



Linux on IBM **@**server zSeries and S/390: ISP/ASP Solutions

Create and maintain hundreds of virtual
Linux images

Running on the mainframe

Managed by z/VM



Michael Maclsaac, Peter Chu
Vic Cross, Chris Curtis
Ivo Gomilšek, Rob van der Heij
Liz Holland, Barton Robinson
Martin Söllig, Simon Williams

ibm.com/redbooks

Redbooks



International Technical Support Organization

**Linux on IBM @server zSeries and S/390: ISP/ASP
Solutions**

December 2001

Take Note! Before using this information and the product it supports, be sure to read the general information in “Special notices” on page 455.

First Edition (December 2001)

This edition applies to z/VM 4.2 (ESP) and many different Linux distributions, but largely an internal development version of SuSE Linux Enterprise Server was used.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2001. All rights reserved.

Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Preface	xiii
The team that wrote this redbook	xiv
Special notice	xvii
IBM trademarks	xviii
Comments welcome	xviii
Part 1. Theoretical considerations	1
Chapter 1. Introduction	3
1.1 What this redbook is	4
1.2 What this redbook is not	4
1.3 zSeries and S/390 architecture overview	4
1.3.1 Modes of running Linux	5
1.3.2 Processor architecture	8
1.3.3 Memory architecture	10
1.3.4 I/O architecture	11
1.3.5 Network architecture	13
1.3.6 Disk architecture	14
1.3.7 Models	15
1.4 Solution applicability	15
1.4.1 Better matches	15
1.4.2 More difficult matches	16
1.5 z/VM and why you want it	16
1.6 Skills and resources required	17
1.6.1 Planning and installation	17
1.6.2 Linux image deployment	18
1.6.3 Maintenance	18
Chapter 2. Sizing	19
2.1 The nature of sizing	20
2.1.1 Sizing vs. capacity planning	20
2.2 Relative system capacity	21
2.2.1 Benchmarks	22
2.2.2 The bottom line	25
2.3 Utilization	26
2.3.1 White space - unused capacity	27
2.4 Example sizing - analysis of company XYZ	33
2.5 Concluding remarks	38
2.5.1 Total Cost of Ownership (TCO)	39

2.5.2	Some trade-offs	42
2.5.3	Final reminder	42
Chapter 3.	Virtual server architecture	45
3.1	Why an architecture is required	46
3.1.1	Components of the virtual server architecture	46
3.2	Disk topology	46
3.2.1	The DASD driver	47
3.2.2	Linux instances with dedicated disk	47
3.2.3	Linux sharing data	49
3.2.4	Sharing disk in memory	53
3.2.5	Minidisk caching	53
3.2.6	Limitations of sharing disk	54
3.3	Memory topology	55
3.3.1	Linux ‘jiffies’	55
3.3.2	Large guest memory	56
3.3.3	Linux swap to VM virtual disk	58
3.3.4	Linux swap files in memory	59
3.3.5	“Hybrid” swap method	61
3.3.6	Sharing Linux memory	63
3.4	Network topology	64
3.4.1	Network devices for Linux	64
3.4.2	Network structure	65
3.4.3	General network considerations	70
3.5	Workload tuning	70
Chapter 4.	Networking a penguin colony	73
4.1	Network devices	74
4.1.1	Open Systems Adapter (OSA)	74
4.1.2	Channel-to-channel (CTC)	74
4.1.3	Inter-User Communications Vehicle (IUCV)	75
4.1.4	Other devices	75
4.1.5	HiperSockets	76
4.2	Resilient IP addressing	77
4.2.1	DNS manipulation	77
4.2.2	Connection balancing	79
4.2.3	Virtual IP addressing	79
4.3	Packet filtering and Network Address Translation	81
4.3.1	What is packet filtering	81
4.3.2	What you can do with Linux packet filtering	82
4.3.3	Planning for packet filtering implementations	82
4.3.4	How packets travel through a gateway	87
4.3.5	Network Address Translation (NAT)	88

4.4	General network considerations	89
4.4.1	How penguin herding is different.	90
4.4.2	Dynamic routing	91
4.4.3	Using the OSA with Linux	95
4.5	Other issues	102
4.5.1	CTC/IUCV device restrictions	102
4.5.2	VM TCP/IP restriction	104
Chapter 5. Security architecture		105
5.1	Sharing isolation and reconfiguration of resources	106
5.1.1	Running Linux in native mode.	106
5.1.2	Running Linux in LPAR mode.	106
5.1.3	Running Linux under VM.	110
5.2	Linux security	116
5.2.1	What kind of protection do you need.	117
5.2.2	Additional security documentation	120
5.3	Cryptography on z/Series	120
Chapter 6. Migration planning		123
6.1	Where to start	124
6.2	What to look for	124
6.2.1	Small scope.	124
6.3	Total cost comparison framework	125
6.3.1	Staffing	125
6.3.2	Hardware.	126
6.3.3	Occupancy	126
6.3.4	Other factors	126
Chapter 7. Backup and restore		127
7.1	Backup methodologies	128
7.1.1	Disaster recovery	128
7.1.2	Logical backup	128
7.1.3	Backup types.	128
7.1.4	Complex application backup	129
7.1.5	In-service backup	129
7.2	Hardware possibilities	130
7.2.1	FlashCopy	130
7.2.2	Point-to-Point Remote Copy (PPRC)	131
7.2.3	IBM 3480/3490 tapes	133
7.3	Software tools	135
7.3.1	Software RAID	135
7.3.2	Network block device	137
7.3.3	VM DASD Dump Restore (DDR)	138
7.3.4	Amanda	138

7.3.5 Tivoli Storage Manager (TSM)	139
7.4 High availability choices with zSeries	139
7.4.1 Loss of a DASD control unit	142
7.4.2 Loss of a S/390 or zSeries server	144
Chapter 8. Performance analysis	147
8.1 Performance considerations	148
8.2 Why measure performance	148
8.2.1 Cost of running applications	149
8.2.2 Controlling costs	149
8.2.3 Controlling the impact of one application on another	150
8.3 Measurement tools	150
8.3.1 Measurement tool used.	151
8.3.2 Screen display.	152
8.4 Measurement data sources	153
8.5 Local vs. global performance tuning	154
8.5.1 Local environment.	155
8.5.2 Global environment.	155
8.6 Linux operational choice.	155
8.7 The CP scheduler.	156
8.7.1 Queue definitions	156
8.7.2 Global controls	157
8.7.3 Local controls	158
8.8 Processor subsystem	159
8.9 Storage subsystem.	160
8.9.1 Storage options	161
8.10 DASD subsystem	161
8.10.1 VM Diagnose I/O.	163
8.10.2 DASD MDC measurement	166
8.10.3 High connect time analysis	167
8.10.4 DASD write analysis	170
8.10.5 DASD/cache	171
8.11 Network performance	172
8.11.1 Network errors.	173
8.12 Server resources	174
8.12.1 Resources by application	174
8.12.2 Resources by server	174
8.12.3 Resources by accounting	175
8.13 Alerts	175
8.13.1 Defining and modifying alerts	176
8.14 Service Level Agreements	177
8.14.1 Availability alerts	178
8.14.2 Cost of measuring availability	178

8.14.3	Availability reporting	178
8.14.4	Measuring service	179
8.15	Measurement function installation	179
8.15.1	ESALPS installation	179
8.15.2	NETSNMP installation	179
8.16	Measurement methodology	180
8.16.1	Measuring Linux applications	180
8.16.2	Measuring Linux server requirements	180
8.16.3	Measuring VM Virtual Machine	182
8.17	Tuning guidelines	182
8.17.1	Paging and spooling (one extent per Real Device Block)	182
8.17.2	Enterprise Storage Server (ESS)	182
8.17.3	Virtual machine sizes	183
8.17.4	DASD format	183
8.17.5	MDC: fair share considerations (NOMDCFS)	183
8.17.6	Swap: RAMdisk vs. virtual disk	184
8.17.7	Timer tick kernel changes	185
8.17.8	Kernel storage sharing	185
Part 2.	Practical considerations	187
Chapter 9.	VM configuration	189
9.1	General VM configuration issues	190
9.1.1	Allocate sufficient paging space	190
9.2	Things to do for new Linux images	190
9.2.1	Create a central registry	190
9.2.2	Create the user ID in the CP directory	191
9.2.3	Allocate the minidisks	191
9.2.4	Define the IP configuration	191
9.2.5	Install and configure the Linux system	191
9.2.6	Register the user ID with automation processes	192
9.2.7	Register the user ID so backups can be made	192
9.3	Using VM TCP/IP as the virtual router	192
9.3.1	Dynamic definitions and the PROFILE TCPIP file	193
9.3.2	Creating the device and link	194
9.3.3	Defining the home address for the interface	195
9.3.4	Defining the routing information	196
9.3.5	Starting the connection	197
9.3.6	Putting all the pieces together	198
9.3.7	Define and couple the CTC devices	201
9.4	Using DirMaint to create Linux virtual machines	202
9.4.1	Why to avoid GET and REPLACE	202
9.4.2	Keeping the user directory manageable	203

9.4.3 Rotating allocation	205
9.4.4 Implement exit for minidisk copy	206
9.5 Using an alternate boot volume	206
Chapter 10. Cloning Linux images	209
10.1 Overview	210
10.2 Installing Linux images the easy way	210
10.2.1 Providing fast access to the install medium	211
10.3 Building a quick start disk	212
10.4 Copying disks instead of doing a full install	213
10.4.1 Copying disks for cloning images	214
10.4.2 Determine the point to copy the disks	215
10.4.3 Locating the files to be customized for each cloned image	215
10.4.4 Reducing the number of changes required	220
10.4.5 When to apply the configuration changes	221
10.5 Sharing code among images	221
10.6 Breeding a colony of penguins	223
10.6.1 Images used in the cloning process	223
10.6.2 Create a patch file for cloning	223
10.7 Linux IPL from NSS	228
10.7.1 Using an NSS with just the kernel	228
10.7.2 Using an NSS as a starter system	230
10.7.3 Picking up IPL parameters	230
Chapter 11. Network infrastructure design	235
11.1 Virtual IP addressing	236
11.1.1 Sample configuration	236
11.1.2 Compiling dummy.o	236
11.1.3 Configuring dummy0	238
11.1.4 Using virtual IP addressing in penguin colonies	240
11.2 Packet filtering and NAT with IPTables	243
11.2.1 What you need to run packet filtering	244
11.2.2 Network configuration for a packet filtering implementation	244
11.2.3 How to permanently enable IP forwarding	251
11.2.4 The first IP Tables rules	253
11.2.5 Checking your filter	255
11.2.6 Using IP Tables	256
11.2.7 How to create a rule	257
11.2.8 Using the inversion ! option	260
11.2.9 Making the rules permanent	260
11.2.10 Sample packet filtering configuration for ISP/ASP	261
11.2.11 Using IPTables for NAT	263
11.2.12 Examples for using NAT in the enterprise and ISP/ASP	265

11.2.13 Additional information	268
Chapter 12. Backup using Amanda	269
12.1 About Amanda	270
12.1.1 How Amanda works	270
12.2 Using Amanda in a penguin colony.	272
12.2.1 Planning for Amanda.	272
12.2.2 Configuring Amanda	272
12.2.3 Backing up with Amanda.	278
12.2.4 Restoring.	285
12.2.5 Reporting.	285
12.2.6 Disaster recovery using Amanda	291
12.3 Backup and recovery scenarios	292
12.3.1 Single file or directory restoration with Amanda	292
Chapter 13. System monitoring	297
13.1 Why measure resource consumption	298
13.2 Charge back method	298
13.2.1 Service Provider accounting and billing	298
13.2.2 Enterprise accounting and billing	299
13.3 What can we measure	299
13.4 CPU time accounting	299
13.4.1 VM accounting	300
13.4.2 Setting up virtual machines for accounting	300
13.4.3 Virtual machine resource usage - record type 01	302
13.4.4 Processing accounting records	303
13.4.5 Linux process accounting	303
13.5 Disk space utilization	304
13.6 Network bandwidth usage	305
13.6.1 An introduction to SNMP	306
13.6.2 SNMP installation	306
13.6.3 SNMP configuration	307
13.6.4 Network bandwidth monitoring	309
13.6.5 MRTG	309
13.6.6 MRTG installation and customization	309
13.6.7 MRTG reporting	312
13.6.8 MRTG reporting for Virtual CTC or IUCV devices.	314
13.6.9 Monitoring multiple Linux guests.	316
13.6.10 Total bandwidth reporting	318
13.7 Availability monitoring.	322
13.7.1 NetSaint	323
13.7.2 Installing NetSaint.	324
13.7.3 Configuring the Web interface.	325

13.7.4	User authorization	327
13.7.5	Configuring NetSaint	329
13.7.6	Starting and stopping NetSaint	333
13.7.7	Using NetSaint	333
13.8	Summary	337
Chapter 14. Web application servers		339
14.1	WebSphere issues	340
14.1.1	Java Virtual Machine (JVM)	340
14.1.2	Objects	340
14.2	Options for running WebSphere	341
14.2.1	WebSphere Application Server for zSeries and S/390 Linux	341
14.2.2	WebSphere Application Server for z/OS	342
14.2.3	Separate servers	342
14.3	Planning for WebSphere Application Server	342
14.4	Test environments	343
Chapter 15. Integrating and replacing Microsoft servers		345
15.1	Using Samba as a domain controller	345
15.1.1	Setting up a Samba PDC	346
15.1.2	Creating a machine trust account	348
15.2	Using Samba in Windows domains	349
15.2.1	Recompiling the latest Samba package	350
15.2.2	Joining the Active Directory	352
15.2.3	Setting up Winbind	354
15.2.4	Setting up /etc/nsswitch.conf	355
15.2.5	Setting up the PAM authentication	356
15.3	Replacing Microsoft Exchange Server	359
15.3.1	The Bynari Insight Server	361
15.3.2	The Cyrus IMAP server	361
15.4	Using AFS in an enterprise environment	362
15.4.1	What is AFS	362
15.4.2	Building OpenAFS	363
15.4.3	Installing OpenAFS	363
15.4.4	Installing client functionality	372
15.4.5	Completing the installation of the first AFS server	377
15.4.6	Installing clients on other servers	382
15.4.7	Installing Windows 2000 OpenAFS Client	386
Chapter 16. z/VM 4.2 Linux features		393
16.1	System Administration Facility	394
16.1.1	Who should use the System Administration Facility	394
16.1.2	Initializing the System Administration Facility	395
16.1.3	Using VMADMIN	398

16.1.4	Creating the initial Linux guest	399
16.2	VM LAN support	401
16.2.1	Creating a VM LAN	403
16.2.2	Using the VM LAN with Linux guests	404
Chapter 17.	Roadmap	407
17.1	Ability to reconfigure CTC and IUCV	408
17.2	SOURCEVIPA equivalence for Linux	409
17.3	DCSS-mapped block devices	410
17.3.1	Sharing between processes	410
17.3.2	Sharing between images	411
17.3.3	Using shared segments	411
17.4	Shadowed disk support	413
17.5	File system access tool for systems management	414
17.5.1	Possible use for the utility	414
17.5.2	Design outline	415
17.5.3	Detailed design	416
17.5.4	Additional points for the implementation	416
17.6	Synergy with CMS Pipelines	417
17.7	Make DirMaint the registration vehicle	418
Appendix A.	Linux Community Development System	421
Components of the system		422
Linux on a mainframe for free		422
Community: the global response		422
Development: what is being tried		424
System: what it is being run on		424
Technical implementation of the LCDS		424
Hardware specifications		424
Network		425
Staff and processes		427
Evolution and lessons learned		431
Summary		435
Appendix B.	Using the hcp command	437
Appendix C.	Using the Linux 2.4 kernel	441
Steps to upgrade SuSE 7.0 to Linux-2.4.5 kernel		443
Using the device file system		447
Related publications		451
IBM Redbooks		451
Other resources		451
Referenced Web sites		451

How to get IBM Redbooks	453
IBM Redbooks collections.....	453
Special notices	455
Index	457

Preface

Now in IBM, we are committed to embracing Linux across everything that we do. Linux runs on our Intel servers, on our Power-based servers, on our iSeries. It runs on our mainframes, on our OEM technology, on our storage servers.

