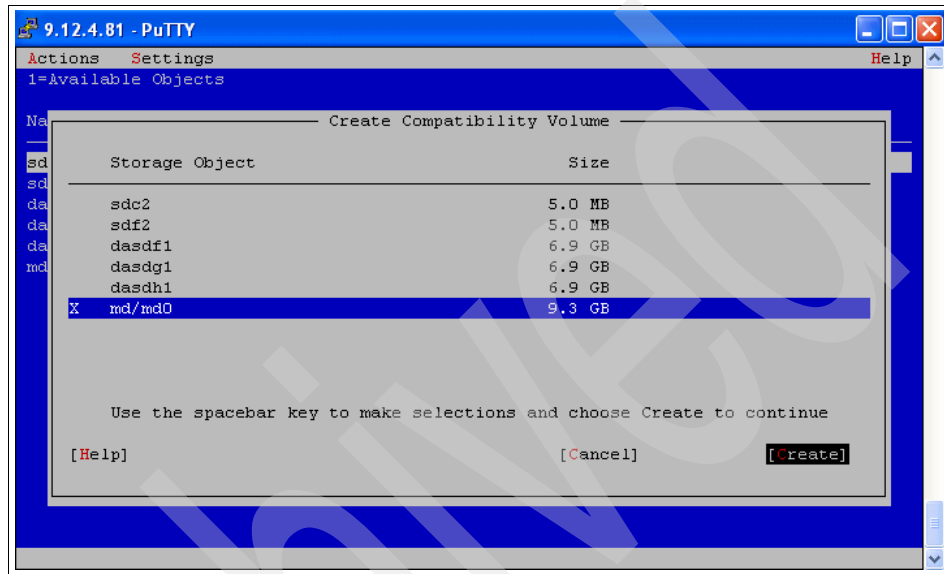


Figure 5-22 Create a compatibility volume

Choose the multipath region created in the previous step (md/md0 in the figure). Hit **Create** to complete the operation. The newly created volume is now available under the Storage Regions view.

5. Create a file system on the volume.

To use the new volume, a filesystem must be created on it. Select the volume in the logical volumes view and pressing **Enter** for the panel shown in Figure 5-23 on page 183.

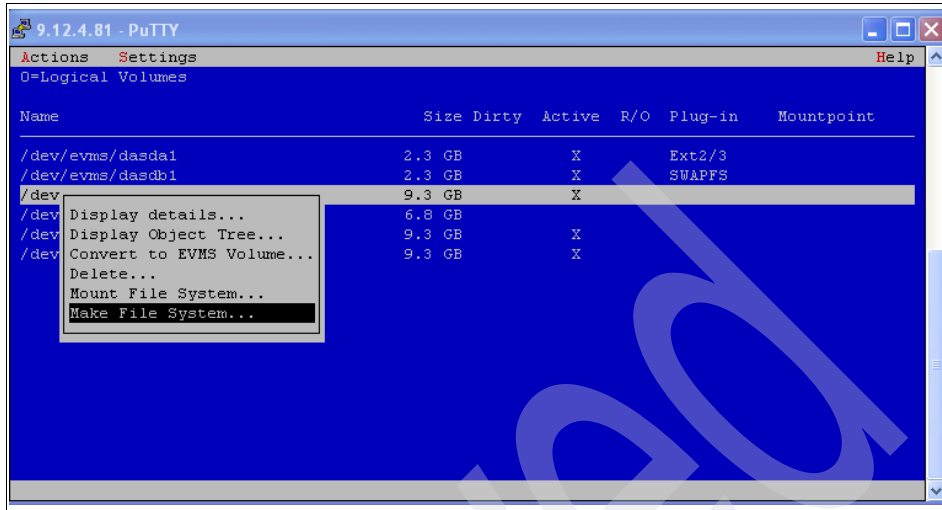


Figure 5-23 Create a filesystem on the volume

Select the **Make file System** option and press **Enter** for the panel shown in Figure 5-24.

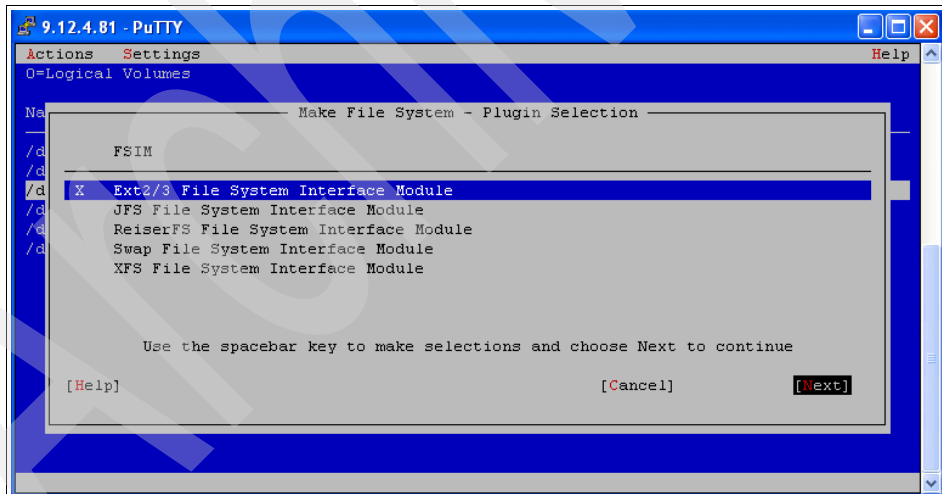


Figure 5-24 Select the filesystem type

Click **Next** to customize options in Figure 5-25 on page 184.

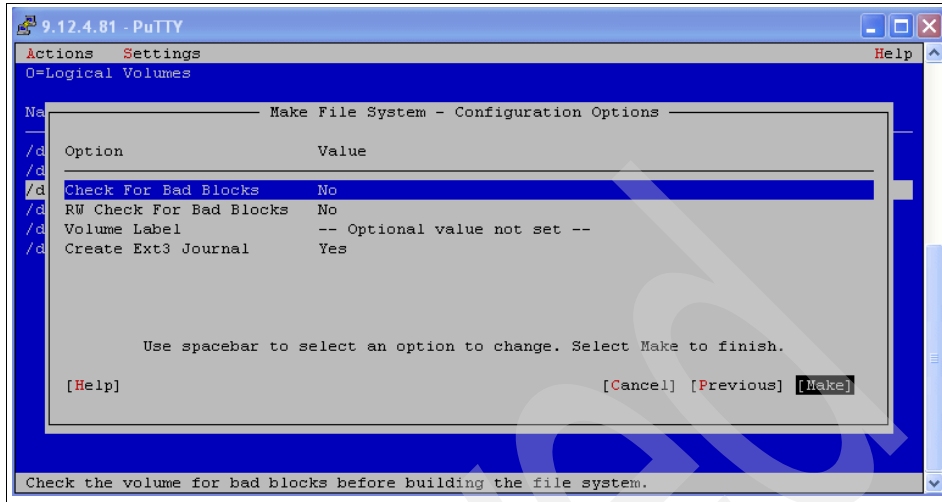


Figure 5-25 Customize filesystem options

Press **Make** to complete the operation.

The EVMS configuration can be activated using the **evms_activate** command

Note: The multipath array requires at least one working path to the LUN. If the last path to the LUN is lost, the file system enters read-only when access is re-established.

5.5.2 Multipathing using mdadm on RHEL4

With RHEL4, multipathing is implemented using the mdadm (multiple device administration) driver. To illustrate using mdadm, we consider the example shown in Figure 5-26 on page 185.