Linux, as we know, is the only operating system of which you can safely say: It will run on architectures that have not yet been invented.

- Dr. Irving Wladawsky-Berger

This IBM Redbook describes how Linux can be combined with z/VM on IBM @server zSeries and S/390 hardware - the mainframe. This combination of hardware and operating systems enables Internet Service Providers (ISPs) and Application Service providers (ASPs) to more efficiently provide services. (We assume a broad definition of ASP, to include production enterprise solutions as simple as file serving.)

When a new resource is required in a world of discrete servers, you can either add the workload to an *existing* server, or add another server to handle the workload. The less costly approach of adding the workload to an existing server often proves to be infeasible because the server lacks adequate capacity—or perhaps because you want to keep only one application on an existing server. As a result, frequently a new server is installed and a *server farm* thus begins to grow in an enterprise.

S/390 and zSeries hardware, microcode and software (especially PR/SM and z/VM) allow physical resources to be made virtual among Linux systems. This allows many hundreds of Linux systems to exist on a single server. Running multiple Linux *images* as guests of VM/ESA or z/VM is a smart choice. Consider the following benefits VM offers a Linux guest environment:

- ▶ Physical resources can be shared among multiple Linux images running on the same VM system. These resources include CPU cycles, memory, storage devices, and network adapters.
- ▶ Consolidating servers by running many Linux images on a single S/390 or zSeries offers savings in space, power consumption and administrative staffing.
- ▶ The virtual machine environment is flexible and adaptable. New Linux images can be added to a VM system quickly and easily without requiring dedicated resources. This also allows for a flexible test environment.

- ▶ The Linux images are able to take advantage of the hardware's reliability, availability and serviceability (RAS) features.
- ▶ VM allows high-speed communication among the Linux images, as much of the networking infrastructure can be virtual and thus performed in memory.
- ▶ VM's minidisk cache and virtual disks allow data-in-memory performance boosts.
- ▶ VM offers a rich debug environment that can be particularly valuable for diagnosing problems among the Linux images.
- ▶ VM's heritage of support for scheduling, automation, performance monitoring and reporting, accounting information and virtual machine management is available for Linux virtual machines as well.
- ▶ An effective way to grow the Linux workload capacity is either to add more Linux guests to a VM system (horizontal growth) or to simply raise the resources available to a Linux virtual machine (vertical growth).

This redbook is divided into two parts. The first part consists of theoretical discussions about installing and managing z/VM and Linux for zSeries and S/390 systems. The second part contains explicit examples of the work we did during the residency to create this redbook.

In our approach, we metaphorically refer to a collection of Linux servers running under VM as a “penguin colony”. Like a colony of penguins, our virtual servers are individuals with their own attributes, sharing resources with neighbors. Some perform better than others, some provide services to other members of the colony, and some try to consume more resources than others. Penguins will be born, live, and (sometimes) die—in our environment, all of these events must be managed.

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.

Michael MacIsaac is a team leader for S/390 redbooks and workshops at the ITSO Poughkeepsie Center. He writes about and teaches classes on Linux for S/390 and zSeries. Michael has worked at IBM for 14 years, mainly as a UNIX programmer. He has led teams that have written Redbooks on OS/390 UNIX, S/390 file and print serving, and Linux.

Peter Chu is an IBM zSeries Techline Specialist in California. He holds a degree in Computer Science from George Mason University. His areas of expertise include Capacity Planning for the zSeries and Linux.

Vic Cross is Linux for S/390 and zSeries Team Leader in the Professional Services division of Independent Systems Integrators, an IBM Large Systems Business Partner in Australia. He has 15 years of experience in general computing, six of which was spent working on S/390. He holds a Bachelor of Computing Science degree from Queensland University of Technology. His areas of expertise include networking and Linux.

Chris Curtis is a Senior Consultant with Satel Corporation, an IBM Business Partner in Salt Lake City, Utah, specializing in security consulting and providing managed security services. He has 12 years of experience in software and systems engineering on platforms of all sizes, from embedded through mainframe. He holds a degree in Computer Science from Westminster College. His areas of expertise include high-availability systems, databases, and advanced software architectures.

Ivo Gomilšek is an IT Specialist for Storage Area Networks, Storage and Linux in IBM Global Services - Slovenia for the CEE region. His areas of expertise include Storage Area Networks (SAN), Storage, IBM eServers xSeries servers, network operating systems (Linux, MS Windows, OS/2), and Lotus Domino servers. He is an IBM @server Certified Specialist in xSeries, a Red Hat Certified Engineer, and an OS/2 Warp Certified Engineer. Ivo was a member of the team that wrote the IBM Redbook *Designing an IBM Storage Area Network*, SG24-5758, and contributed to various eServer xSeries and Linux Integration Guides. He also provides Level 2 support for SAN, IBM eServer xSeries, and high availability solutions for IBM eServer xSeries and Linux. Ivo has been employed at IBM for four years.

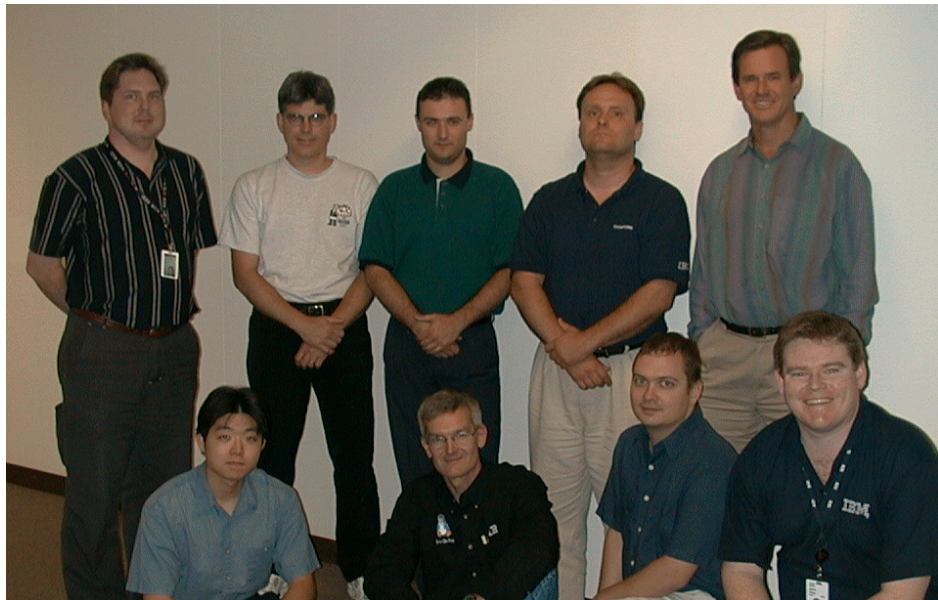
Rob van der Heij is a VM Systems Programmer with IBM Global Services in The Netherlands. He has 20 years of experience with VM Systems Programming and has been working with Linux for S/390 since late 1999. His area of expertise focuses on VM, including running a large number of Linux images on a single S/390.

Liz Holland is a Consulting IT Specialist at the Dallas Global e-business Solution Center and is a member of the IT Specialist Certification Board. She specializes in e-business on S/390 and zSeries, supporting OS/390 WebSphere, Linux, Domino, and UNIX System Services. Her technical papers include a Redpaper on Domino performance on S/390, revisions to the VIF Users Guide, and Redbooks on Component Broker. She began working for IBM in 1980 fixing Selectric typewriters, and began supporting MVS/XA as a PSR in 1983.

Barton Robinson is president of Velocity Software, Inc. He started working with VM in 1975, specializing in performance starting in 1983. His previous publication experience includes the *VM/HPO Tuning Guide* published by IBM, and the *VM/ESA Tuning Guide* published by Velocity Software. He is the author and developer of ESAMAP and ESATCP.

Martin Söllig is an IT Specialist with Enterprise System Sales in Germany. He has 11 years of experience working in the S/390 field. He holds a degree in Mathematics from the University of Hamburg. His areas of expertise include S/390 and zSeries hardware, and major SW products on OS/390 and z/OS, with a special focus on the “new applications” on this hardware since 1999.

Simon Williams is a Senior I/T Specialist with IBM Australia. He has been working on Mainframe Systems since 1988. He specializes in System/390 and zSeries “new technologies” such as Linux, UNIX System Services and WebSphere. His technical publications include an ITSO Redbook on migrating to Domino/390 Release 5, and a Redpaper on how to install Linux for zSeries and S/390 guests under VM.



The team. Standing: Vic Cross, Michael MacIsaac, Ivo Gomilsek, Martin Soellig, Barton Robinson; kneeling: Peter Chu, Rob van der Heij, Chris Curtis, Simon Williams (Liz Holland was not available for the photograph).

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

Terry Barthel, Roy Costa, Al Schwab, Bill White
International Technical Support Organization, Poughkeepsie Center

Bob Haimowitz
International Technical Support Organization, Raleigh Center

John Eilert, Joe Temple
IBM Poughkeepsie

Alan Altmark, Romney White
IBM Endicott

Michael Kershaw
Marist University



Erich Amrehn
IBM Germany

Special notice

This publication is intended to help S/390 systems programmers and system administrators to install and manage z/VM and Linux for zSeries and S/390 systems. The information in this publication is not intended as the specification of any programming interfaces that are provided by Linux for zSeries and S/390. See the PUBLICATIONS section of the IBM Programming Announcement for Linux for zSeries and S/390 for more information about what publications are considered to be product documentation.

IBM trademarks

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

e (logo)® 	Redbooks™
IBM ®	Redbooks Logo 
AFS®	PR/SM™
AIX®	pSeries™
DB2®	RACF®
DFS™	RAMAC®
DirMaint™	RS/6000®
e (logo)®	S/370™
ECKD™	S/390®
Enterprise Storage Server™	SP™
ESCON®	SP2®
FICON™	System/390®
FlashCopy™	VM/ESA®
GDPST™	WebSphere®
iSeries™	xSeries™
Multiprise®	z/Architecture™
MVS™	z/OS™
MVS/XA™	z/VM™
“OS/2®	zSeries™
OS/390®	Lotus®
Parallel Sysplex®	Lotus Notes®
Perform™	Notes®
PowerPC®	Domino™

Comments welcome

Your comments are important to us!

We want our IBM 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 Internet note to:
redbook@us.ibm.com
- ▶ Mail your comments to the address on page ii.



Part 1

Theoretical considerations

In this part of the book, we discuss the theory behind installing and managing z/VM and Linux for zSeries and S/390 systems. For explicit examples of the work we did during this residency, see Part 2, “Practical considerations” on page 187.



Introduction

Then anyone who leaves behind him a written manual, and likewise anyone who receives it, in the belief that such writing will be clear and certain, must be exceedingly simple-minded.

- Plato

The introduction of Linux for the S/390 architecture has brought enterprise-level scalability and reliability to the successful Linux software platform. One area where Linux for IBM [®]server zSeries and S/390 has shown particular promise is in the Internet Service Provider (ISP) and Application Service Provider (ASP) market. These providers tend to run many identical servers, creating huge server farms with rack after rack of Web and application servers. In this competitive market, the cost of maintaining and supporting a large number of servers can have a substantial impact on the company's bottom line. In some geographic areas, floor space is at a premium as well. In this environment, the advantages of the S/390 are significant.

However, deploying an S/390 in an ISP/ASP setting is not a simple undertaking. Consolidating many Linux servers onto one set of hardware has unique implications for managing the individual "machines" for optimum performance.

This chapter discusses the high-level issues involved in deploying Linux for zSeries and S/390 in the ISP/ASP setting, and offers guidelines for determining the suitability of Linux for zSeries and S/390 for a particular environment.

1.1 What this redbook is

The goal of this redbook is to address some of the issues involved in managing large numbers of Linux instances on the zSeries and S/390 architecture. We include information that will be useful to ISP/ASP users who are new to zSeries and S/390, as well as to current zSeries and S/390 customers who are interested in consolidating many servers onto their existing hardware.

1.2 What this redbook is not

A growing body of other documentation already exists on Linux for zSeries and S/390, and we do not duplicate that information here; we do not cover how to install, configure and use Linux for zSeries and S/390, nor do we address specifics of the various available Linux distributions for S/390. These issues are covered in depth in the following IBM Redbooks:

- ▶ *Linux for IBM @server zSeries and S/390: Distributions*, SG24-6264
<http://www.ibm.com/redbooks/abstracts/sg246264.html>
- ▶ *Linux for S/390*, SG24-4987
<http://www.ibm.com/redbooks/abstracts/sg244987.html>

1.3 zSeries and S/390 architecture overview

The IBM @server zSeries is based on the z/Architecture, which is a new, 64-bit superset of the ESA/390 architecture. (This architecture is covered in detail in *IBM @server zSeries 900 Technical Guide*, SG24-5975.) The zSeries architecture is fundamentally different from other common systems in the ISP/ASP environment, and these differences have a profound influence on planning and implementing Linux for zSeries and S/390.

In our redbook, we highlight the ways in which the zSeries differs from Intel and traditional UNIX servers, and the impact of those differences on the Linux for zSeries and S/390 solution for ISPs and ASPs.

This section will also serve as a brief introduction to the architecture for readers new to zSeries. While it is not a complete guide to all the available options and features of the zSeries architecture, it provides a quick tour through the most important aspects that a reader familiar with existing UNIX and Intel servers would need to understand.

1.3.1 Modes of running Linux

Linux is supported on the S/390 G5 and G6 models and the z900 series. It will run G4 and older systems, but as these systems lack IEEE floating-point support, performance will be strongly affected. The following discussion briefly outlines the modes in which Linux can be run, which are illustrated in Figure 1-1 on page 5.

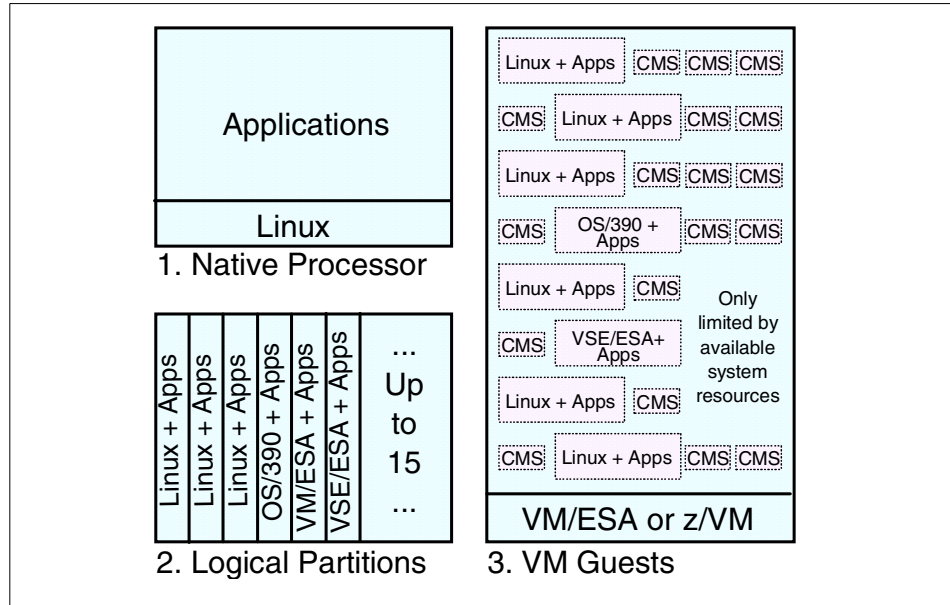


Figure 1-1 Linux for zSeries and S/390 operation modes

Native mode

The first option is to run Linux natively on a S/390 or zSeries server, as illustrated in Figure 1-2 on page 6. This indicates that only one Linux system can be run at a time, with full access to the entire system. This option has no software prerequisites other than the Linux operating system itself, and a single application that has large requirements for memory, I/O access and/or processor power can profit from the full resources of a standalone S/390 or zSeries system.

On the other hand, since there is no other operating system available for the communication between Linux and the hardware, the only possible access to the system prior to booting the Linux operating system is through a hardware console, and debugging can also be done in only this way.