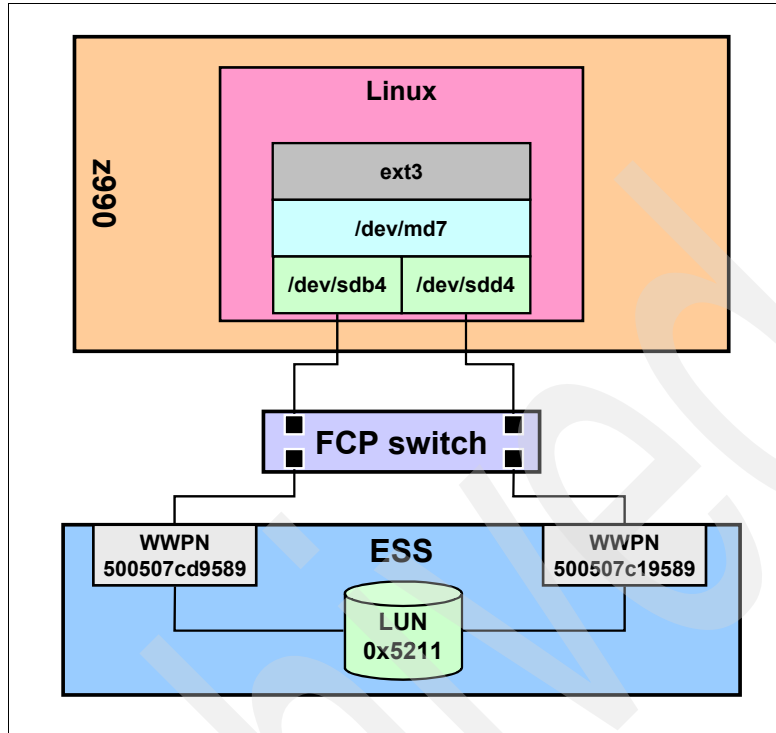


Figure 5-26 Multipath example using mdadm

In this example, LUN 0x5211000000000000 is accessed over two paths to device nodes /dev/sdb4 and /dev/sdd4. These devices are combined in the /dev/md7 mdadm device node, and an ext3 filesystem is created on the device.

To create the mdadm device node, we use the **mdadm** command:

```
# mdadm -C /dev/md7 --level=multipath --raid-devices=2 /dev/sdb4 /dev/sdd4
mdadm: array /dev/md7 started.
```

We can check the new device node status using the **-D** option of the **mdadm** command (Example 5-5):

Example 5-5 Using mdadm with **-D**

```
# mdadm -D /dev/md7
/dev/md7:
  Version : 00.90.01
  Creation Time : Wed May 4 12:03:43 2005
  Raid Level : multipath
  Array Size : 19530624 (18.63 GiB 19.100 GB)
  Raid Devices : 2
```

```
Total Devices : 2
Preferred Minor : 3
Persistence : Superblock is persistent
```

```
Update Time : Wed May 4 12:03:43 2005
State : clean
Active Devices : 2
Working Devices : 2
Failed Devices : 0
Spare Devices : 0
```

Number	Major	Minor	RaidDevice	State	
0	8	20	0	active sync	/dev/sdb4
1	8	52	1	active sync	/dev/sdd4

UUID : dc50d458:39b13736:bd76c363:029d22cb
Events : 0.1

Finally, we create an ext3 filesystem on /dev/md7:

```
# mke2fs -j /dev/md7
mke2fs 1.35 (28-Feb-2004)
warning: 224 blocks unused.
.
.
.
```


Using ECKD storage

This chapter discusses traditional ECKD DASD storage management in the Linux 2.6 kernel on zSeries. Topics include:

- ▶ ESS operating mode
- ▶ Configuring DASD at IPL with SLES9
- ▶ Configuring DASD at IPL with RHEL4
- ▶ Parallel Access Volumes

6.1 ESS operating mode

The IBM TotalStorage Enterprise Storage Server (ESS) offers different modes of operation. Depending on the workload, the operation mode can be adjusted for better performance. Operation modes include:

- ▶ Normal Cache Replacement
In this mode, the cache is managed using the ESS standard algorithms. This is the default value for SCSI disks, and for ECKD DASD in SLES9 and SLES8 (beginning with Service Pack 3).
- ▶ Sequential Access or Prestage
Sequential access is the default for SLES7 and earlier versions of SLES8. This mode can lead to degraded performance for workloads that require random I/O.
- ▶ Inhibit Cache Loading or Record Access
These special modes should be used carefully, but can provide good performance when performing random I/O.
- ▶ Bypass Cache
This special mode should also be used carefully. It may provide a moderate performance improvement for random I/O.

Operation mode cannot be changed for SCSI disks; the FCP driver uses Normal Cache Replacement. The `-g` option of the `tunedasd` command returns the current mode; the `-c` option sets the operation mode (Example 6-1):

Example 6-1 Using the tunedasd command

```
# tunedasd -g /dev/dasdb
normal
# tunedasd -c sequential /dev/dasda
Setting cache mode for device </dev/dasda>...
Done.
```

Note: For more information about ESS operating mode, see the developerWorks site:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_rec_dasd_cac_hemode.html

6.2 Configuring DASD at IPL with SLES9

With SLES9, you no longer need to run the `mkinitrd` (to create an initial ramdisk) and `zipl` (to modify the bootloader) commands in order to configure new ECKD DASD devices at Linux IPL. These are automatically recognized by `sysfs` at system boot. Figure 6-1 shows the `sysfs` entry for the DASD device at address 0403.

```
# ls /sys/bus/ccw/drivers/dasd-eckd/ -go
total 0
drwxr-xr-x  2 0 Apr 25 16:04 .
drwxr-xr-x  7 0 Apr 25 16:04 ..
lrwxrwxrwx  1 0 Apr 25 16:04 0.0.0403 ->../../../../devices/css0/0.0.0000/0.0.0403
```

Figure 6-1 The `sysfs` structure for ECKD DASD

The configuration files for DASD devices are found in the `/etc/sysconfig/hardware` directory. File names take the form `hwcfg-dasd-bus-ccw-0.0.xxxx` where `xxxx` is the value of the device virtual address. To create an ECKD DASD configuration file manually, use the skeletal configuration file in the `/etc/sysconfig/hardware` directory:

```
# cp /etc/sysconfig/hardware/skel/hwcfg-dasd-eckd \
/etc/sysconfig/hardware/hwcfg-dasd-bus-ccw-0.0.0201
```

In the example, the skeletal configuration file is used to configure the ECKD DASD device at address 0201. To bring the new online, use the `hwup` command:

```
# hwup /etc/sysconfig/hardware/skel/dasd-bus-ccw-0.0.0201
```

Alternatively, the `chccwdev` command with the `-e` option brings the device online:

```
# chccwdev -e 0.0.0201
```

Note: The boot device still needs to be specified in `/etc/fstab` and `/etc/zipl.conf`. No further definitions needed for ECKD DASD devices needed at system boot.

6.3 Configuring DASD at IPL with RHEL4

Configuring DASD devices at IPL for RHEL4 is slightly different than SLES9. To configure the device at system boot:

1. Add the device to the `/etc/modprobe.conf` file:

```
# cat /etc/modprobe.conf
alias eth0 qeth
```

```
options dasd_mod dasd=201,202,1592
alias scsi_hostadapter zfc
```

Note: Be sure to first bring the device online:

```
# chccwdev -e 0.0.1592
Device already is online
```

2. Creating a new initial ramdisk (Example 6-2):

Example 6-2 New initial ramdisk

```
# cd /boot
# mv initrd-2.6.9-6.37.EL.img initrd-2.6.9-6.37.EL.img.old
# mkinitrd -v --with=scsi_mod --with=zfc --with=sd_mod \
initrd-2.6.9-6.37.EL.img 2.6.9-6.37.EL
```

3. Execute the `zipl` command to update the bootloader.

Note: If DASD is to be a boot device, do not forget to define it in `/etc/fstab` and `/etc/zipl.conf`.

For more detailed information, refer to the RedHat documentation, *Installation Guide for the IBM S/390 and IBM eServer zSeries Architectures*.