Usually this choice will only be considered for testing purposes on small S/390 systems such as the P/390 or R/390. Because this solution does not exploit the abilities of the hardware and VM to share the resources of the server between several operating systems, it is of no importance for server consolidation in an ISP or ASP environment.

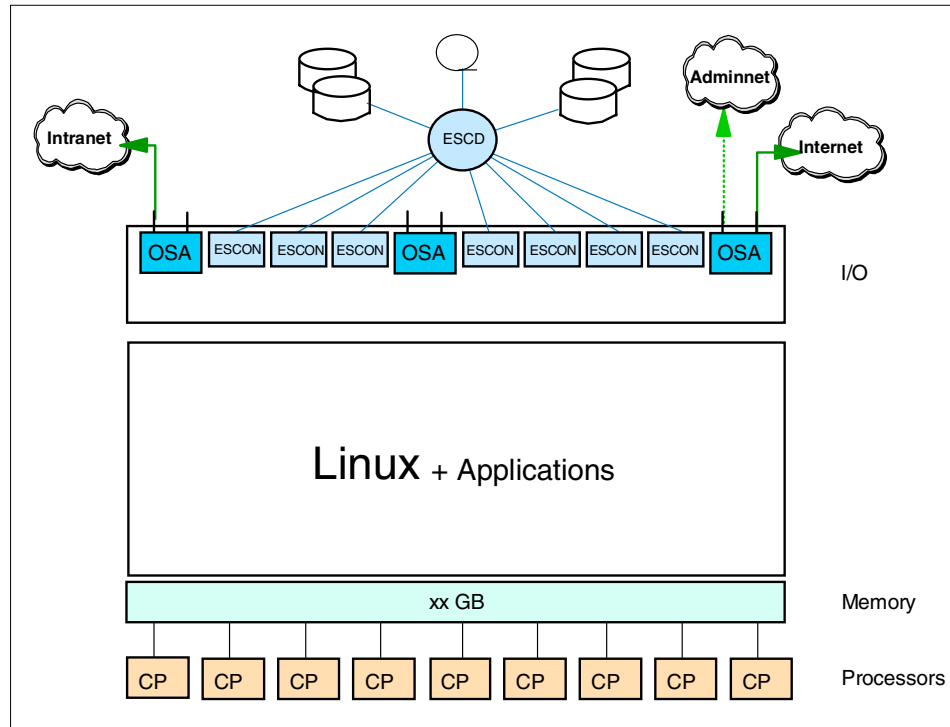


Figure 1-2 Linux for zSeries and S/390 in native mode

Running Linux in a Logical Partition

An S/390 or zSeries server can be divided into up to 15 logical partitions (LPARs), as shown in Figure 1-3 on page 7. These partitions each have their own allocation of memory, and either dedicated or shared processors, as well as dedicated and/or shared channels for I/O operations. The LPARs can operate independently of each other, since the Processor Resource/System Manager (PR/SM) microcode guarantees that operations in one LPAR are not allowed to interfere with operations in another LPAR.

Like running Linux on native hardware, no additional software is required. Additionally, the advantages of virtualization of the S/390 hardware, provided by the PR/SM microcode, can be exploited, which offers a flexibility superior to that of a static hardware solution.

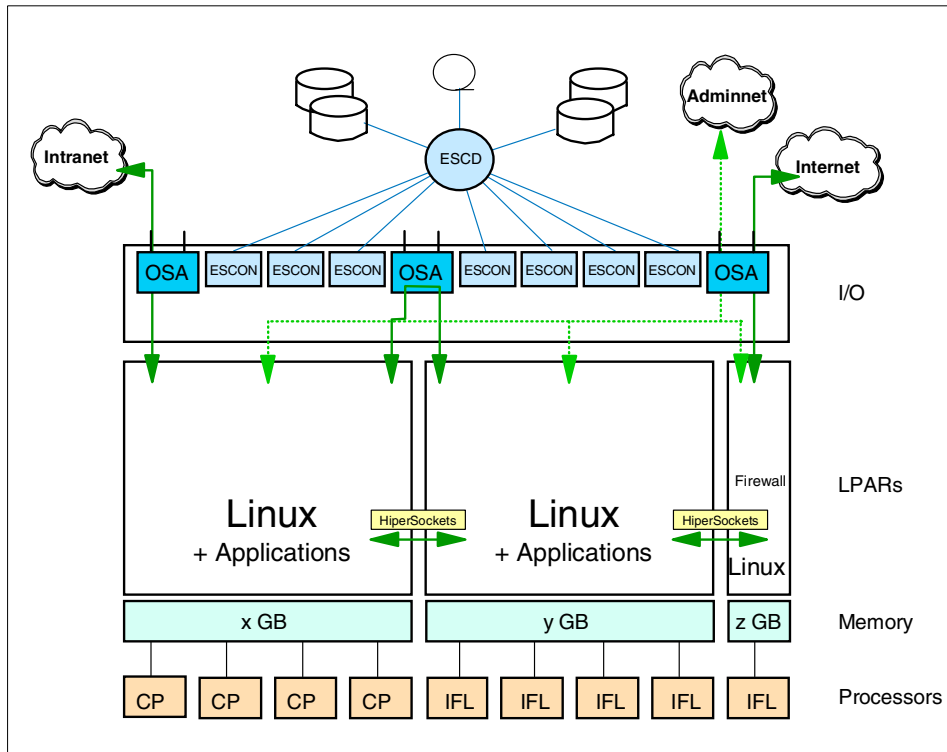


Figure 1-3 Linux for zSeries and S/390 in LPAR mode

Due to the limitation that only 15 LPARs can be defined on a S/390 or zSeries server, and that all definitions, management and debugging of these LPARs have to be done from the hardware console, this choice is also of limited suitability for server consolidation in an ISP or ASP environment. This scenario is mostly used for Linux systems for test or development purposes, when no additional maintenance of VM is wanted.

Running Linux as a VM guest

Linux can also run as a guest operating system in a VM Virtual Machine, which is illustrated in Figure 1-4 on page 8. The VM operating system provides an environment that allows you to create and manage hundreds to thousands of VM guests, in which multiple Linux images can be operated on a single S/390 or zSeries platform. Additionally a repertoire of approved VM tools and utilities is available.

Depending on the hardware platform, there are two versions of VM available: 31-bit addressing VM/ESA, which runs on 9672 or 2064 zSeries processors; or 64-bit addressing z/VM for 2064 zSeries processors, which can also run on 9672 processors in 31-bit mode.

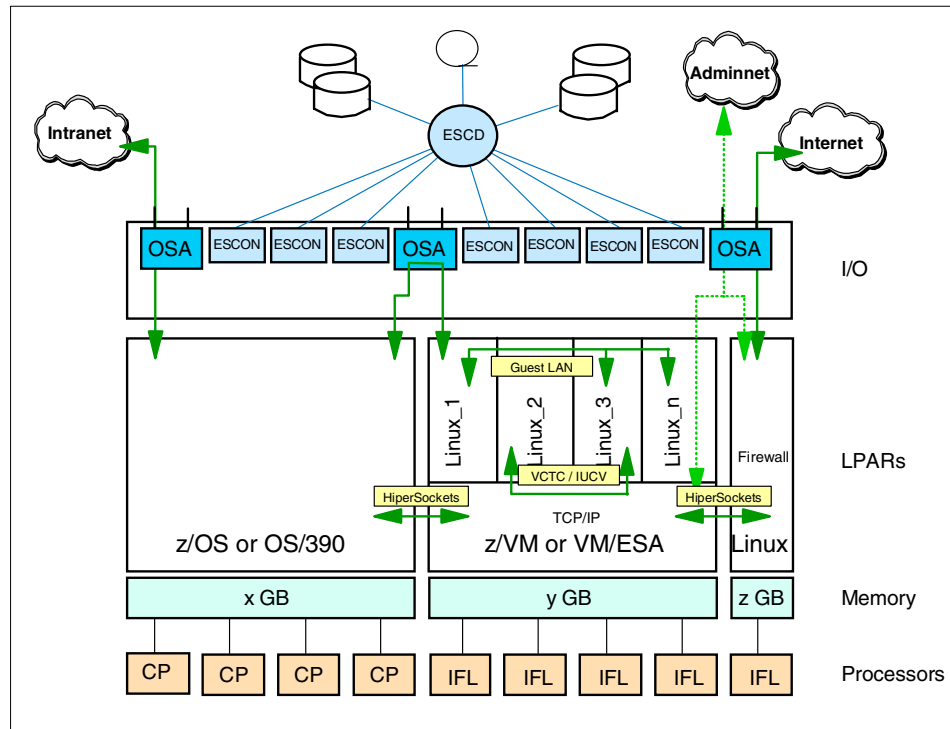


Figure 1-4 Linux for zSeries and S/390 as a VM guest

So if hundreds of virtual Linux systems have to be set up, monitored, managed, tuned and operated, using VM/ESA or z/VM is the only feasible solution. However, a certain operating knowledge of VM is required.

Normally an ISP or ASP is faced with the problem of running multiple servers with the same operating system and similar applications. These can be isolated systems or multitier applications with a database server on a central host system.

1.3.2 Processor architecture

The core of the zSeries architecture is the Multi-Chip Module (MCM). The MCM is an extremely dense package that contains either 12 or 20 Processor Units (PUs), depending on model.

Each PU can be configured to perform one of several functions, including:

- ▶ Central Processor (CP)
- ▶ Integrated Facility for Linux (IFL)
- ▶ System Assist Processor (SAP)
- ▶ Integrated Coupling Facility (ICF)

A PU assigned as a CP can be used to run any supported operating system: OS/390, z/OS, Linux, VM/ESA, z/VM, and others. By contrast, an IFL may only be used to run Linux, VIF and z/VM version 4.

SAPs are processors dedicated to performing I/O processing, offloading that task from the primary processors (CPs or IFLs), and are discussed in 1.3.4, “I/O architecture” on page 11.

The ICF is part of the Parallel Sysplex architecture, which is not currently applicable to Linux.

The associated configuration rules and options are complex, but the basic guidelines are:

- ▶ At least one PU must always be assigned as a CP.
- ▶ At least one PU must be unassigned (spare).
- ▶ 12-PU models have 2 SAPs standard, 20-PU models have 3 SAPs standard.

Thus, for a 12-PU model, there is a maximum of 9 CPs or 1 CP and 8 IFLs; for a 20-PU model, the maximum is 16 CPs or 1 CP and 15 IFLs.

Note: All processors assigned to an LPAR must be either CPs or IFLs; that is, you cannot mix them (therefore any Linux system with IFLs will have at least two LPARs - one for the minimum required CP, plus another for the IFLs).

Each PU is actually a *dual* processor, where both processors execute the same code at the same time. If the results do not match, the PU can signal that it is experiencing errors and remove itself from the active processor pool. When this occurs, one of the spare PUs will automatically be brought into service to replace the failed PU. This processor sparing is completely transparent to the application.

An important element of the MCM and PU design is the massive bandwidth available to each processor. The MCM has a total of 24 GB/sec of bandwidth, resulting in an available bandwidth to each processor of 1.5 GB/sec. This is an order of magnitude, or more, greater than traditional enterprise-class UNIX servers. This is significant because the traditional UNIX approach is to try and minimize I/O operations; using the same approach on the z900 architecture will not make maximum use of its capabilities.

Another important factor to note is that, using the MCM packaging, each z900 machine comes delivered with all 12 or 20 PUs already physically installed. The number of CPs and IFLs purchased are enabled by IBM upon installation. An advantage of this arrangement is that additional processors can be “added” by IBM as needed without taking the system down, up to the maximum allowed for that model type. (Upgrading from 9 total CPs+IFLs to 10 would require changing to the 20-PU MCM, which requires that the system be powered down.) This feature is referred to as Capacity Upgrade on Demand (CUoD).

1.3.3 Memory architecture

Note: In the zSeries and S/390 environment, the term “storage” usually refers to memory, as in central storage or main storage. In the UNIX and Intel environment, a direct access storage device (DASD), or disk drive, is considered to be storage.

It is usually clear from the context which definition is intended, but without being aware of the difference, it can be rather confusing.

Main memory

The z900 architecture supports memory configurations from 5 to 32 GB for the 12-PU models, and 10 to 64 GB for the 20-PU models. The 12-PU MCM has two memory buses and is configured with two memory cards; the 20-PU MCM has four buses and four cards.

Note: The 20-PU models have twice the memory bandwidth of the 12-PU models, regardless of how many active CPs and IFLs are configured.

The physical memory cards have 4, 8, or 16 GB of memory built in, but it is enabled in steps of 1, 2, and 4 GB, respectively. This allows a CUoD capability for memory, up to the maximum available on the installed memory cards. However, CUoD for memory is disruptive and requires a full Power-on Reset (POR).

Processor cache

z900 MCM has both L1 and L2 cache internally integrated. Each PU has 256 KB of L1 cache (128 KB instruction/128 KB data), which is similar to many other processor designs. However, the L2 cache design of the S/390 and zSeries is fundamentally different from typical UNIX server and Linux/Intel server designs. Though the L2 cache is invisible to the user and largely invisible to the operating system, the zSeries cache architecture is perhaps the most critical component of the system for you to understand in planning for, and managing, performance.

The typical UNIX or Intel server has a private L2 cache for each processor. Processors do not share L2 cache contents, and the caches themselves are fairly narrow but deep. For example, the xSeries 380 has up to 2 MB of L2 cache per processor, and each processor in the pSeries 680 has up to 16 MB of private L2 cache.

These architectures do very well when the “working set” (which is the code the processor tends to keep executing over a period of time) fits into this cache, and when the workload is easily partitioned into parallel tasks that can be assigned to a particular processor consistently. The TPC-C benchmark is an example of a workload that is well-suited to this type of architecture.

However, in general these servers have relatively limited memory bandwidth, so that the more frequently cache misses occur and data must be retrieved from main memory, the less the deep, private cache helps. In particular, when the system is heavily loaded and tasks must compete for processor time, each task’s working set must be loaded into the private cache each time that task moves to a different processor. It is for this reason that most SMP UNIX servers are typically sized to run at utilization levels of approximately 40 to 50%.

By contrast, the zSeries has two 16 MB L2 caches which are shared by 6 and 10 PUs each (in the 12-PU and 20-PU models, respectively). These caches are also highly connected to each other and contain a great deal of interconnect logic, in addition to the cache memory itself. With this shared cache, tasks can be moved from processor to processor as needed to manage load without necessarily incurring a cache miss.

In addition, when cache misses do occur, the significantly larger memory bandwidth available minimizes the cost of retrieving data from main memory. Thus the zSeries cache design tends to favor applications that do not have cache-friendly working sets and behavior. The shared cache design also explains why zSeries servers are typically run at much higher utilization than UNIX servers; loads of 80 or 90% are not uncommon.

The MCM, memory cards, Cryptographic Element (CE) chips, Distributed Converter Assemblies (DCAs) (processor cage power supplies), and External Time Reference/Oscillator (ETR/OSC) cards are installed in the Central Processor Complex (CPC) cage.

1.3.4 I/O architecture

Because the options and configurations available for I/O on the z900 are quite complex, a comprehensive description of them is beyond the scope of this redbook. However, we touch upon the subject briefly in this section, since I/O on the z900 is fundamentally different from typical UNIX or Intel server I/O.

Channels and subchannels

The basic building block of I/O connectivity on the z900 is the *channel*. Conceptually a channel can be viewed as a path from the processor (MCM) to a physical device. Indeed, for external devices (DASD, for example) a channel eventually translates to a physical cable or fiber.

A *subchannel*, by comparison, can be viewed as a logical connection from a specific process to some I/O resource. For example, a specific network interface in a process is connected to a subchannel. Many subchannels are carried over a single real channel.

System Assist Processors (SAPs)

SAPs are PUs configured to run specialized microcode, and they manage I/O operations on behalf of a CP or IFL. The SAP selects the appropriate channel to use for the I/O operation, and is also responsible for scheduling and balancing to optimize access to devices.

From the CP's perspective, the I/O request is handed off to the SAP, and the SAP informs the CP when the data transfer to or from main storage has been performed and the I/O operation is complete. The entire process is transparent to the programs running on the CP (e.g. the Linux guests), but it means that the CP is freed from dealing with the details of I/O.

Self-Timed Interfaces (STIs)

The zSeries and S/390 servers do not have a traditional bus architecture like Intel servers and many UNIX servers. Rather, the MCM has 24 Self-Timed Interface (STI) ports. Each STI on the zSeries has a maximum bandwidth of 1 GB/sec full-duplex. An STI port can be connected via copper cable to an STI-Multiplexer (STI-M) card in the z900 I/O cage. Each STI-M card creates an I/O domain of four slots, with one secondary 333 MB/sec STI link per slot.¹ Each slot can hold one of the following I/O cards:

- ▶ ESCON-16 channel cards
- ▶ FICON channel cards
- ▶ OSA-Express (OSA-E) channel cards
 - Gigabit Ethernet (GbE)
 - Asynchronous Transfer Mode (ATM)
 - Fast Ethernet (FENET)
- ▶ PCI-Cryptographic Coprocessor (PCI-CC) cards

¹ The secondary STI link operates at 500 MB/sec for the InterSystem Channel-3 (ISC-3); this is used for Parallel Sysplex operations and is not applicable to Linux.

Note that there is also a Compatibility I/O Cage available, which uses cards designed for the G5/G6 series of servers. In this case the Compatibility I/O Cage is connected via an STI-H multiplexer card plugged into the CPC, providing 333 MB/sec ports. The Compatibility I/O Cage is necessary to use any of the following cards:

- ▶ OSA-2
 - Token Ring
 - FDDI
- ▶ ESCON-4
- ▶ Parallel-4
- ▶ Parallel-3

ESCON and FICON

The Enterprise Systems CONnection (ESCON) channel is the current standard interface for connecting external I/O devices to zSeries and S/390. It is a 20 MB/sec half-duplex serial bit transmission interface carried over fiber optic cables.

ESCON ports are provided by ESCON-16 cards, which provide 15 active ports plus one spare per card. The ESCON-16 cards are always installed in pairs, and ports are activated in ordered blocks of four ports (e.g. if four ESCON ports are ordered, then two ESCON-16 cards will be installed, each with two ports enabled).

The Fiber CONnection (FICON) was introduced on the G5/G6 S/390 and zSeries servers to provide higher bandwidth and increased connectivity. FICON provides a 100 MB/sec, full-duplex serial interface over fiber optic cables. In addition, FICON allows multiple I/O operations to be outstanding at the same time to different channel control units. Taken together, these new features allow one FICON channel to provide the same I/O concurrency as up to eight ESCON channels. FICON cards provide two FICON ports each.

1.3.5 Network architecture

zSeries and S/390 network connectivity is provided by the Open Systems Adapter-2 (OSA-2) and OSA-Express (OSA-E) interfaces. These interfaces provide full TCP/IP connectivity to a broad selection of industry-standard networks. Several OSA-2 cards are supported by the zSeries; the newer network interfaces are supported by the OSA-E cards.

OSA-2

OSA-2 cards are the previous generation of network interfaces, and are supported as standard channel-attached devices.

The OSA-2 family consists of the following cards:

- ▶ OSA-2 ENTR (Ethernet/Token Ring)²
- ▶ OSA-2 FENET (Fast Ethernet) - not supported on zSeries
- ▶ OSA-2 FDDI (Fiber Distributed Data Interface)
- ▶ OSA-2 ATM (Asynchronous Transfer Mode) - not supported on zSeries

For the Ethernet, Fast Ethernet, and ATM interfaces, there are new OSA-Express features available that support these network types, in addition to some new types.

OSA-Express

The OSA-Express cards are the new generation of network interface and are supported on both the G5/G6 series S/390 (one port per card) and the zSeries (two ports per card).

The OSA-Express features introduce a new operating mode, Queued Direct I/O (QDIO). QDIO is a highly efficient data transfer mechanism that significantly increases data throughput. It uses shared memory queues and an internal signaling protocol to exchange data directly with the TCP/IP stack. The OSA-Express features appear as channel types OSE for non-QDIO mode, and OSD for QDIO mode. The following types of OSA-Express features are supported:

- ▶ OSA-Express GbE (Gigabit Ethernet)
- ▶ OSA-Express FENET
- ▶ OSA-Express ATM³

1.3.6 Disk architecture

The zSeries and S/390 have many different options for connecting fixed-disk storage (DASD). In general, they all appear as channel-attached devices using ESCON or FICON ports, though some older systems may use the IBM bus and tag parallel interface.

The current recommended storage device for the zSeries is the IBM Enterprise Storage Server (ESS), also known as Shark.

² The zSeries only supports Token Ring mode for OSA-2 ENTR.

³ It only supports QDIO mode in ATM LAN Emulation (LANE) mode.

ESS/Shark

The IBM Enterprise Storage Server (ESS) is a complete disk storage system that provides up to 13.9 TB of managed storage to all major types of servers. The ESS uses large caches and two 4-way SMP RISC processors to provide extremely high performance. It can be integrated into Storage Area Network (SAN) architecture. Multiple servers can be attached to the ESS using the following types of interfaces:

- ▶ Fibre Channel
- ▶ ESCON
- ▶ FICON
- ▶ UltraSCSI

More information about the ESS is available at:

<http://www.storage.ibm.com/hardsoft/products/ess/ess.htm>

1.3.7 Models

The z900 family of systems is designated by a model number of the form 2064-101. The last two digits of the part number designate how many central processors (CPs) are enabled on the machine; thus the 2064-101 has one CP enabled; the 2064-116 (the largest model) has 16 CPs enabled. There are also capacity models which have part numbers of the form 2064-1C1. These models are intended to be ordered to support Capacity Backup, and are 20-PU models. The -1C6 model, for example, has 6 CPs enabled.

1.4 Solution applicability

One of the side effects of consolidating many discrete server workloads onto one zSeries server is that the applications interact and affect each other in ways that they do not in a discrete environment. Accordingly, some workload types are more appropriate matches and are likely to be “good citizens” of our penguin colony.

1.4.1 Better matches

Based on the characteristics of the z900 architecture previously described, workloads that are excellent candidates for migration to Linux for zSeries and S/390 are ones that exhibit some of the following characteristics:

- ▶ I/O-intensive operations (e.g. serving Web pages)
- ▶ Lightly-loaded servers
- ▶ Custom-tailored system images (that is, no “default installs”)

1.4.2 More difficult matches

By the same token, some types of workloads are not well-behaved in the Linux for zSeries and S/390 environment and thus are poor candidates:

- ▶ Compute-intensive operations (technical computing)
- ▶ Graphics (X-Window system, etc.)
- ▶ Heavily-loaded servers
- ▶ Tasks that check the system clock often (to see if configuration files have changed, for example)

Floating-point calculations can be especially problematic for older machines; prior to G5, there was no support for the IEEE floating-point format in the ESA/390 family. As all Linux floating-point expects IEEE, these machines are especially poor performers for graphics and other computational loads. The G5 family introduced IEEE support via microcode; the G6 family received a significant floating-point performance boost by moving IEEE support into actual silicon.

1.5 z/VM and why you want it

Linux for zSeries and S/390 can be run in several different modes, each with its own advantages and disadvantages. At the present time the available modes are Native, LPAR, and z/VM⁴. These options are discussed briefly in the IBM Redbook *Linux for S/390*, SG24-4987.

We strongly recommend that z/VM be used to deploy Linux in the ISP/ASP environment. There are some additional complexities and skills that will need to be learned, but we believe that z/VM offers an extremely powerful environment which is necessary to successfully manage hundreds or thousands of virtual Linux servers. Some of the features z/VM brings are:

- ▶ Resources can be shared among multiple Linux images, including CPU, memory, storage, and network adapters.
- ▶ New guests can be added quickly, without requiring dedicated resources.
- ▶ There are extremely high-speed virtual networks between guests.
- ▶ It provides centralized storage management and backup.
- ▶ A rich debug and test environment allows images to be created that duplicate production systems, without requiring additional physical resources.
- ▶ It provides comprehensive workload monitoring and control facilities.

⁴ With the release of z/VM 4, we no longer recommend the Virtual Image Facility (VIF) as a configuration option. z/VM version 4 removes many of the limitations present in VIF and has additional functionality specifically designed to support Linux for zSeries and S/390.

1.6 Skills and resources required

While running Linux for zSeries and S/390 has many advantages in terms of potential reductions in staffing requirements and hardware management costs, this does not mean that it is a simple undertaking. Running hundreds or thousands of servers is a complex undertaking in any environment and requires qualified, knowledgeable staff to implement successfully.

Each phase of the deployment of Linux for zSeries and S/390 in the ISP/ASP requires a slightly different mix of skills. To a certain extent, the requirements will vary from company to company, depending on existing business processes and systems in place. The general guidelines offered here may offer some direction.

1.6.1 Planning and installation

This is by far the most critical phase of any ISP/ASP system deployment. The decisions taken during this phase can have a dramatic impact on the future performance and ultimate scalability of the system. This is especially the case for Linux for zSeries and S/390, where the system itself serves as infrastructure for many virtual servers. The infrastructure must be solidly in place and well-designed in order for the guest servers to also be stable and efficient.

For this phase the following individuals should be heavily involved (some roles may be filled by the same person):

- ▶ Linux system administrator
- ▶ Network administrator
- ▶ z/VM system programmer
- ▶ Technical support manager
- ▶ Sales and marketing representative

The Linux system administrator and network administrator fulfill much the same roles they would in any normal multiserver deployment: assigning IP addresses, designing subnets, determining disk layout, and so on.

We believe that the addition of a technical support manager and sales/marketing representative to the planning team is important to the ultimate success of the project. Both technical support and sales departments need to be aware of characteristic advantages and limitations of the virtual server environment to properly inform and support their customers.

One of the most crucial members of this team is the z/VM system programmer. Unfortunately, this is also likely to be a difficult position to staff for most ISP/ASP operations. There are many parts of the system that need to be configured appropriately, and this requires a relatively high skill level with z/VM. Each

system is slightly different, therefore generic “cookbook” approaches are less likely to be successful. It may be possible to acquire this expertise on a temporary contract or consulting basis during the planning phase, and brought in as needed once production operations begin.

1.6.2 Linux image deployment

Once the system is installed and configured, the deployment of new virtual servers is much less complicated. The network administrator will still be responsible for managing IP addresses and network architecture, and the Linux system administrator is responsible for configuring each individual server. However, the Linux system administrator needs to develop some z/VM skills to maximize the efficiencies of the system. Ideally there will be a fair degree of automation in place to create new images, as we detail later in this book.

1.6.3 Maintenance

Maintenance will typically be the domain of the normal operations support staff. In most cases we anticipate that typical UNIX server monitoring and maintenance skills are readily transferable to the zSeries environment, with minimal additional training in some specifics of the z/VM monitoring and control functions.



Sizing

As effective as zSeries and z/VM are in their ability to run large numbers of Linux guests, there are of course limits to how much work a given system can handle. Accordingly, in this chapter we discuss guidelines concerning sizing for your applications and workloads.

This includes topics such as CPs, storage, DASD, and network bandwidth, as well as a general overview of the reasoning behind some accepted sizing methodologies and our assessment of their validity.

2.1 The nature of sizing

First of all, it's important to realize that despite the best efforts of many experts both within and outside of IBM, sizing is an inexact science, at best. A sizing estimate is an *approximation* of the hardware resources required to support a given workload. This estimate is drawn from the best information available at a point in time, in order to obtain a base to work from.

But of course, the actual result will often differ from the results of the estimate, due to factors such as imperfect data (the Garbage-In, Garbage-Out scenario), workload skew not being accounted for in the sizing, custom code, and other customizations.

Add to this the fact that lab conditions do not fully correspond to real world production environments, and it becomes obvious why sizing results cannot be guaranteed. This is not to say sizing cannot be done, of course, but rather to emphasize the fact that there is always a certain margin of error involved.

This is particularly noteworthy in our case because of the differences in the performance characteristics between Linux on the x86 and Linux on the zSeries platforms. As noted in Chapter 1, "Introduction" on page 3, the processor, memory, and I/O architectures of the zSeries platform are substantially dissimilar to the platforms Linux has historically been run on.

Therefore, extra care is needed when performing sizing estimates for running Linux on the zSeries, as the shift in relative capacity of the systems may be larger than expected. Also, one should keep in mind that sizing is really just a "first guess" based on the best available data, and should always be followed up by further analysis.

This also means that you need to understand relative capacity across server architectures, which serves as the basis for sizing. This subject is covered in the next section.

2.1.1 Sizing vs. capacity planning

Before we move on, we need to provide a clarification: some readers may be familiar with the term *capacity planning*. This is a service that IBM has been offering for many years to its customers. On the S/390 platform, this involves getting SMF data (and projected growth rates) from the customer's installation, and then modeling the combination to provide an understanding of how to handle the growth.

However, this is *not* what we mean by sizing.

Capacity planning is relatively predictable because it involves migrating well-understood workloads with measured usage data within the same architecture. However, when we size Linux applications for the zSeries, not only are we moving across very different platforms, but are often doing so with little or no application data available for analysis to support the sizing.

Given these limitations, readers who are used to the methodology and results of capacity planning should reevaluate their expectations for system sizing. To reiterate: sizing is definitely an area where the margin of error tends to be on the large side. But enough about limitations; let's see what sizing can do for you.

2.2 Relative system capacity

To really understand relative capacity, we need to understand the balance of processor power, the internal bandwidth per processor, and the relative ability of the operating system to schedule work in a way that efficiently utilizes resources.

When estimating the relative capacity of servers, most people base their calculations on the correlation between processor speed and system capacity.

For example, many home PC users tend to regard the clock rate as the prime indicator, but this is too simplistic, not to mention misleading, due to microprocessor design differences (witness the recent battle between Intel and AMD, or the older x86 vs. PowerPC conflict, for some high-profile examples).

Other users turn to benchmarks. Some use a processor-oriented benchmark like SPECint, but differences in the memory and I/O subsystems are masked. Some use an industry standard or published application benchmark like TPC-C or SAP R3 SD, but the presence of workload imbalances (called “skew”) and variance in the amount of data handled (“working set size”) make these imperfect measures, as well. We examine benchmarks in more detail in the next section.

Attention: In the mainframe environment, people often used the term MIPS to indicate relative capacity. MIPS is an acronym for “Millions of Instructions Per Second”, but it has also been called a “Misleading Indicator of Processor Speed”, and even “Meaningless Indicator of Processor Speed”.

There is perhaps some justification for these terms because instruction rate has only a casual relationship to MIPS ratings, which are actually relative throughput metrics on a scale where the S/370 158 was considered 1 MIPS. Mainframes have been designed, with a balance of internal bandwidth, I/O performance, processor speed, and scheduling capability, to achieve their MIPS ratings on the workloads called Large Systems Performance Ratios (LSPRs), which have become well understood over time. LSPR ratios are *not* simply a measure of relative processor speed, even though they are often assigned a metric called MIPS.

The term MIPS has caused many people to compare instruction rates of various processor architectures as a way to compare server capacity. But because mainframe MIPS ratings are not really instruction rates, they cannot be used directly in this manner.

2.2.1 Benchmarks

Traditionally, benchmarks have played an important role in migration and workload planning. If you’re going to deploy new applications or port existing applications to a new platform, you’ll want to know how the new system will perform—this where benchmarking comes in.

Well-established organizations such as the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC) have long maintained sets of performance metrics with which to evaluate the many platforms that are commercially available: These organizations are on the Web at:

<http://www.tpc.org>

<http://www.spec.org>

However, these benchmarks are not very useful, for two basic reasons:

- ▶ At the time of writing, IBM has not released any benchmarks for Linux on the zSeries.
- ▶ Benchmarks by nature tend to be artificial, and in some cases misleading.

The second comment may be considered a bit controversial, so let’s discuss further what we mean by “artificial” and “misleading”.

Real throughput by a system is the result of many factors. Processor design and clock speed (collectively referred to here as processor power), internal bandwidth for memory and I/O, and scheduling of work in order to effectively utilize available resources (done by the operating system) are the primary considerations. Therefore, system capacity can only be effectively compared when all these factors are taken into account.