6.4 Parallel Access Volumes

Parallel Access Volumes (PAV) allows a zSeries host system to perform multiple, concurrent data transfers to a single volume on a storage server. With PAV, the storage server volume is accessed using a base device and one more alias devices. The base represents the real volume, and aliases represent alternate access paths to the volume. Multiple read requests can be processed concurrently. To maintain data integrity, write operations to the same domain are serialized (however, as almost all writes are cache hits, there is typically only a short delay). PAV functionality is handled by the storage server (eliminating I/O enqueues and rescheduling that would normally be handled by the host channel subsystem).

Linux does not attempt to start more than one simultaneous I/O operation a single device. In order to take advantage of parallel access, a PAV volume is represented as multiple Linux devices, the base device and its alias devices. These devices are aggregated into a single logical volume using a volume manager such as LVM, EVMS, or mdadm.

To use PAV with Linux for zSeries:

- ▶ The volume must reside on a storage server that supports PAV.
- ▶ Linux must run as a z/VM guest (Linux running in an LPAR does not support PAV).
- ▶ A multipath tool (such as LVM, mdadm, or EVMS) must be used to prevent data inconsistency when using PAV.

6.4.1 Defining PAV devices

PAV devices must first be defined in the IOCDs, and configured in the ESS (for details, refer to *IBM TotalStorage Enterprise Storage Server Model 800, SG24-6424*). In Figure 6-2, C70D is defined as base devices, and C7C8 and C7C9 are defined as alias devices. The relevant IOCDs statements are in Example 6-3:

Example 6-3 IOCDs statements

```
IODEVICE ADDRESS=(C70D),UNITADD=00,CUNUMBR=(C70C), *
  STADET=Y,UNIT=3390B
IODEVICE ADDRESS=(C7C8),UNITADD=18,CUNUMBR=(C7CC), *
  STADET=Y,UNIT=3390A
IODEVICE ADDRESS=(C7C9),UNITADD=18,CUNUMBR=(C7CD), *
  STADET=Y,UNIT=3390A
```

After the IOCDs changes are activated, we check their availability in z/VM using the Q PAV command shown in Figure 6-2.

```
Q PAV
Device C70D is a base Parallel Access Volume with the following aliases: C7CC C7CD
Device C7C8 is an alias Parallel Access Volume device whose base device is C70C
Device C7C9 is an alias Parallel Access Volume device whose base device is C70C

Ready; T=0.01/0.01 10:14:19
ATT C70D LNXSU2
DASD C70D ATTACHED TO LNXRH2 C70C WITH DEVCTL
Ready; T=0.01/0.01 10:17:04
ATT C7C8-C7C9 LNXSU2
C7C8-C7C9 ATTACHED TO LNXSU2
Ready; T=0.01/0.01 16:03:20
```

Figure 6-2 Checking PAV device availability

Once the PAV devices are available, we attach the devices to the Linux guest.

6.4.2 Preparing PAV devices in Linux

When attached to the Linux guest and brought online, the PAV device nodes should be created automatically. The base and alias devices address the same physical disk. Therefore, the volume can be formatted and partitioned using either the base or alias device. In Figure 6-3, we bring the PAV devices online and check their status using the `lsdasd` command.

```
# chccwdev -e c70d c7c8 c7c9
Setting device 0.0.c70d online
Done
Setting device 0.0.c7c8 online
Done
Setting device 0.0.c7c9 online
Done
# lsdasd
0.0.0201(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0301(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0401(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 601020 blocks, 2347 MB
0.0.c70d(ECKD) at ( 94: 12) is dasdd : active at blocksize 4096, 1803060 blocks, 7043 MB
0.0.c7c8(ECKD) at ( 94: 16) is dasde : active at blocksize 4096, 1803060 blocks, 7043 MB
0.0.c7c9(ECKD) at ( 94: 20) is dasdf : active at blocksize 4096, 1803060 blocks, 7043 MB
```

Figure 6-3 Setting the PAV devices online

The `lscss` command shows the subchannels used to the devices. These subchannels share the physical channels 88, 89, 8C and 8D (Example 6-4):

Example 6-4 Using the `lscss` command

```
# lscss | grep "C7"
0.0.C70D 0.0.0015 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000
0.0.C7C8 0.0.0016 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000
0.0.C7C9 0.0.0017 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000
```

6.4.3 Defining PAV devices with EVMS

When they are online, we format and partition the volume:

```
dasdfmt -f /dev/dasdd
fdasd /dev/dasdd
```

Next, we start EVMS using the `evmsn` command. Use the **Tab** key to switch between the different views. Using EVMS, we create a single logical volume from the PAV devices. From the EVMS logical view, we:

1. Delete any existing logical volumes.

We first delete any existing logical volume on the PAV devices. Navigate **Actions** → **Delete** → **Volumes** to the display the Delete Logical Volumes dialog shown in Figure 6-4.

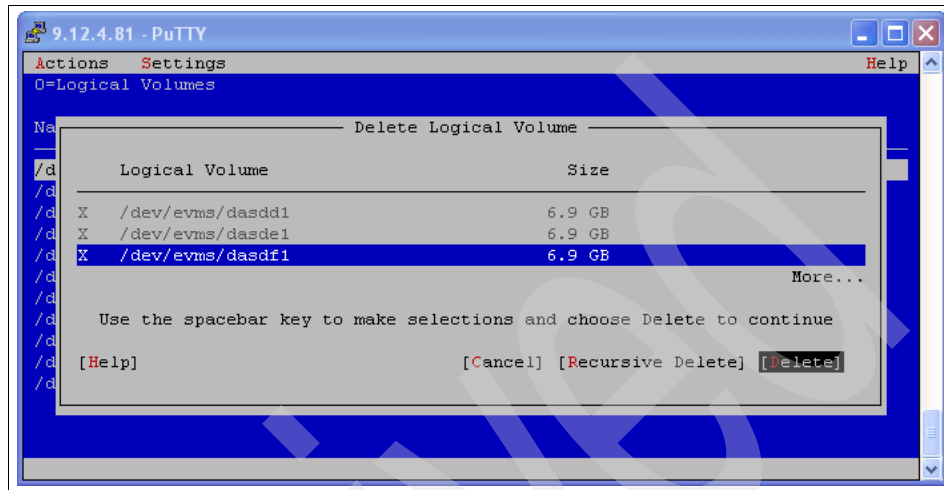


Figure 6-4 EVMS Delete Logical Volumes dialog

Use the spacebar to select the PAV logical volumes; click **Delete** to continue. In the dialog box that follows, select the **Write zeros** option to delete any existing data.

Note: Any existing data on the volume will be lost.

2. Create an EVMS storage region.

Next, change to the Available Objects view and navigate **Actions** → **Create** → **Region** to display the create storage region dialog. Select the **MD Multipath Region Manager** option as shown in Figure 6-5 on page 194.

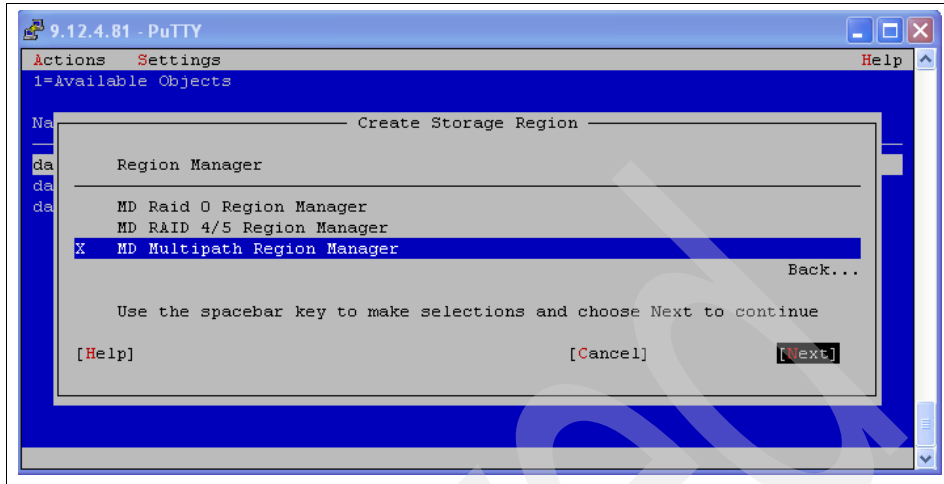


Figure 6-5 EVMS create storage region dialog

Click **Next** to go to the dialog box in Figure 6-6, add the desired PAV devices, and click **Create**.