Figure 2-1 shows a graphical representation of this concept.

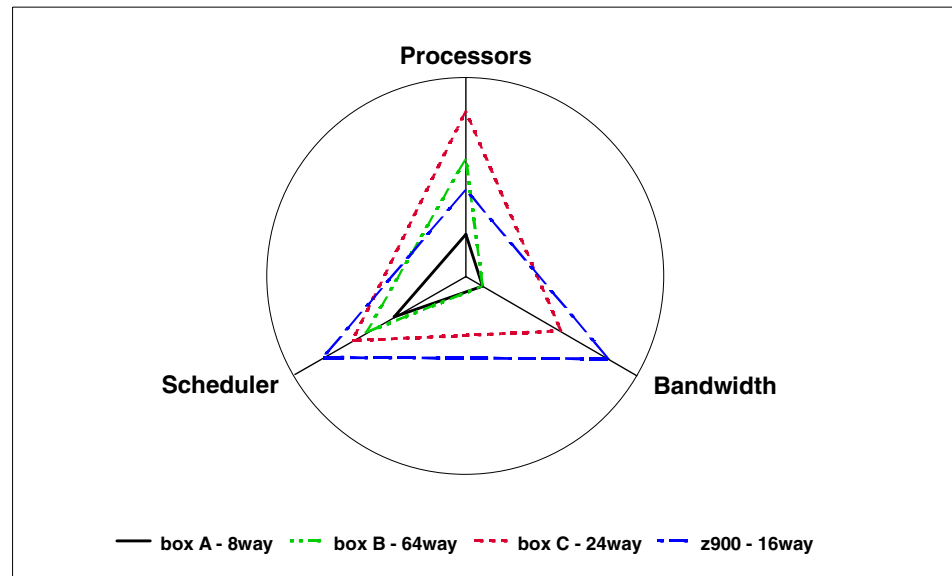


Figure 2-1 Relative system capacity on 3 axes

The misleading part comes into play when we realize that some benchmarks do not look at the “big picture” of the system. For instance, the SPECcpu benchmark, true to its name, focuses almost exclusively on the processor. In other words, it is only looking at the vertical axis of this graph, which represents the processor power of the engines in question, multiplied by the number of engines in the system, as noted in the legend.

But with this approach, bandwidth (which measures the data rate per processor between the cache and memory), and scheduling (which represents processor utilization typical to the particular system), are both ignored. What good is a blazing fast CPU that is sitting idle? The slowest processor in the world can do nothing just as quickly as the fastest.

So if processors, bandwidth, and scheduler are equally stressed by the workload, the relative capacity of the machines can be represented by the geometric mean of the parameters. This is shown in Figure 2-2 on page 25.

If the workload shifts *away* from balance, then relative capacity also shifts. Thus, for CPU-intense environments where the parallel portion of the work is in balance (also known as “skewless”), the relative scaling will shift away from the mean toward the machines with more and faster CPUs. When significant data stress (large “working set”, mixed workload, large user counts, etc.) or skew (data sharing, variable usage patterns, “spikiness” causing workload imbalances) is present in the workload, the relative scaling will shift in favor of machines with higher internal bandwidth per engine and better scheduling.

The result is that the relative capacity of machines will vary significantly from workload to workload. Figure 2-2 on page 25 illustrates the difference between the relative capacity indicated by most commercial benchmarks (the processor bars) and a workload which has enough data stress and/or skew to move the capacity toward the geometric mean of processor, internal bandwidth, and scheduler, using the data from Figure 2-1 on page 23.

You can see that the IBM machines are more balanced than the others, and also that the S/390 does not fare well with CPU-intense work. This follows intuitively from the conflicting views offered by the proponents and opponents of the machine: scaling on the processor bars indicates that the machine is “big”, while scaling by the bandwidth bars indicates that it is “slow”.

The situation is compounded by the fact that most people work with workstations or PCs, where processor speed has a much higher leverage on total capacity to do work. But because servers need to respond to many users, there is much more “context switching” in their workloads than in that of a PC. This means the internal bandwidth and scheduling (the second and third bars) become more critical to the capacity to do work, and to do it quickly.

Understood in this sense, then, system capacity is measured by the size (or, geometrically speaking, the area) of the triangles seen in Figure 2-1 on page 23. No longer are you restricted by a single aspect of the system’s performance.

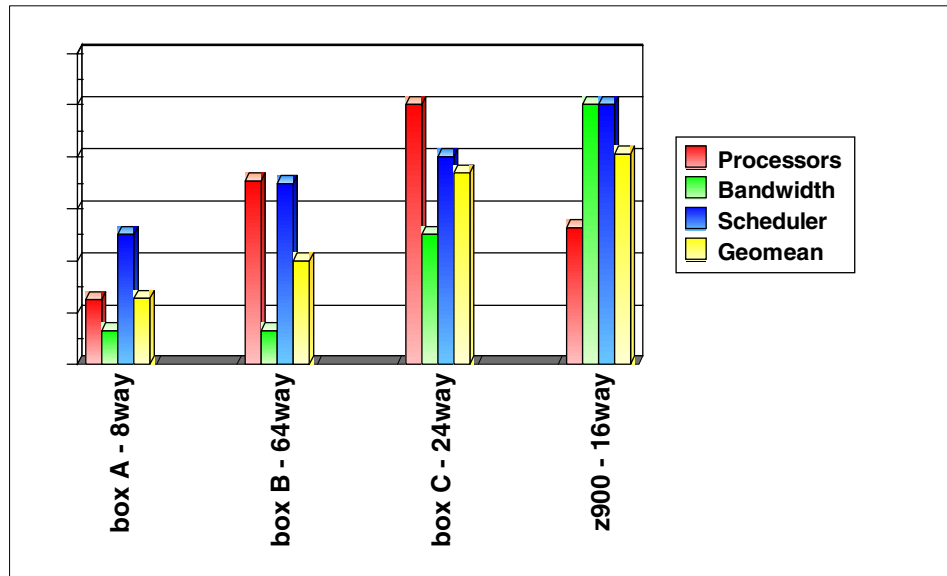


Figure 2-2 Geometric Mean of sample systems

2.2.2 The bottom line

The bottom line of this discussion is that the relative capacity of machines varies with workload. This is most dramatic when it comes to the S/390, but differences also show up between “enterprise” and “midrange” servers, UNIX and NT servers, older and newer servers, etc.

It is true that within a machine type (family), relative capacity is close to the relative processor speed, unless a bottleneck develops. But this is not true for systems with different architectures. Therefore, benchmarks which are reliable indicators in comparing like machines cannot be used to compare different machine types with the same confidence. Since the objective is to make the best possible decision when choosing a server for a given workload, it is important to consider how the characteristics of the workload match the design structure and capabilities of the various machines.

For these reasons, it is impossible to simply position the various servers in a list from largest to smallest. IBM's large commercial servers have more robust scheduling and higher internal bandwidth per processor than other servers, allowing them to maintain high processing rates in the face of skew. As a result, they will have higher capacity than their competition on many real workloads.

To address the issue of workload dependence, IBM has put resources in place to provide sizing, capacity analysis, and capacity planning assistance. If you need a sizing performed, contact Techline at (888) 426 5525, or fill out a TechXpress form on the following Web site:

<http://dalnotes1.sl.dfw.ibm.com/atss/techxpress.nsf/request>

2.3 Utilization

Linux on zSeries is really about doing the work of many smaller servers on one (or a few) larger servers. This is typically done to gain advantages in total cost. In most scenarios, the advantage is realized by reductions in the growth of floorspace, power, people, network complexity, etc. It turns out that server consolidation is most viable when there is some inefficiency in the current operation. This can come about in a variety of ways, and no two total cost scenarios will be exactly the same. However, returning to the capacity effects of server consolidation, we observe the following.

Clearly, we are not going to be able to show consolidation viability if the application is CPU-intensive enough to show the S/390 engines at a disadvantage. However, this does not turn out to be the biggest lever. Utilization can have up to a tenfold (or more) leverage on relative capacity. This is because distributed servers are designed for the individual peaks of the various workloads in question. This level of capacity is an overdesign for the composite peak of the workload, unless all workloads peak simultaneously, which almost never happens.

Furthermore, as we have seen, response time on some servers and workloads can be very sensitive to utilization. When servers are inexpensive, there is a tendency to buy more when the response time starts to grow, regardless of the actual load on the machine. In some cases, this can happen at 40% or 50% utilization, resulting in many machines with very low average utilization.

Then there are all the other servers which tend to go with a production system - backups, test, development, and infrastructure servers such as file, print, DNS, security, system managers, and so on.

When all these are added together in a composite utilization picture, the utilization can be quite low, even at the peaks.

2.3.1 White space - unused capacity

Since utilization is a major factor in assessing the effect of consolidation, we need to understand it better. We also need to know its inverse, which is the “white space”, or unused capacity of a system.

We start with definitions. This is particularly important because utilization is defined by how we measure it. All machines contain a counter that is incremented every cycle. This counter is used to generate a running notion of time within the system. The hardware counter is not infinitely long, but it is typically rolled over into a software counter kept in storage that is sufficiently long that we don't have to worry about it rolling over. Most machines have another counter which increments only on cycles when the processor is busy (this includes when it is busy waiting for memory on a cache miss). This counter is also extended by software and the count of busy cycles is kept in memory. Utilization is defined as the change in the “busy count” divided by the change in the total “cycle count”.

Since cycles occur at a fixed rate in time, the change in cycle count is a measure of a time interval. Thus, the software that generates utilization data operates by periodically checking the change in the busy count. The cycle count in the denominator is determined by how often the software is run. This in turn is controlled by starting the data gathering code at regular intervals by dispatching it when the change in the cycle counter indicates that an interval has completed. When it runs, the data gatherer reads both counters, does the division, and stores or displays the result for use to look at.

White space is simply defined as $1 - utilization$. Thus, utilization is a statistic which is always an average or probability. This is why we need to be careful when we talk about peak and average utilization. The shorter the interval used to gather the utilization data, the more the utilization number looks like either one or zero. This is because if we get to the ultimately short interval of one cycle, the machine is either busy or it is not. Moreover, the shorter the interval used to gather utilization data, the more impact the gathering of data has on what we are measuring. In the most extreme case, the software that does the gathering keeps the rest of the system from running and the utilization becomes 100%, even though the throughput has gone to zero.

Because of these factors, utilization graphs will look spikier when the interval is short, and smoother when it is long. Because it is easier to see the white space on smoother graphs, and because the impact of gathering statistics on longer intervals has lower impact on the system, we typically look at 1- to 15-minute intervals when gathering statistics.

Is this enough? See the utilization profiles in Figure 2-3 on page 28, which all have 50% white space.

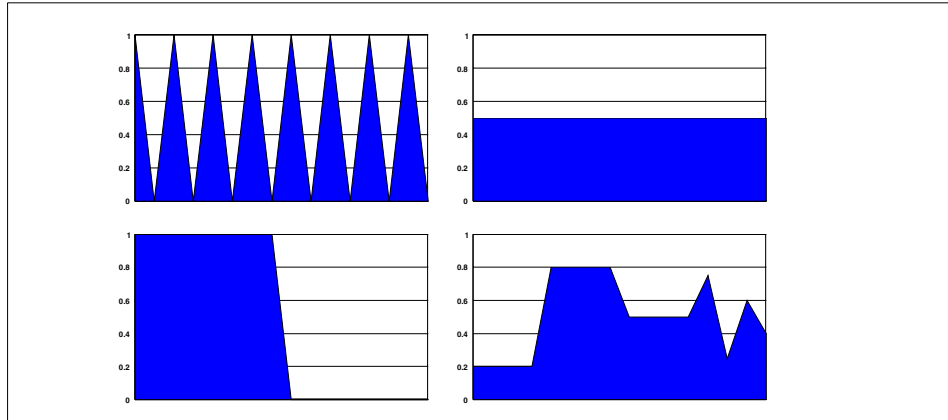


Figure 2-3 Sample utilization charts with 50% white space

Obviously, the average utilization is not enough, because the peak of each workload is also of interest. On one hand, we need to have enough capacity to handle the peak, but we'd also like to minimize the white space in order to run more efficiently. In each of these cases, 50% of the capacity goes unused.

When we build a composite of all 4 workloads, we get the graph shown in Figure 2-4:

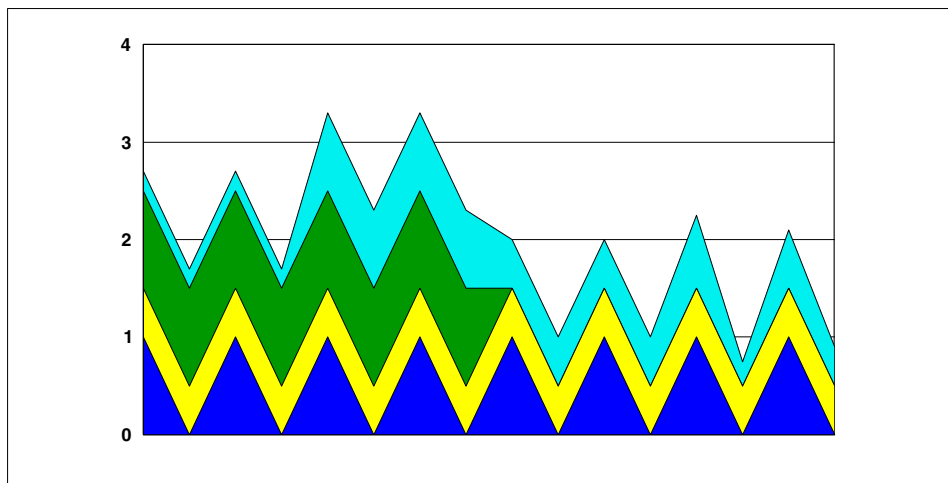


Figure 2-4 Utilization after consolidation - also 50% white space

Here we see that the peak workload is about 80% of the total individual configured capacity. If this peak holds up, we probably are not getting a large lever from consolidating. However, often such peaks occur because of scheduled batch windows for data loads, reports, backups, etc, which can be prioritized or rescheduled using white space in different periods, thus smoothing the curve and reducing the peak.

The example of 30+ servers shown in Figure 2-5 illustrates this case; the peaks are identified as database backups that are all scheduled at the same time on the distributed solution, but can be staggered to reduce the peaks on the consolidated machine. In this example the distributed solution has about 62% white space, meaning that over half of the configuration goes unused.

Some of this occurs naturally, even if the individual systems are efficiently configured, because individual systems must be configured for each workload's peak; this is shown as the grey space on the chart. Only 30% of the distributed configuration is "headroom", even though 60% of the composite is white space. This means that even if 30% headroom is maintained, the composite can be built with 30% less total capacity.

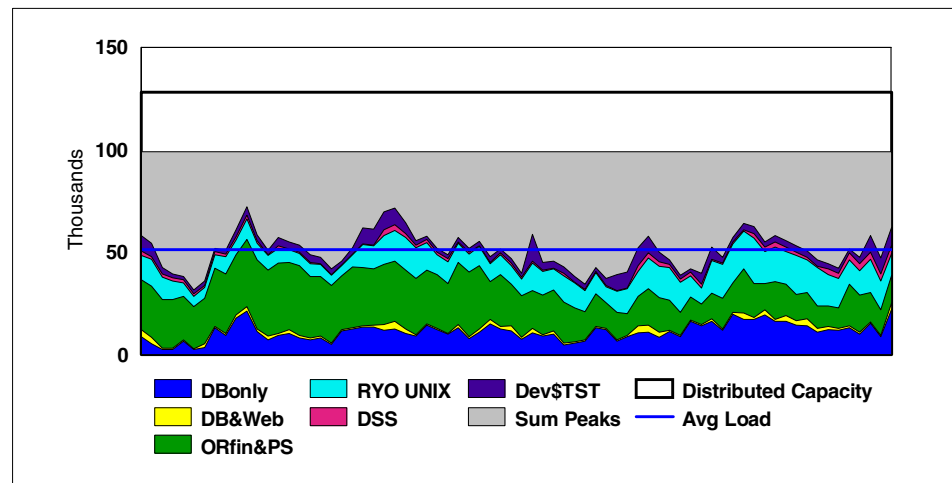


Figure 2-5 Using Virtual Servers

In this particular case, the composite utilization is actually quite high for a distributed solution. In many other cases, there is even more white space. For example, Figure 2-6 on page 30 shows the composite of 147 servers in IBM's "universal server farm" run by IGS as part of their Web-hosting operation. Here, the composite peaks at 13%—this means there's over 87% white space in the configuration.