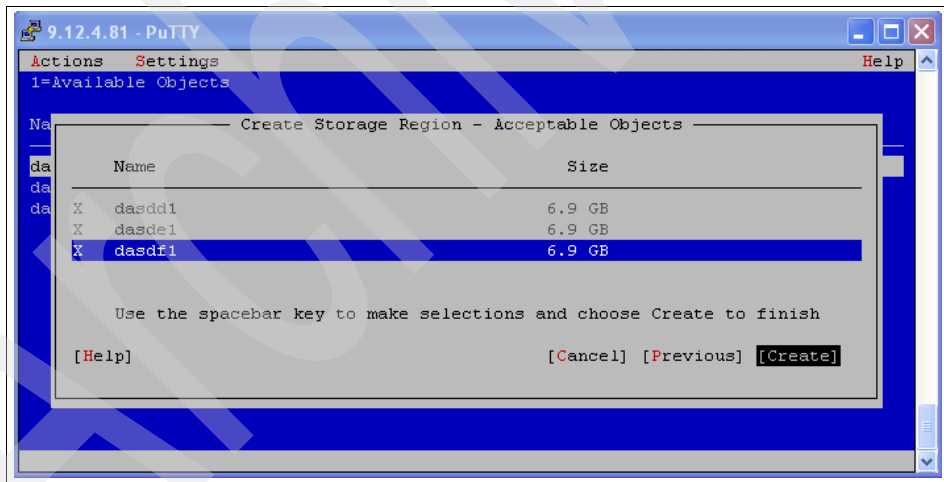


Figure 6-6 Select storage object

3. Create an EVMS volume on the region.

Next, we create an EVMS volume on the region. Navigate **Action** → **Create** → **EVMS Volume** to display the dialog in Figure 6-7 on page 195. Enter the name of the EVMS volume and hit **Enter**. When prompted, click **Create** to build the multipath volume.

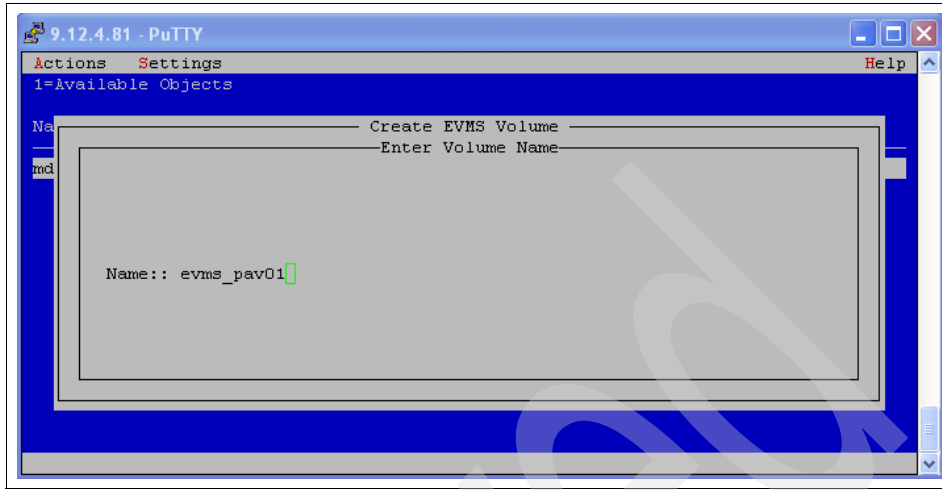


Figure 6-7 EVMS create volume dialog

4. Save the EVMS configuration.

Up to this point, changes have not been committed to disk (this is indicated by an **X** in the dirty column). To save the definitions:

- a. Navigate **Actions** → **Save** to the Save Changes Configuration dialog box.

After an EVMS multipath volume is created, the participating devices cannot be brought offline (for instance, using the **chccwdev** command).

Note: PAV multipath arrays built by LVM2 and EVMS *are* not compatible. However, EVMS storage container are compatible with LVM1 volume groups. This means that EMVS regions (created on a storage container) are compatible with LVM logical volumes (built on LVM1 volume groups).

Create a file system on the multipath volume

To use the multipath volume, we create a file system on it using EVMS. First, we must quit, then restart EVMS (EVMS needs to scan the volumes which compose the multipath volume).

When EVMS is restarted, we:

1. Select the multipath array in the Logical Volumes view.

Press Enter to open the volume related properties window shown in Figure 6-8 on page 196.

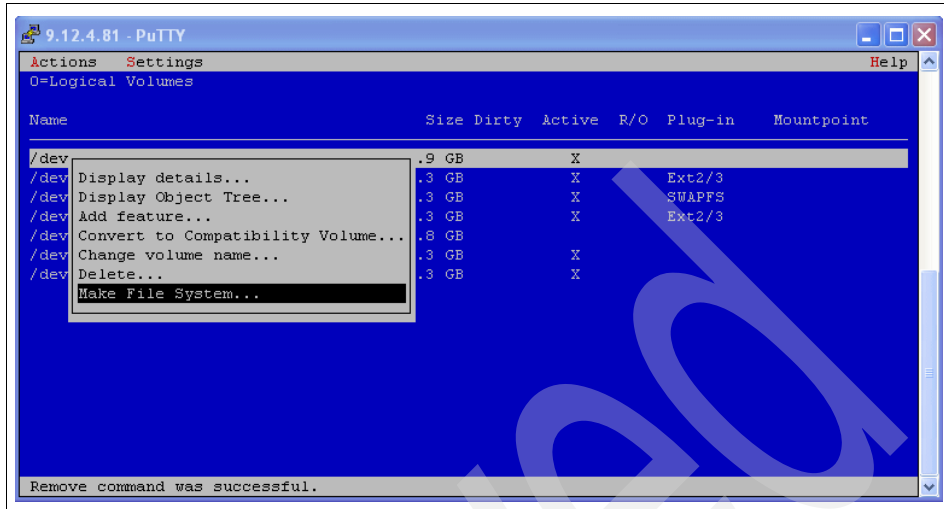


Figure 6-8 EVMS volume properties window

Select the **Make File System** option.

2. Select filesystem type.

Next, create your type of filesystem. In Figure 6-9, we choose an ext2/ext3 filesystem. Click **Next** to continue.

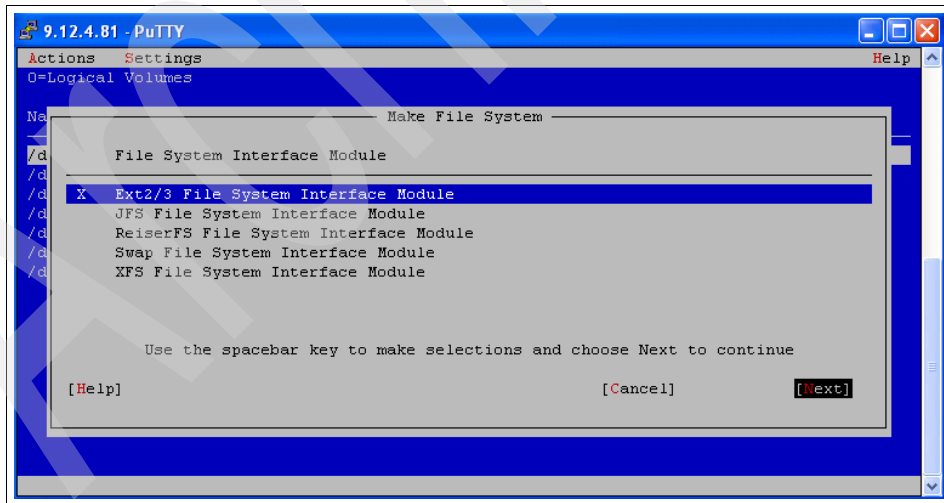


Figure 6-9 Select filesystem type to create

3. Select the PAV logical volume.

Choose the PAV logical volume on which the filesystem is to be created. In Figure 6-10, we choose the previously create /dev/evms/dasd_pav logical volume.

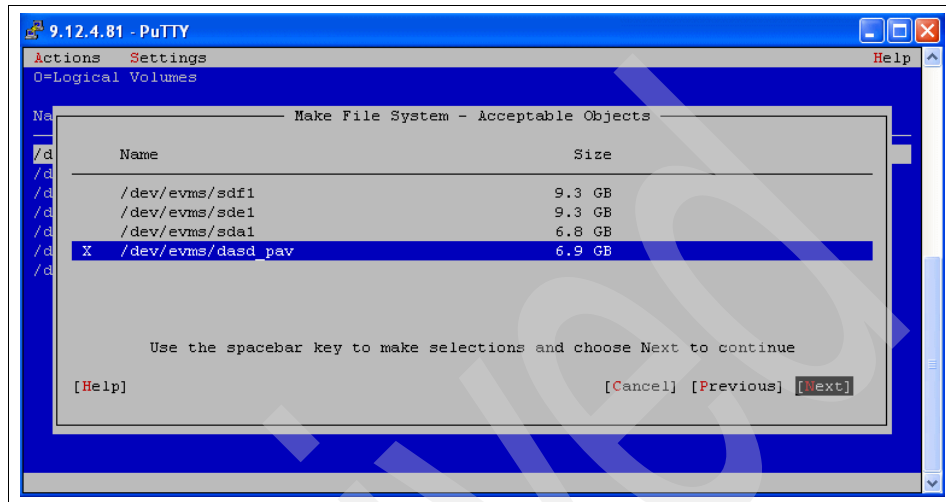


Figure 6-10 Select the PAV logical volume

Click **Next** to customize the filesystem configuration as shown in Figure 6-11.

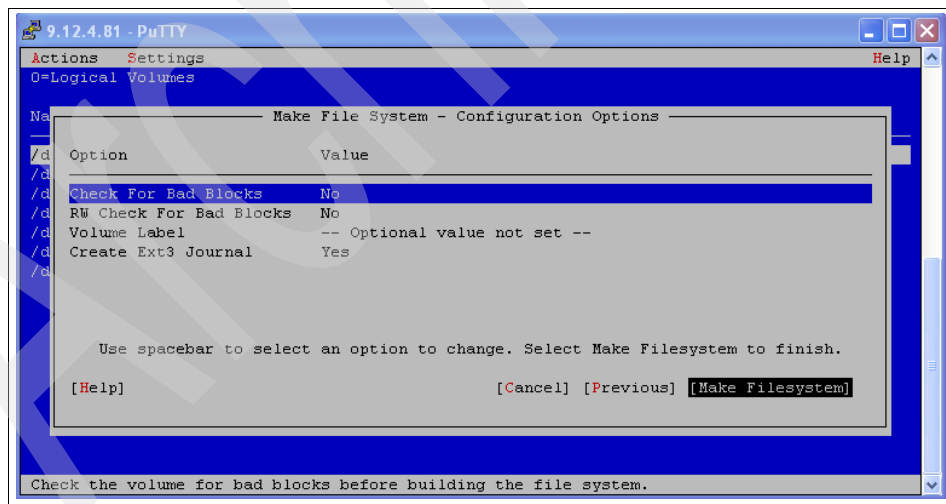


Figure 6-11 Filesystem configuration options

4. To complete the operation, click **Make Filesystem**.

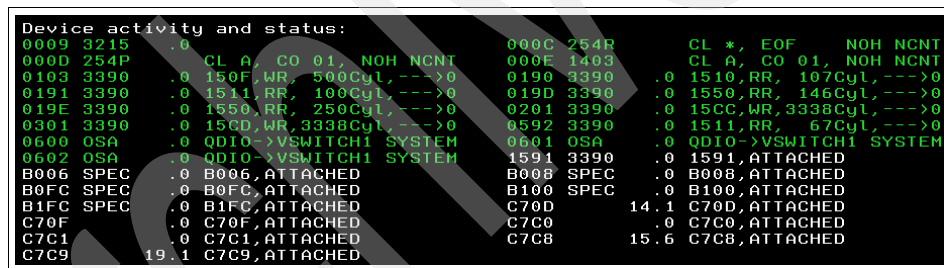
Checking PAV operation

To ensure EVMS is utilizing PAV, we can use the Performance Toolkit to monitor disk I/O. First we mount the PAV logical volume, then write to a file on the filesystem (Example 6-5):

Example 6-5 Mounting the PAV and writing a file

```
# mount /dev/evms/dasd_pav /mnt
# cd /mnt
# cat /dev/zero > output
cat: write error: No space left on device
# ls -l
total 7065808
drwxr-xr-x  3 root root      4096 May  4 11:53 .
drwxr-xr-x 20 root root      4096 May  4 08:50 ..
drwx-----  2 root root    16384 May  4 11:49 lost+found
-rw-r--r--  1 root root 7228289024 May  4 11:58 output
```

In Figure 6-12, we see all three paths to the physical device are used during the process.



```
Device activity and status:
0009 3215 .0
000D 254P .0 CL A, CO 01, NOH NCNT
0103 3390 .0 150F,WR, 500Cyl,--->0
0191 3390 .0 1511,RR, 100Cyl,--->0
019E 3390 .0 1550,RR, 250Cyl,--->0
0301 3390 .0 15CD,WR,3338Cyl,--->0
0600 OSA .0 QDIO->VSWITCH1 SYSTEM
0602 OSA .0 QDIO->VSWITCH1 SYSTEM
B006 SPEC .0 B006,ATTACHED
B0FC SPEC .0 B0FC,ATTACHED
B1FC SPEC .0 B1FC,ATTACHED
C70F .0 C70F,ATTACHED
C7C1 .0 C7C1,ATTACHED
C7C9 19.1 C7C9,ATTACHED
060C 254R CL *, EOF NOH NCNT
000E 1403 CL A, CO 01, NOH NCNT
0190 3390 .0 1510,RR, 107Cyl,--->0
019D 3390 .0 1550,RR, 146Cyl,--->0
0201 3390 .0 15CC,WR,3338Cyl,--->0
0592 3390 .0 1511,RR, 67Cyl,--->0
0601 OSA .0 QDIO->VSWITCH1 SYSTEM
1591 3390 .0 1591,ATTACHED
B008 SPEC .0 B008,ATTACHED
B100 SPEC .0 B100,ATTACHED
C70D 14.1 C70D,ATTACHED
C7C0 .0 C7C0,ATTACHED
C7C8 15.6 C7C8,ATTACHED
```

Figure 6-12 Checking I/O activity to a PAV logical volume

Note: An existing EVMS configuration can be activated after Linux is rebooted using the command `evms_activate` command.

6.4.4 Using PAV on with mdadm

With RHEL4, we use `mdadm` to take advantage of the PAV devices. We create an array of DASD devices: one base DASD device and two alias DASD devices. In our example, we configure PAV for devices `/dev/dasdd`, `/dev/dasde`, and `/dev/dasdf` as shown in Figure 6-13 on page 199.

```

# lsdasd
0.0.0201(ECKD) at ( 94: 0) is dasda : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0202(ECKD) at ( 94: 4) is dasdb : active at blocksize 4096, 600840 blocks, 2347 MB
0.0.0300(ECKD) at ( 94: 8) is dasdc : active at blocksize 4096, 900 blocks, 3 MB
0.0.c70c(ECKD) at ( 94: 12) is dasdd : active at blocksize 4096, 1803060 blocks, 7043 MB
0.0.c7cc(ECKD) at ( 94: 16) is dasde : active at blocksize 4096, 1803060 blocks, 7043 MB
0.0.c7cd(ECKD) at ( 94: 20) is dasdf : active at blocksize 4096, 1803060 blocks, 7043 MB

```

Figure 6-13 PAV devices /dev/dasdd, /dev/dasde, and /dev/dasdf

As illustrated in Figure 6-14, different subchannels are used to access the base and alias volumes. These subchannels share physical channels 88, 89, 8C and 8D.

```

# lscss | grep "C7"
0.0.C70C 0.0.0030 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000
0.0.C7CC 0.0.0031 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000
0.0.C7CD 0.0.0032 3390/0C 3990/E9 yes F0 F0 FF 88898C8D 00000000

```

Figure 6-14 Subchannel used by PAV devices

1. We first create PAV DASD array /dev/md4 using the mdadm command:

```

# mdadm -C /dev/md4 --level=multipath --raid-devices=3 /dev/dasdd1 \
/dev/dasde1 /dev/dasdf1
mdadm: array /dev/md4 started.