Figure 2-6 Web servers - consolidation candidate

Sources of white space

So where do such large inefficiencies in the deployment of computer power come from? White space comes from six main sources:

- ▶ Spikes
- ▶ Headroom
- ▶ Redundancy
- ▶ Fragmentation
- ▶ Partitioning
- ▶ Skew

Spikes

“Spikes” in a workload result from the variance of its demand for service over time. If the demand is very variable and the work is high priority, it is difficult to use the white space, because the work that fills the troughs must be overridden by the high priority spike.

This has several implications. First, when white space is caused by spikiness, there must be a scheduler in place that can dispatch work according to pri-

Rapid context switching of large context changes is a characteristic of workload mixing that occurs when work is added to a system to use white space. The S/390 hardware design is particularly well-suited to this environment. It is the underlying shared L2 cache and high memory bandwidth per processor which enables this, and by extension, the ability to run virtual machines.

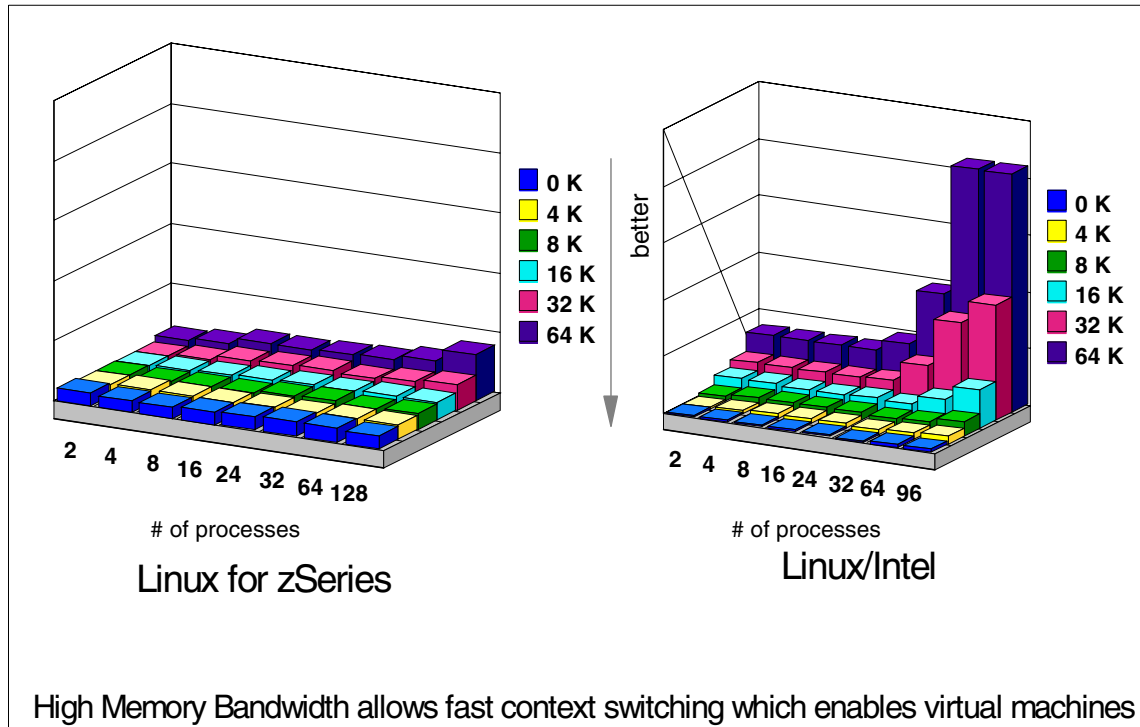


Figure 2-7 Context Switch Profile

Headroom

Many distributed systems experience an “accelerating ramp” in response time as the load grows. Under light load, they exhibit very good-to-excellent response time, but this tends to slip as load is applied. Users often use perceived response time instead of utilization to understand how heavily loaded a system is. In some workloads, the loss of performance occurs at 50% or less utilization. This means that additional capacity is brought online at relatively low utilization.

Refer to Figure 2-8 on page 32 for an example of this concept. In this graph, the 250 MHz 32-way machine achieved 25% faster turnaround than an S/390 G4 2X 10-way sysplex on this group of batch jobs, employing 60% more processors at low utilization. However, as the load was doubled and then tripled, this advantage was not maintained.

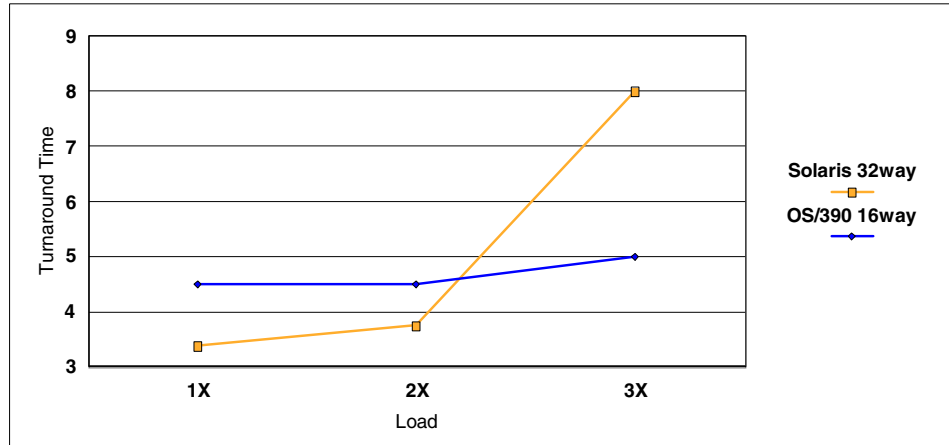


Figure 2-8 An SAS workload capacity curve

Redundancy

Many distributed solutions contain redundant systems as backups for availability, test, development, integration, etc. These extra non-production systems add capacity but are sometimes not used at all, or have very low utilization, or peak in utilization at different times than the production systems.

Fragmentation

Workloads grow as a continuum, whereas capacity is added in discrete quantities. As a result, whenever there is enough capacity configured, there is some amount of white space just based on the difference between the quantum step and the continuous growth in the workload. When the solution is distributed, the quanta are often whole systems, or the capacity is a set of individual sub-workloads, each continuously growing and having its demands met by its own quantum steps in capacity. This results in fragmentation of the resulting white space. By combining the loads, only one quantum of workload is only partially filled, instead of several.

Partitioning for integrity and isolation

Sometimes the user will create many small systems so that a failure will only impact a small subset of users; this is known as partitioning. Work is also partitioned, in order to create isolation and prevent interference. Finally, partitioning is often used in place of prioritization, so that each sub-workload gets the full attention of the machine upon which it is run.

The end result of any of these actions is white space. Partitioning also leads to the replication of data, often requiring off-shift batch updates which can drive higher peaks than the daily online workload. This leads to white space outside the batch window.

Skew

When multiple processors or systems are working in parallel, they are almost never kept uniformly busy. Much effort is expended in balancing workload, but in the end, none of the various methods for doing so work perfectly. The result is that white space emerges on some machines, when others are heavily loaded.

Regardless of the source, white space represents wasted compute power. Server consolidation is a means by which to remove some of the waste and run more efficiently.

2.4 Example sizing - analysis of company XYZ

Now that most of the theory for sizing has been discussed, let's take a look at sizing a given configuration.

This is the setup we inherit at the fictitious company XYZ.

Table 2-1 Setup for company XYZ

Function	Server type	# of servers	Average utilization
File Server	Compaq DL380	10	10%
DNS Server	Sun 5S	4	15%
Firewall	Sun420R	2	15%
Web Server	Sun280R	10	15%

Note: Before we go through each of the elements of sizing, keep in mind that many of the calculations we base our sizing on are confidential and cannot be explicitly written out. There are several reasons for this, the most important being we do not want to set a “standard” for how to size. Although this may seem counterintuitive, when one considers how many variations there can be in hardware (notice that our setup is fairly small and homogeneous, which will not always be the case), software, and workload, one can see why we cannot endorse a generic formula with some constants and a few variables. Since each situation is different, each sizing will have to vary accordingly. The intent here is to illustrate the principle, and not the specific implementation.

CPs

The engines of a system are always considered its primary attribute—even in the traditional mainframe environment (where emphasis on the processor is not as heavy as it is in the PC world). Accordingly, most of our efforts will be concentrated here.

The theory is deceptively simple: you measure how much load is on the current system (in other words, how heavily utilized it is), then translate this number into a zSeries equivalent via a workload factor (WLF), and that's that. The formula which represents this calculation is:

$$\text{MIPS needed} = \% \text{Utilization} * \text{Current Capacity} * \text{WLF}$$

Making comparisons

As mentioned previously, architectural differences between processor families make comparisons between them very difficult. The clock speed competition between chip manufacturers has escalated recently, and this competition helps underscore the point. What makes it even more remarkable is that some of these chips share the same architecture. How much harder it is, then, to compare processors from completely different architectures.

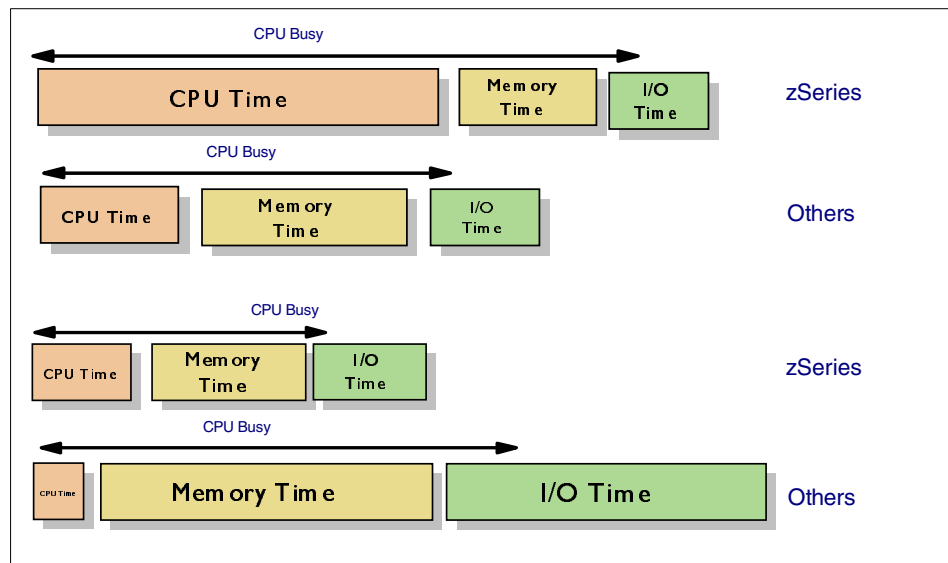


Figure 2-9 CPU-intensive work (top) vs. data-intensive work (bottom)

This is especially problematic for the S/390 and zSeries processors, since they are much better at data-intensive workloads such as databases, Business Intelligence, On-Line Transaction Processing, and “cache-killer” applications than they are at CPU-intensive tasks such as benchmarks and graphics rendering (see Example 2-9).

Nevertheless, that is the task before us now. Let’s examine each component of the equation separately.

Utilization

Determining utilization is the most straightforward element. Since running Linux under VM means consolidating discrete, underutilized servers, it is important to find out the current usage. There are a variety of tools for the job, depending on the platform you’re starting out on (Norton Systemworks for Windows servers, for instance). The key here is to have a duration that is long enough to give a realistic average utilization, and to identify what the peak time characteristics are, as explained in the previous section.

Remember that the more accurate the reading, the better off your sizing will be.

Current capacity

Determining current capacity should be fairly simple, as well. What we are referring to here is the TPC-C benchmark “transaction per minute” (tpm) numbers for each machine as they are configured. Obviously, the way these numbers reflect reality will vary somewhat depending on the workload, but they are good starting points.

If your workload is usual in some way, you are always free to compile your own data and come up with an unofficial tpm number, or even “tweak” the official numbers so they are more representative of your situation. Just be aware of the risks involved should these numbers be off.

Workload factor

The challenge here is to find a conversion factor to take us from our utilized tpm rating to the traditional S/390 MIPS rating.

Note: As already discussed, using MIPS numbers is not the best way to gauge performance for S/390 and zSeries machines. For performance measurements, LSPR is much better. However, keeping the end goal in mind, all we're after here is a conversion from tpm to MIPS so we can identify how many processors are needed to run the same workload as our starting environment on our target S/390 or zSeries machine. This is not to be taken as an endorsement of the MIPS rating in general.

Much easier said than done, of course; the pitfalls are many. However, we'll limit our discussion to the two primary issues:

- ▶ Usage & workload - Chances are you're extremely tired of seeing this by now, but the fact remains that you must consider the application and work that is being ported, in order to size it properly. This is not idle speculation we're engaging in. In terms of tpm's, the relative capacity of a S/390 engine can vary from around 20 tpm/MIPS in CPU-heavy benchmarks to well over 200 tpm/MIPS for Samba workloads.
- ▶ "Generation Gap" - At the time of writing, there are three generations of IBM eServer zSeries processors we are looking at running Linux under: the G5, G6, and z900 engines. Needless to say, there are numerous differences between them. For instance, the IEEE floating point instructions are implemented in microcode in the G5 chips, but are done via actual circuitry in the G6 processors. The resulting performance increase is somewhere in the neighborhood of an order of magnitude. Therefore, the S/390 or zSeries processors you'll be migrating to is also an important consideration.

Memory

Also known as *storage* to the traditional mainframe folks, the zSeries machines are at a definite advantage when it comes to memory. Compared to its Intel and UNIX counterparts, the zSeries memory architecture is much more mature and efficient.

Consider Figure 2-10 on page 37. You can see the large breaks in the bandwidth as the L1 and then the L2 cache sizes are exceeded by the amount of data to be moved. Converting these results to tpm/MIPS yields the results shown in Figure 2-11 on page 37. Thus we can surmise that the tpm/MIPS grows with working set size, which drives cache misses and stressing the internal bandwidth shown here.

Refer to 3.3, "Memory topology" on page 55 for a more detailed analysis of this topic.

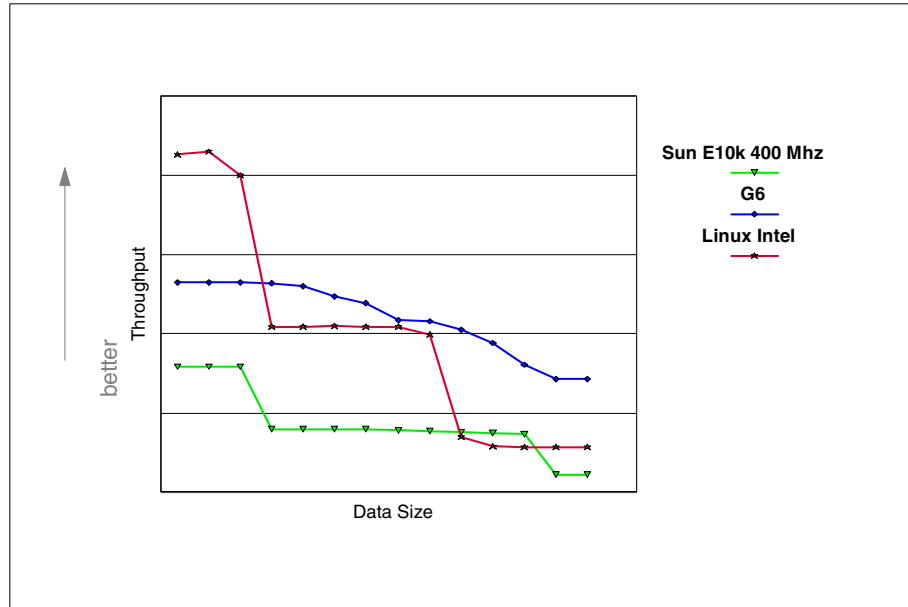


Figure 2-10 Memory read bandwidth

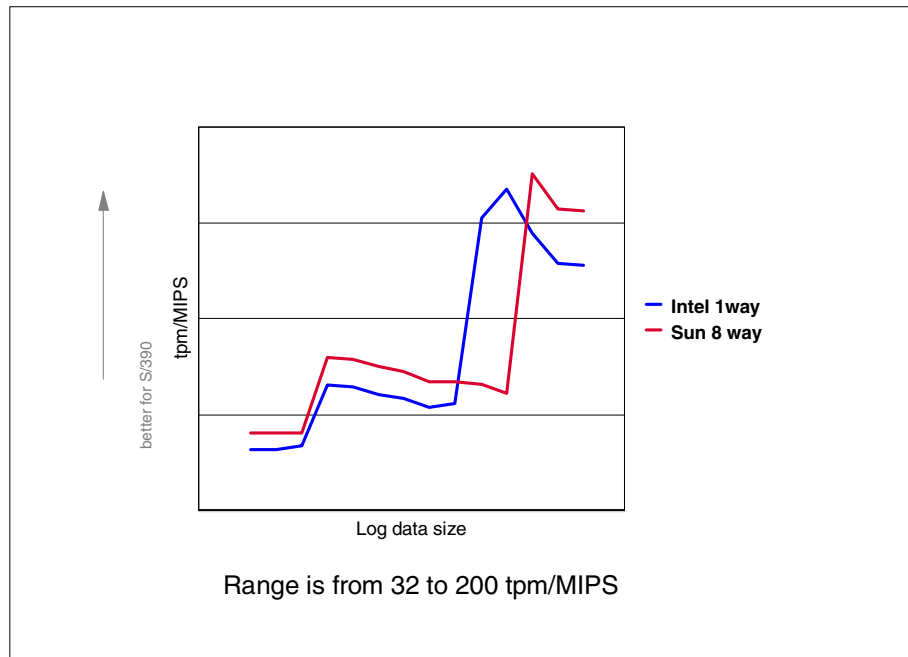


Figure 2-11 tpm/MIPS vs. working set size

DASD, channels and network bandwidth

I/O is heavily dependent on application characteristics. The situation on the source platform can be assessed using tools like Netbench, but a porting discipline has not really been developed for these factors yet. Refer to 3.2, “Disk topology” on page 46 and 3.4, “Network topology” on page 64 for a discussion of how these systems work.

2.5 Concluding remarks

In general, we are looking at a range of 30 to 200 tpm per MIPS of relative capacity. Data-intensive workloads fall at the high end of this range, while CPU-bound workloads fall at the low end. The tpm per MIPS also can go up about 30% between a zSeries uniprocessor solution and a 16-way solution.

The art of sizing is in measuring the utilization properly, and choosing a WLF to use.

The TPC policies do not allow for the publication of unofficial TPC-C results or estimates, and IBM supports this policy. No single benchmark can accurately represent the way a system will perform in a specific customer environment. IBM sizing teams use multiple sources of information, including confidential estimates of benchmark results to deliver the best proposal possible for a given situation.

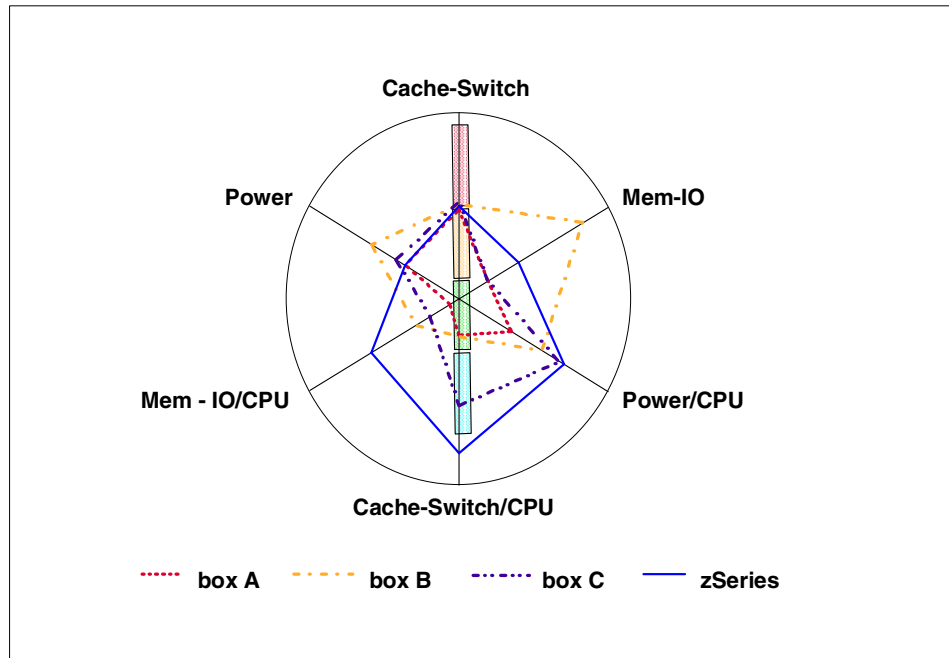


Figure 2-12 Capacity Profile - a more complete view

In Figure 2-12, you can see that box C is stronger on cache and internal bandwidth. You'll also notice that both boxes A and B are weak in "parallel hell" (for example, highly integrated transaction systems which requires significant sharing and synchronization which cannot be buried by redundancy) and strong in "parallel nirvana" (where the serial portion of the execution is small). The zSeries is ahead of the rest in parallel hell but weak in parallel nirvana, particularly if the work is CPU-bound. Box A is a relatively small machine when looked at this way.

2.5.1 Total Cost of Ownership (TCO)

Now that we've established how to quantify relative capacity, we need to understand how to compare costs between servers. It is not enough to tally price/performance, but rather it is necessary to understand the total cost associated with each server type. The reason for this is that in raw capacity, you can always put together enough small machines with low prices to show a better price and capacity advantage for a distributed solution. Therefore, simply looking at \$ per tpm as suggested by the published TPC-C benchmark results will be misleading, even if the tpm/MIPS (or tpmA/tpmB for that matter) is adjusted to account for different architectures.

This is because differences in server architectures and implementation go beyond capacity. Even *within* a family of servers, differences between clusters and a large machines appear. These variables drive differences in a variety of non-acquisition costs, such as occupancy, network engineering, operations (people), and outage costs. Therefore, to understand which server type should be deployed, it is necessary to look at the Total Cost of Ownership (TCO) of competing solutions.

However, even this is not a simple matter of adding up well-known average values to get a “typical” result. The problem is that there is a very large variance in each of the variables involved. There are usually unique and significant cost factors which can only be listed as “other costs” in any one-size-fits-all method of computing the TCO. Having said that, the following is an attempt to build a TCO model which fits most situations.

The Total Cost components

Here is an outline of the components of Total Cost:

1. Time
 - a. How far into future
 - b. How often are upgrades
 - c. How many upgrade steps in Study
2. Hardware
 - a. Price
 - b. Discount
 - c. Maintenance
 - d. Financing
 - e. Overlap, Deployment, or Depreciation cost
 - f. Node Count
 - g. Rack Count
3. Software
 - a. Price
 - i. Per seat
 - ii. Per CPU
 - iii. Per Unit of Capacity
 - iv. Per Box
 - v. One time or Monthly License Fee
 - vi. Cost per Rack (includes power connection)
 - b. Maintenance
4. Occupancy
 - a. Burden per SQ ft. (Rent, facilities, lights, heat, cooling, etc.)
 - i. Rent
 - ii. Facilities
 - iii. Lighting
 - iv. Heat/Cooling

- v. Clearances (Sq. foot per Rack)
- b. Power
 - i. \$ Per Kilowatt Hour
 - ii. Kilowatt Hour per Rack (estimated)
- 5. Storage
 - a. Total Bytes
 - b. Compressed Bytes
 - c. Redundant Bytes
 - d. Tape Drives
 - e. Replicated Data (does the solution consolidate it?)
 - f. SAN, Integrated or External implementation (or apportionment)
- 6. Network
 - a. Cost per Connection
 - b. Routers per Rack
- 7. People
 - a. Operational
 - i. Unclassified
 - ii. Classified
 - iii. Super Classified
 - b. Skills
 - i. Per Architecture
 - ii. Servers per Administrator
 - c. Automation investment
 - i. Current Investment
 - ii. To be developed
- 8. Outage Costs - Loss Model
 - a. Productivity Loss
 - i. User Count
 - ii. % Users effected
 - iii. User Burden Rate
 - b. Revenue Loss
 - i. \$/Minute of opportunity losses
 - ii. \$/Minute of penalty losses
 - c. Loss of Confidence or Reputation (Stock Price loss)
 - i. Publicity exposure
 - ii. Stock price vulnerability
- 9. Other costs
 - a. Migration Costs (Databases, Middleware, ISV code, etc.)
 - b. Porting Costs (Home Grown Applications)
 - c. Facilities engineering costs
 - d. Network Engineering costs
 - e. Solution Architecture costs
 - f. Reengineering for scalability costs

2.5.2 Some trade-offs

With sizing, there is a fundamental decision to be made up front: do you upgrade, replace, or add to the existing infrastructure? Each of these paths leads to a different set of trade-offs in TCO.

For example, a customer may need to double its capacity. If the customer decides to upgrade existing boxes, this is typically done by adding processors, memory and I/O to each box. In this case, overlap costs are small and network engineering costs are small to large, depending on the whether network connections are added to the existing boxes. Staffing can remain flat, power is up marginally, and floorspace only goes up if routers are added. However, there is the risk that scalability engineering costs occur, if the applications reach internal scaling limits on the upgraded systems.

Alternatively, suppose the customer decides to replace hardware. Now there are depreciation costs and deployment costs to consider. Network, staff/operations, and floorspace costs may potentially go down, but one is still faced with the same scalability engineering issues (more work per instance).

Finally, let's assume that the customer decides to add hardware. In this case, floorspace, people/operations, network and other costs go up, but scalability engineering issues are different (for example how parallel is the workload, how good is the load balancing, and how good is the data partitioning). Now, assuming we are doubling the boxes to double the capacity, this typically means that utilization on the new boxes is lower than on the existing boxes. Alternatively, we can chose to add fewer, but larger boxes, in which case the scalability engineering requirements would include both the larger instance and more instances forms of scaling. In this case there is little or no depreciation, but there may be some overlap cost.

Another fundamental decision is the choice of upgrade granularity, which is how frequently will capacity be added, or in what increments.

2.5.3 Final reminder

We've stated several times in this chapter that sizing is an important but difficult process. In our experience, even if you were to consult sizing experts on the best methodology (especially for Linux on the zSeries and S/390), chances are that you'd end up with as many methods (and results) as there were experts.

As a matter of fact, a Gartner Research Note (P-13-7373) released in June 2001 states:

There is no easy way of initially sizing how many MIPS an S/390 or zSeries will require to handle projected loads, especially with the varying system utilization of a large number of servers.

However, with the proper preparation and an understanding of the topic, a good sizing can be done. This chapter should serve as a guide to that understanding, as well as a bridge to resources that will help you along the process. It may be helpful to refer again to 2.2.2, “The bottom line” on page 25. While neither simple nor straightforward, this is an important phase of any migration project which cannot simply be ignored.



Virtual server architecture

In this chapter we discuss aspects of running many “virtual” servers on a single mainframe. We do not address the clustering of virtual servers, but instead examine the issues involved in structuring a large collection of servers for manageability and efficiency. At the end of the chapter, we offer recommendations based on our testing.

The chapter introduces concepts that are expanded upon in later chapters.

3.1 Why an architecture is required

In large, discrete server environments (and even in some smaller ones), there are a number of management and operational issues to be faced. These include:

Scalability	How to allow for increasing capacity in the environment
Management	How to control, configure and monitor the environment
Growth	How to add new servers to the environment

The penguin colony provides solutions to some of these issues, which one reason why the concept is attractive. However, if the structure of the environment is not planned in advance, the benefits of the solutions provided by the penguin colony are reduced.

By designing the penguin colony according to a set architecture (what we refer to as a "virtual server architecture"), you can build the environment so that issues of scalability and management are addressed, and so that it becomes easy to create new servers for the environment when required.

3.1.1 Components of the virtual server architecture

Here we introduce the components of the virtual server architecture, and discuss alternatives for their implementation. The components that must be considered in the architecture are:

- ▶ Disk topology - see 3.2, "Disk topology"
- ▶ Memory topology - see 3.3, "Memory topology" on page 55
- ▶ Network topology - see 3.4, "Network topology" on page 64

At the end of this chapter, we offer recommendations based on the testing we performed. However, different approaches may suit your environment better than the choices we made in our testing. The purpose of this chapter is to present a discussion of the issues, so that you can decide on an architecture which works for your installation; in the language of Internet newsgroups, YMWV¹.

3.2 Disk topology

This section discusses the ways that disk can be allocated to Linux images in a penguin colony.

¹ Your Mileage Will Vary, meaning your experiences will almost certainly differ from ours.

3.2.1 The DASD driver

Linux disk support is provided by the DASD driver code, `dasd.c`. It provides support for Count-Key-Data (CKD) and Fixed Block Address (FBA) disk devices, as well as VM minidisks.

In the Linux installation systems, the DASD driver is provided as a module. This is because the disk configuration is not known, and is determined as part of the installation. The installed system has the DASD code built in to the kernel, and parameters to the DASD driver are passed as part of the kernel parameter line.

The DASD driver uses channel I/O to perform read and write operations. For VM minidisks using the CMS RESERVE format, VM Diagnose I/O can be used to provide better performance. More detail on this can be found in 8.10.1, “VM Diagnose I/O” on page 163.

3.2.2 Linux instances with dedicated disk

The usual method of installing Linux is to dedicate disk volumes to each instance. In this scenario, every Linux instance has its own disk volumes that no other instance has physical access to, and each instance is installed in the same way.

This is not a very efficient installation method for a penguin colony, as there will be a large amount of redundant data kept in each instance—much of which will be the Linux installation itself. Software management in this scenario is intensive, because each Linux instance maintains its own copy of all applications, from user-level software right down to the kernel.

Also, because there is currently no way to partition DASDs at the Linux level, there is very little granularity in the possible DASD allocation (however, the command `fdasd` will be coming to the Linux for zSeries and S/390 distributions).

This problem can be addressed in two ways:

- ▶ At the Linux level, you can use LVM to create virtual volumes that may span multiple physical DASDs (at the expense of a longer code path to the actual disk).
- ▶ Using VM, minidisks provide a solution by dividing a physical volume into multiple virtual disks at the VM level.

Either of these options allows the equivalent of partitioning.

The benefit of this approach is its isolation. In some environments, the ability to create Linux instances that are *entirely* separate from each other is very attractive, and worth the management and definition overhead. By keeping every disk separate from all others, administrators are free to treat the instance just like they would a discrete server; they can install their own software, maintain their own configuration, and run their own services.

However, in spite of the operational isolation of this approach, you can still use VM concepts to improve management over a discrete server approach. Providing disaster recovery facilities in this scenario might be as simple as taking copies of the minidisks allocated to Linux machines, and backing them up. Restoration of a failed system would then be a matter of restoring the disk images and booting up.

This approach could also be taken with massive data loss, as Figure 3-1 illustrates.