```

2. Next, we create a file system on the newly created array (Example 6-6):

Example 6-6 Creating files on the array

```

# mke2fs -j /dev/md4
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
903168 inodes, 1803008 blocks
90150 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1849688064
56 block groups
32768 blocks per group, 32768 fragments per group
16128 inodes per group

```

Superblock backups stored on blocks:

32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Writing inode tables: done

Using the `-D` option, the `mdadm` command displays the device array information (Example 6-7):

Example 6-7 Using mdadm with -d for device arrays

```
# mdadm -D /dev/md4
/dev/md4:
  Version : 00.90.01
  Creation Time : Wed May 4 13:06:08 2005
  Raid Level : multipath
  Array Size : 7212032 (6.88 GiB 7.39 GB)
  Raid Devices : 3
  Total Devices : 3
  Preferred Minor : 4
  Persistence : Superblock is persistent

  Update Time : Wed May 4 13:06:08 2005
  State : clean
  Active Devices : 3
  Working Devices : 3
  Failed Devices : 0
  Spare Devices : 0

   Number   Major   Minor   RaidDevice State
    0         94      13         0   active sync  /dev/dasdd1
    1         94      17         1   active sync  /dev/dasde1
    2         94      21         2   active sync  /dev/dasdf1
  UUID : 2cf81aac:309e4182:508ba054:fc51d3f3
  Events : 0.1
```

To check the currently active members of the array, use the `--examine` option as shown in Figure 6-15.

```
# mdadm --examine --brief --scan --config=partitions
ARRAY /dev/md4 level=multipath num-devices=3 UUID=9252b5f8:57ea5f9e:a96b9e66:f1d3ed62
devices=/dev/dasdd1,/dev/dasde1,/dev/dasdf1
```

Figure 6-15 Print the contents of the superblock on the md device

In the RHEL4, mdadm is not able to take full advantage of the PAV. It can only use alias devices as redundant paths to the physical device. To illustrate this, we mount the same array twice and started two disk write processes (Example 6-8):

Example 6-8 Multiple paths in RHEL4

```
# mount /dev/md4 /mnt
# cd /mnt
# cat /dev/zero > output
# mount /dev/md4 /tmp
# cd /tmp
# cat /dev/zero > other
```

In Figure 6-16, we see that only the base device (C70C) is used to write the two files.

Data	Space	Name	Size	Mode	PgRd/s	PgWr/s	XRd/s	XWr/s	Migr/s	Steal/s
BASE			256MB	Priv	.0	.0	.0	.0	.0	.0
Device activity and status:										
0009	3215	.0			000C	254R				
000D	254P	.0		CL A, CO 01, NOH NCNT	000E	1403				
0103	3390	.0	150C, WR,	500Cyl, --->0	0190	3390	.0	1510, RR,	107Cyl, --->0	
0191	3390	.0	1511, RR,	100Cyl, --->0	019D	3390	.0	1550, RR,	146Cyl, --->0	
019E	3390	.0	1550, RR,	250Cyl, --->0	0201	3390	.0	15D2, WR,	3338Cyl, ->163	
0202	3390	1.6	158C, WR,	33% MDC eff.	0203	3390	.0	158D, WR,	3338Cyl, --->0	
0300	3390	.0	150C, WR,	5Cyl, --->4	0321	3390	.0	150C, WR,	5Cyl, --->0	
0322	3390	.0	150C, WR,	5Cyl, --->0	0323	3390	.0	150C, WR,	5Cyl, --->0	
0324	3390	.0	150C, WR,	5Cyl, --->0	0325	3390	.0	150C, WR,	5Cyl, --->0	
0326	3390	.0	150C, WR,	5Cyl, --->0	0592	3390	.0	1511, RR,	67Cyl, --->0	
0600	OSA	.0	QD10->	VSWITCH1 SYSTEM	0601	OSA	.0	QD10->	VSWITCH1 SYSTEM	
0602	OSA	.0	QD10->	VSWITCH1 SYSTEM	B020	SPEC	.0	B020, ATTACHED		
B022	SPEC	.0	B022, ATTACHED		B030	SPEC	.0	B030, ATTACHED		
B033	SPEC	.0	B033, ATTACHED		C70C	SPEC	18.9	C70C, ATTACHED		
C7CC		.0	C7CC, ATTACHED		C7CD		.0	C7CD, ATTACHED		

Figure 6-16 Monitoring I/O activity to an mdadm device array

6.4.5 Using PAV devices with LVM2

LVM2 can also be used in RHEL4 for PAV devices. In our example, the LVM2 volume group and logical volume contains one base (/dev/dasde1) and two alias DASD devices (/dev/dasdf1 and /dev/dasdg1). In Figure 6-17 on page 202, we create the physical volumes using the **pvcreate** command.

```

# pvcreate /dev/dasde1
Physical volume "/dev/dasde1" successfully created
# pvcreate /dev/dasdf1
Found duplicate PV 5BFiCJrRPxEhLXqZXClySSI9tT1TACdT: using /dev/dasde1 not /dev/dasdf1
Found duplicate PV 5BFiCJrRPxEhLXqZXClySSI9tT1TACdT: using /dev/dasdg1 not /dev/dasdf1
Physical volume "/dev/dasdf1" successfully created
# pvcreate /dev/dasdg1
Found duplicate PV W1aYpEb3GPcwnFWPqpEN2o6z8ArG6D3F: using /dev/dasde1 not /dev/dasdg1
Found duplicate PV W1aYpEb3GPcwnFWPqpEN2o6z8ArG6D3F: using /dev/dasdf1 not /dev/dasdg1
Physical volume "/dev/dasdg1" successfully created

```

Figure 6-17 Creating LVM2 physical volumes

Next, we create a logical volume group the **vgcreate** command in Figure 6-18 on page 202.

```

# vgcreate pav_rh /dev/dasde1
Found duplicate PV rcKqqJcKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdf1 not /dev/dasde1
Found duplicate PV rcKqqJcKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdg1 not /dev/dasde1
Volume group "pav_rh" successfully created

```

Figure 6-18 Creating an LVM volume group

The we create a logical volume group using the **lvcreate** command in Figure 6-19.

```

# lvcreate -L 100M -n lvpav_rh pav_rh
Found duplicate PV rcKqqJcKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdf1 not /dev/dasde1
Found duplicate PV rcKqqJcKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdg1 not /dev/dasde1
Logical volume "lvpav_rh" created

```

Figure 6-19 Creating an LVM logical volume

In Figure 6-20, the **vgdisplay** command shows information about the created logical volume.


```

# lvsdisplay
Found duplicate PV rcKqqJCKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdf1 not /dev/dasde1
Found duplicate PV rcKqqJCKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdg1 not /dev/dasde1
--- Logical volume ---
LV Name                /dev/pav_rh/lvpav_rh
VG Name                pav_rh
LV UUID                cyuF4y-Ssd4-PNxQ-46P6-CZy4-W6Ts-rcyCke
LV Write Access        read/write
LV Status              available
# open                 0
LV Size                100.00 MB
Current LE             25
Segments               1
Allocation             inherit
Read ahead sectors     0
Block device           253:2

```

Figure 6-20 Display information about the /dev/pav_rh/lvpav_rh logical volume

In Figure 6-21 on page 203, output from the `lvmdiskscan` command shows duplicate physical volumes have been found by LVM (device nodes /dev/dasde1 and /dev/dasdf1). The duplicate devices correspond the PAV aliases devices.

```

# lvmdiskscan
/dev/dasda [          2.29 GB]
/dev/dasda1 [        99.98 MB]
/dev/dasda2 [          2.19 GB]
/dev/dasdb [          2.29 GB]
/dev/dasdb1 [          2.29 GB]
/dev/dasdc [          3.52 MB]
/dev/dasdc1 [          3.50 MB]
/dev/dasde [          6.88 GB]
/dev/dasde1 [          6.88 GB] LVM physical volume
/dev/sdb4 [         18.63 GB]
/dev/dasdf [          6.88 GB]
Found duplicate PV rcKqqJCKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdf1 not /dev/dasde1
/dev/dasdf1 [          6.88 GB] LVM physical volume
/dev/dasdg [          6.88 GB]
Found duplicate PV rcKqqJCKsHwW8ow4Z6AHEtn2b8v5w19: using /dev/dasdg1 not /dev/dasde1
/dev/dasdg1 [          6.88 GB] LVM physical volume
6 disks
6 partitions
0 LVM physical volume whole disks
3 LVM physical volumes

```

Figure 6-21 Showing the PAV members with `lvmdiskscan`

To remove an alias device from an active logical volume group, it must be first toggled offline by Linux (before being detached by z/VM). To bring all PAV devices offline, the logical volume first be deactivated. As with mdadm, RHEL4 uses LVM2 only for redundant paths (and not concurrent access).

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 207. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Linux for S/390*, SG24-4987
- ▶ *Linux on IBM @server zSeries and S/390: Performance Toolkit for VM*, SG24-6059
- ▶ *Linux on IBM @server zSeries and S/390: Distributions*, SG24-6264
- ▶ *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299
- ▶ *Linux on zSeries and S/390: Fibre Channel Protocol Implementation Guide*, SG24-6344
- ▶ *IBM TotalStorage Enterprise Storage Server Model 800 Performance Monitoring and Tuning Guide*, SG24-6422
- ▶ *IBM TotalStorage Enterprise Storage Server Model 800*, SG24-6424
- ▶ *zSeries HiperSockets*, SG24-6816
- ▶ *Linux on IBM @server zSeries and S/390: Systems Management*, SG24-6820
- ▶ *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824
- ▶ *IBM @server zSeries 990 Technical Guide*, SG24-6947
- ▶ *Linux on IBM @server zSeries and S/390: Best Security Practices*, SG24-7023
- ▶ *Linux on IBM @server zSeries and S/390: Securing Linux for zSeries with a z/OS Central LDAP Server*, REDP-0221
- ▶ *Linux on IBM @server zSeries and S/390: TCPIP Broadcast on z/VM Guest LAN*, REDP-3596

- ▶ *Linux on IBM @server zSeries and S/390: VSWITCH and VLAN Features of z/VM 4.4*, REDP-3719
- ▶ *Running Linux Guests in less than CP Privilege Class G*, REDP-3870
- ▶ *Cloning FCP-attached SCSI SLES9 Linux*, REDP3871
- ▶ *Networking Overview for Linux on zSeries*, REDP-3901

Other publications

These publications are also relevant as further information sources:

- ▶ *Connectivity*, SC24-6080
- ▶ *CP Commands and Utilities Reference*, SC24-6081
- ▶ *z/VM: CP Planning and Administration*, SC24-6083
- ▶ *Getting Started with Linux on zSeries*, SC24-6096
- ▶ *Directory Maintenance Facility Commands Reference*, SC24-6133
- ▶ *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6135
- ▶ *z/VM: Migration Guide*, GC24-6103
- ▶ *Secure Configuration Guide*, SC24-6138
- ▶ *What's new in Performance Toolkit for VM in z/VM V5.1*, GM130637
- ▶ *Linux 2.4 to 2.6 Transition Guide*, LINUX-14MG
- ▶ *Linux on zSeries Device Drivers, Features, and Commands March 23, 2005*, SC33-8281
- ▶ *How to use Execute-in-Place Technology with Linux on z/VM March 23, 2005*, SC33-8283
- ▶ *udev - A Userspace Implementation of devfs* by Greg Kroah-Hartman reprinted from the Proceedings of the Linux Synopsium, July 23-26, 2003
http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- ▶ *Asynchronous I/O Support in Linux 2.5* by Suparna Bhattacharya, et al reprinted from the Proceedings of the Linux Synopsium, July 23-26, 2003
<http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Pulavarty-OLS2003.pdf>
- ▶ "Kernel Korner - I/O Schedulers" by Rob Love in Linux Journal, February 2004
<http://www.linuxjournal.com/article/6931>

- ▶ "Kernel Korner - What's New in the 2.6 Scheduler" by Rick Lindsley in Linux Journal, March 2004
<http://www.linuxjournal.com/article/7178>
- ▶ *IBM Middleware Available on Linux*, found at:
<ftp://ftp.software.ibm.com/software/linux/IBMSoftwareOnLinux.pdf>

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM developersWorks Linux on zSeries
<http://www.ibm.com/developerworks/linux/linux390/>
- ▶ SUSE Linux Enterprise Server 9 home page
<http://www.novell.com/products/linuxenterpriseserver/>
- ▶ RedHat Enterprise Linux home page
<http://www.redhat.com/software/rhel/server/>
- ▶ IBM Software for Linux
<http://www.ibm.com/linux/matrix/linuxmatrixhwz.html>
- ▶ IBM TotalStorage products for IBM @server zSeries and System z9
http://www.ibm.com/servers/storage/product/products_zseries.html
- ▶ XML C parser and toolkit of Gnome
<http://xmlsoft.org/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

Archived

Index

Symbols

- /dev/disk/by-id 167
- /dev/disk/by-path 167
- /dev/zfcp_cfdc 172
- /etc/chandev.conf 72
- /etc/init.d/boot.udev 36, 167
- /etc/modprobe.conf 163, 189
 - alias scsi_hostadapter zfcp 163
- /etc/rc.d/boot.scsidev 167
- /etc/sysconfig/hardware 24
- /etc/sysconfig/hardware/skel 24
- /etc/sysconfig/hardware/skel/hwcfg-zfcp 160
- /etc/sysconfig/sysctl 144
- /etc/udev/rules.d 165
 - 10-udev.rules 165
 - 50-udev.rules 165
- /etc/udev/rules.d/50-udev.rule 33
- /etc/udev/udev.rules 33, 167
- /etc/zfcp.conf 162
- /proc/scsi/scsi 158
- /proc/sys/vm/swappiness 13
- /proc/vlan/config 138
- /sys/block/dev/queue/scheduler 16

Numerics

- 8021q module 71

A

- actcli command 171
- ARP 67
- asynchronous I/O 14
- AUTOLOG1 122

C

- channel 3
- channel path 3
- channel subsystem (CSS) 2
- channel-to-channel adapter 76
- chccwdev command 23, 161, 189
- CHPID 3
 - type IQD 83
- CLAW 82

- cmsfslst command 38
- control unit 3
- COUPLE command 103–104
- CP 2
- custom udev rules 167

D

- dasdview command 39
- Debian 48
- DEDICATE statement 71
- DEFINE LAN command 104
- DEFINE NIC command 102, 104
- DEFINE VSWITCH command 124
 - operands 125
- DETACH LAN command 107
- DETACH NIC command 106
- direct I/O 14

E

- ECKD 43
 - DASD configuration file 189
 - DASD configuration in RHEL4 189
 - DASD configuration in SLES9 189
- EDEVICE statement 46
- emulated FBA device 46
- ESCON channel 76
- ESS 188
 - operation mode 188
 - Bypass Cache 188
 - Inhibit Cache Loading 188
 - Normal Cache Replacement 188
 - Prestage 188
 - Record Access 188
 - Sequential Access 188
- EVMS 192
 - add Segment Manager 177
 - create compatibility volume 182
 - create EVMS segment 179
 - create filesystem 182
 - create storage region 193
 - create volume 194
 - delete logical volume 193
 - multipath array 195

- PAV logical volume 197
- save configuration 195
- evms_activate command 184, 198
- evmsn command 177, 192

F

- FCP 43
 - /sbin/zfcpcnf.sh script 163
 - add LUN 157
 - add WWPN 157
 - bring device online 157
 - configuration files 159
 - configure device using YaST 161
 - create initial ramdisk 163
 - device attributes 158
 - LUN 46
 - masking 168
 - LUN Access Control 168
 - ACT 169
 - ACT template file 171
 - configuration utility 170
 - installation 171
 - operation 170
 - prerequisite software 171
 - privileged FCP unit address 171
 - sample scenario 173
 - manually configure device 161
 - multipathing 175
 - create multipath storage region 180
 - using EVMS 176
 - using mdadm 184
 - udev device naming 165
 - in RHEL4 165
 - in SLES9 167
 - WWPN 46
 - zoning 168
- FICON channel 76
- firewall
 - packet filtering 144
 - routing 144
- futexes 13

G

- getconf command 19
- getfacl command 16
- GNU_LIBPTHREAD_VERSION 19

H

- HiperSockets 83
 - IOCP statements 85
 - z/VM 86
- HiperSockets Network Concentrator 89
- hwup command 25, 161, 189

I

- I/O scheduler 14
 - anticipatory (as) 15
 - completely fair queuing (cfq) 15
 - deadline 15
 - elevator parameter 15
 - noop 15
 - request merging 15
 - request sorting 15
- IBM developersWorks 207
- IBM middleware for Linux 48
- ICF 2
- IFA 2
- IFL 2
- initial ramdisk 190
- insserv command 167
- ip command 139
- iproute2 utility 139
- iptables command 144
- ipv6 module 71
- iQDIO 83
- IUCV 91

L

- Layer 2 126
 - prerequisites 126
 - sample scenario 128
 - /etc/chandev.conf file 137
 - firewall configuration 144
 - IP addresses 131
 - load balancer 145
 - logical diagram 128
 - NFS server configuration 141
 - physical implementation 129
 - start firewall 145
 - virtual Linux servers 133
 - VLAN definitions 130
 - VLAN interface 138
 - VLAN permissions 134
 - VMLAN statement 133
 - VSWITCH controllers 133

- Web server configuration 142
- z/VM system configuration 132
- unique MAC addresses 127

Layer 3 64

LCS device 66

LD_ASSUME_KERNEL environment variable 19

libxml2 package 171

Linux 2.6 kernel 10

- kernel preemption 13
 - access control lists 16
 - choosing 64-bit or 31-bit 10
 - compatibility mode 10
 - extended attributes 16
 - global locks 11
 - major changes 11
 - maximum number of users 11
 - number of attached devices 11
 - reverse page mapping (rmap) 17
 - swap setting 13
- Linux under z/VM performance tuning 49
 - configuring z/VM paging 51
 - Linux guest virtual machine size 52
- Linux Virtual Server project 145
- LinuxThreads 17
- LLC header 68
- load balancer 121, 145
 - direct routing 146
 - IP tunneling 145
 - network address translation 145
- logical CP 4
- LPAR 4
 - dispatching 4
- LPAR-to-LPAR networking 64
- lscss command 22, 192
- lsdasd command 22, 192
- lsscsi command 158
- lstape command 23
- lvcreate command 202
- LVM2 201
- lvmdiskcan command 203

M

- MAC address 64, 127
 - prefix 127
- mdadm 184, 198
 - check device node status 185
 - create device node 185

- mdadm command 185
- mkinitrd command 161

N

- NFSv4 19
- NIC 94
- NICDEF statement 100, 127
 - MACID parameter 127
- non-QDIO 66
- NPTL 17

O

- O(1) scheduler 12
- OSA
 - ARP processing 64
 - checksum offload 66
 - IP assist functions 66
 - Layer 2 64
 - non-QDIO 66
 - non-QDIO mode 62, 66
 - OAT 63
 - Open Systems Adapter-2 60
 - Open Systems Adapter-Express 60
 - OSA-Express 60, 71
 - ethernet adapter 127
 - port name 73–74
 - port sharing 64–65
 - primary router 70

P

- PAV 190
 - base and alias devices 192
 - IOCDS definition 191
 - prerequisites for Linux 191
 - using EVMS 192
 - using LVM2 201
- physical CP 4
- PR/SM 4
- priority queuing 63
- process scheduler 12
 - CPU affinity 12
 - interactive performance 13
- ps command 17
- pvcreate command 201

Q

- qdio mode 61

qdio module 71
qeth device driver 127
qeth group 73, 88
qeth module 71
QUERY ALLOC PAGE command 51
QUERY LAN command 105
QUERY NIC command 105
QUERY PAV command 191
QUERY VIRTUAL OSA command 106
QUERY VMLAN command 127

R

Redbooks Web site 207
 Contact us xi
RHEL4 47

S

s390-tools package 37
s390utils package 37
SAP 2–3
SCSI disks 43
 allocation restrictions 44
 FBA emulation 43–44
 IBM TotalStorage Enterprise Storage Server 43
 size restrictions 44
SET EDEVICE command 46
SET SRM command 54
setfac command 16
Slack/390 48
SLES9 47
subchannel 3
swap device
 DCSS 52
 VDISK 51
sysfs filesystem 73, 80, 88
systool command 158

T

TCP/IP passthru 67
tunedasd command 38, 188
 -c option 188
 -g option 188

U

udevinfo command 32, 159, 166
UNCOUPLE command 106

V

vconfig command 138
vgcreate command 202
vgdisplay command 202
virtual channel-to-channel adapter 91
virtualization 42
VLAN 109
 access port 110
 difference from ELAN 113
 example 111
 IEEE 802.1Q 114
 ISL 114
 isolation 120
 Linux configuration files 140
 port-based 113
 standards 113
 support on z/VM Guest LAN 119
 tagging 114
 trunk port 111, 132
 virtual interface 116
 VSWITCH filtering 120
 VSWITCH support 119
VM 5.1
 DVD installation 43
VM Guest LAN 93
VM scheduler
 dispatch list 53
 dormant list 53
 eligible list 53
 logical CPUs 56
 OPTION QUICKDSP statement 55
 SHARE settings 56
 absolute share 56
 relative share 56
 SRM settings 54
 virtual machine classification 53
VMLAN statement 98, 127
 MACIDRANGE parameter 127
 MACPREFIX parameter 127
VSWITCH
 configuration 121
 IP vs ethernet mode 121
 controller virtual machine 121
 automatically starting 122
 directory entry 122
 PROFILE EXEC 123
 PROFILE TCPIP 123
 QUERY CONTROLLER command 123
 SYSTEM DTCPARMS 123

creating 124
failover support 126

Z

z/VM

memory contention below 2 GB 50
performance monitoring 56

z/VM storage 49

expanded storage 49
recommendations 50

main storage 49

paging space 50

recommendations 51

z/VM V5.1 127

zAAP 2

zipl command 161, 164

Archived

Archived



Linux for IBM System z9 and IBM zSeries



Redbooks

Running Linux on IBM System z9 and IBM zSeries

This IBM Redbook discusses configuring and administering Linux systems running on IBM System z9 and zSeries. It is intended for system administrators and IT architects responsible for deploying Linux servers on System z9 and zSeries servers.

Using and configuring the Linux 2.6 kernel

We examine new features and enhancements of the Linux 2.6 kernel. We demonstrate Linux device configuration using sysfs and udev and discuss running Linux servers under z/VM. Networking options available for Linux on IBM System z9 and zSeries are examined in detail. Configuration of FCP-attached SCSI disks and traditional ECKD storage is demonstrated.

Virtualizing with z/VM

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