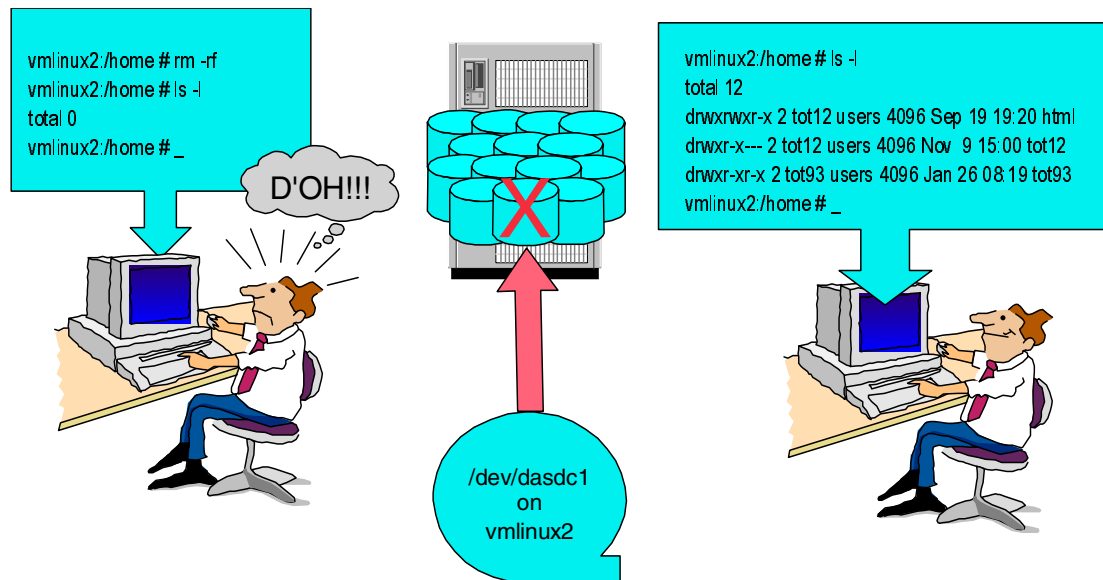


Figure 3-1 Simplified full-volume recovery scenario

In this example, a utility such as VM DDR is used to replace the entire disk image. Done simplistically, as shown, this approach is non-selective (i.e. it does not provide a means of retrieving portions of the file system), so it is a method that would suit the recovery of *entire* file systems rather than individual files in a file system.

More discussion on disaster recovery and backup scenarios appears in Chapter 7, “Backup and restore” on page 127.

3.2.3 Linux sharing data

Because many members of the penguin colony will be identical in software configuration, it would be ideal to have these instances share this common data and programs (for example, this might apply to a group of Linux instances that provide the same service or application to a particular customer). The data relating to the service must be shared between the servers for consistency.

At the same time, there will be certain data that will have to be different from one instance to another (such as network configuration data). This data can be stored on a disk device unique to each instance, but management of this system-unique data would be simplified if it could be accessible from a single location (or from all locations) in the penguin colony.

Since the members of our penguin colony have virtual network connections between them, we can use facilities that make data available across a network. There are at least two ways to do this:

- ▶ “Traditional” network file sharing

A facility such as Server Message Block (SMB) or Network File System (NFS) is used to provide discrete file system access between servers.

- ▶ Global namespace

A virtual file system which introduces a global name space to its members servers, such as AFS or GFS, provides universal access to data across the penguin colony.

Server Message Block (SMB)

Also known as the Common Internet File System (CIFS), SMB is the file sharing protocol used by operating systems like Microsoft Windows and IBM OS/2. Linux supports SMB through the Samba application suite.

In the situation where you want to share the data on your penguin colony with SMB clients (such as Windows desktops), running Samba on your Linux machines is the best approach. However, using Samba to share between Linux instances does not work well, mostly because Samba does not maintain the UNIX security information across the network. See Chapter 15, “Integrating and replacing Microsoft servers” on page 345, for hands-on details.

Note: This does not mean that Samba cannot be used between Linux machines! For simple file access it is quite capable, especially where a Linux machine is set up as a Samba server. In this case, using **smbmount** or **smbclient** from another Linux machine to perform ad hoc sharing is easy. For significant sharing of data between Linux systems, however, a “native” method that observes the Linux security model is a better choice.

Network File System (NFS)

NFS is a common method of sharing file system data among UNIX platforms. An NFS server makes parts of its file system available over the network, allowing NFS clients to mount the served file system data into its own file structure. Also, NFS observes the Linux/UNIX file permission model, so file permissions can be managed uniformly across an installation.

NFS does have limitations; in this discussion the most important consideration is an inherent lack of security based on its use of Remote Procedure Call (RPC). RPC, which uses UDP as the transport mechanism, does not provide a high level of security for two reasons:

- ▶ UDP packets can easily be “spoofed”, defeating security based on the IP address (or host name) of the source machine.
- ▶ Authentication is performed “in the clear” over the network.

Note: A complete discussion of NFS security is beyond the scope of this book. Most Linux security books offer information on NFS security.

One way to confidently use NFS in a penguin colony is to restrict NFS traffic to separate network connections, isolating it from other traffic; see Figure 3-2 on page 51. This obviously increases both management and processing overhead.

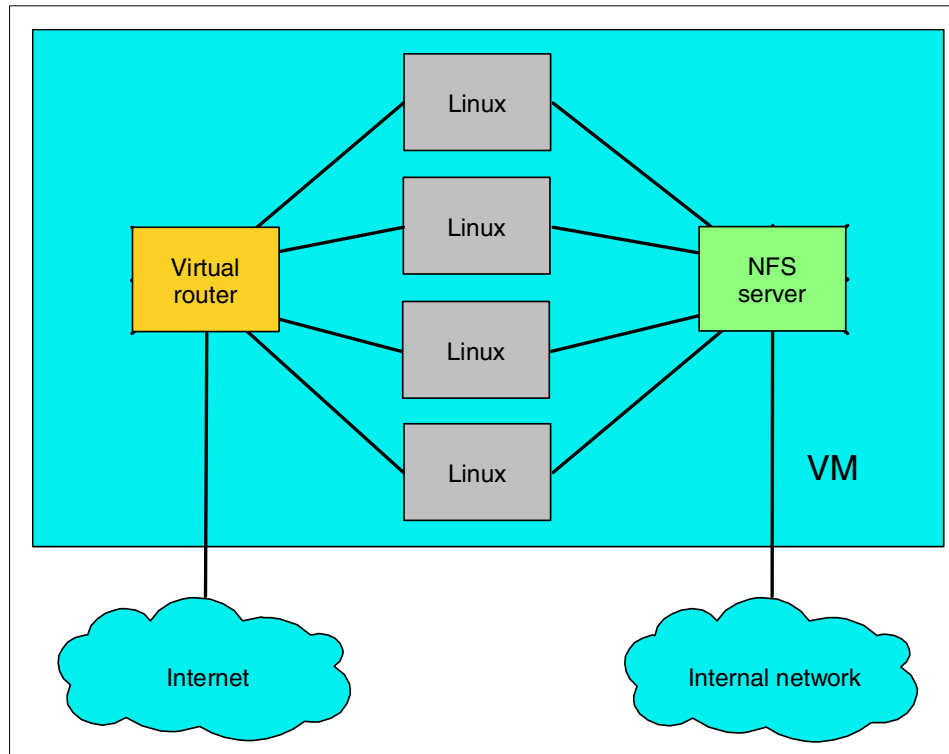


Figure 3-2 NFS isolation

The design of NFS does not automatically lend itself to providing a global namespace, but in theory a single NFS client could mount file systems from many NFS servers, which would provide a location where the relevant parts of all file systems could be accessed. However, as the number of instances increases, this arrangement would be very intensive to administer unless a utility such as **automount** were used to keep track of the mounted file systems.

Global namespace

The concept of a global namespace brings together the file systems of many separate servers into a single structure. This happens seamlessly to each member of the name space.

Global File System (GFS)

GFS is a cluster file system that provides a common name space to member servers. All servers in the GFS structure can access these physical disks. Disk devices are allocated to storage *subpools*, providing the flexibility to group disk devices according to performance attributes. Then, the file system can physically locate files in an appropriate subpool according to performance requirements.

To arbitrate access to the shared disks, either device-level locking is performed, or a global locking server is introduced. On platforms that support it, this locking server uses shared memory for data structures; otherwise, network communication over TCP/IP is used.

In the Linux-VM case, shared access to the physical disk can be provided by linking physical disks (or minidisks) to all of the systems participating in the GFS pool. It is also possible to use the network block device, mounting the physical disks to one system only and accessing the disk via the network.

Figure 3-3 illustrates the various GFS scenarios.

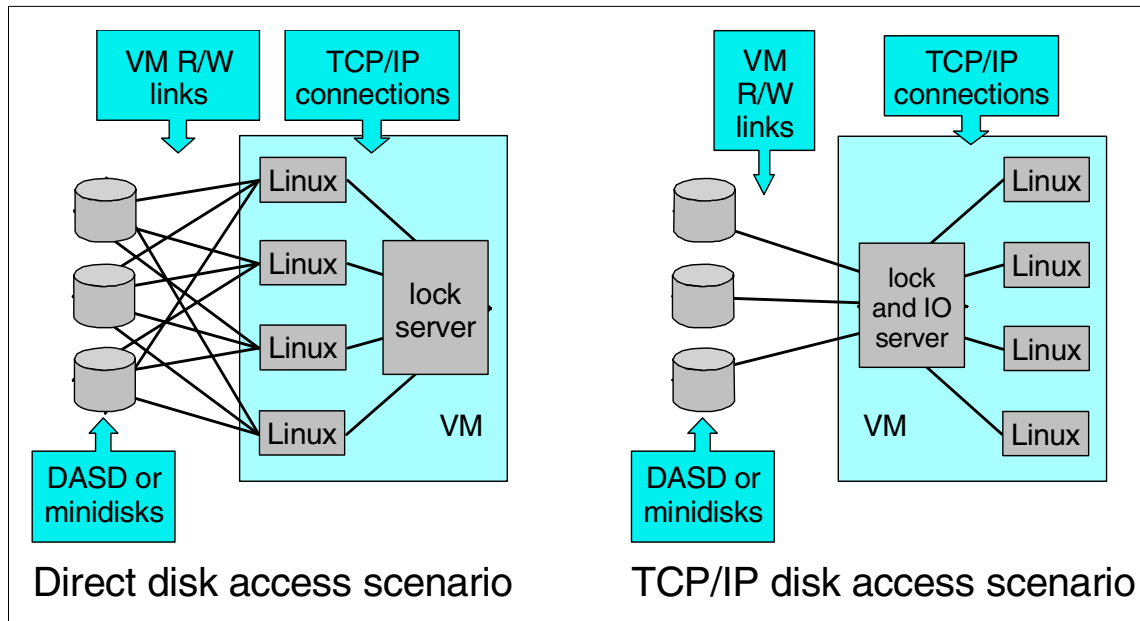


Figure 3-3 GFS scenarios

Another advantage of GFS is that it is a journaling file system, providing better file system integrity and faster recovery after a system failure.

The global name space is implemented (in part) with context-dependent symbolic links (CDSLs). CDSLs are generated by GFS to integrate certain system-specific file system components (such as /etc) into the global name space.

At present, support for GFS in Linux on zSeries is in trial stages. There are difficulties in compiling it on 2.2 kernels, and even on the 2.4 kernels it is not well proven.

Andrew File System (AFS)

AFS is another implementation of a global name space. In addition to the seamless file system, AFS provides much better security than NFS.

An Open Source implementation of AFS can be found in OpenAFS, the latest version of which is 1.0.4a. While AFS is well proven as a commercial product, OpenAFS has yet to prove itself. However, it is reasonably assured of success given the large number of organizations with experience in AFS. See 15.4, “Using AFS in an enterprise environment” on page 362 for a hands-on description.

The Coda file system

Coda (for common data access) is a descendant of AFS, designed by Carnegie-Mellon University. Coda provides similar features to AFS, with the addition of a “disconnected” mode of operation, where clients can work on files in their own local cache if the server is unavailable. When the server is contactable again, the cache is synchronized automatically. This aspect of its design indicates its primary use as an accessible file server for end-user data, rather than a shared file system for servers. More information on Coda can be found at the Coda Web site:

<http://www.coda.cs.cmu.edu>

One advantage of Coda is that the Linux kernel supports it natively. However, its development status is unclear (the bug tracking system is active, but the FAQs don’t appear to be up to date).

3.2.4 Sharing disk in memory

In 17.3.3, “Using shared segments” on page 411, we describe our work in developing a way of using the discontinuous saved segment (DCSS) capability of VM to create an in-memory virtual disk which can be shared across multiple members of the penguin colony.

This is a very promising area, which would have great benefits to large Linux virtual server environments. For example, it would be an ideal way to provide very fast read-only access to the /usr directory for a penguin colony, not only increasing performance but simplifying software management as well.

3.2.5 Minidisk caching

VM provides a feature that can provide a good performance benefit for physical disk access, which will assist in any of the scenarios described here. VM minidisk caching, as the name suggests, allocates memory in VM to cache guest minidisks to accelerate disk access.

While Linux provides its own buffer cache, it is still advantageous to provide a “second-level” disk cache, because the Linux cache takes lower priority to real user processes in Linux’s memory allocation. This means that there may not always be enough free memory available for Linux to provide an effective cache internally. The memory for the minidisk cache is preallocated, so there is always at least one level of caching available to the Linux guests.

Minidisk caching has been shown to dramatically improve the performance of Linux guests with file systems on minidisk. There is a tradeoff, however, because allocating memory to minidisk cache means that there is less memory available to be allocated to guests. Adding more memory to the processor may be justified in order to be able to take advantage of the available performance gain.

Minidisk caching can be done at a track level or at a block level. Refer to 8.10.2, “DASD MDC measurement” on page 166 for an analysis of these options.

3.2.6 Limitations of sharing disk

While it is desirable to share as much common information as possible, the sharing disk approach introduces issues that must be managed.

Caching

If a number of Linux instances have shared access to disk, they will each be holding a copy of disk blocks in buffer cache. Usually this is not a problem, but if one of these Linux instances writes to the disk, there will be a period of time where the other systems sharing that data have an inconsistency. For some applications, this may be intolerable. Therefore, the shared disk access method you choose must allow for this.

Also, because main memory is more expensive than disk, it may not be desirable to have many Linux instances holding copies of the same data in their own localized buffer caches. This would be particularly relevant when the file system is held in memory anyway, such as with VM Virtual Disk (VDISK).

The way that the Linux kernel is designed makes it very difficult to “turn off” the buffer cache. However, in the S/390 environment, in certain circumstances this would be desirable. The considerable amount of work that this would entail might be justified for certain configurations.

Locking

While discussing GFS, we discussed the need for a lock manager to arbitrate access to shared media. This requirement will vary, depending on the importance of the data and the way in which Linux instances access it. For example, a file system carrying HTML files mounted read-only by a number of Web-serving Linux instances arguably does not require a lock server (because none of the instances can alter the data).

3.3 Memory topology

The allocation of memory in a penguin colony is critical to the performance of individual instances, and of the entire installation. VM provides ways to improve the basic operation of Linux instances on S/390 and zSeries.

3.3.1 Linux ‘jiffies’

The Linux kernel employs a 100 Hz timer that “wakes up” the kernel to check for new work. The 10-millisecond units of time between timer pops are generally referred to as *jiffies*, after the name of the kernel variable where the timer count is stored. Parts of the kernel (and some other software) use this timer to check for new work when the timer pops, every 10 ms.

To VM, the constant popping of the jiffies timer means that the Linux guest is always busy. This is because the idle detection processing in VM requires a longer idle time than the jiffies timer provides. This affects the way in which VM pages Linux’s memory, because VM pages more aggressively on a guest that is idle than on a guest that is active. Also, the processing done by Linux at each of these timer pops becomes significant with large numbers of Linux instances, causing CPU constraints.

Note: The CPU impact caused by the timer results some confusion.

If the Linux instance was not processing at the time of the timer pop, it is driven to check for work, and this costs cycles. If the Linux instance was doing “real work” at the timer pop, that processing would be interrupted to allow the timer pop to be handled. Again, this costs cycles.

Therefore, in the penguin colony, reducing (or eliminating) timer processing has benefits regardless of the average load of the Linux instances.

The 100 Hz timer causes the most significant issue in management of memory in a penguin colony. The alternatives discussed in this section all provide ways to manage memory, but all are affected somehow by the 100 Hz timer. A patch to the Linux kernel has been written by IBM Boeblingen, which provides a kernel configuration option that removes the 100 Hz timer and replaces it with a different scheduling mechanism. Unfortunately, Linux developers often use the `jiffies` variable for general timing purposes (which sometimes have nothing to do with work scheduling), so simply removing `jiffies` from the kernel could have unpredictable results on other software outside the kernel.

IBM's patch is still regarded as very experimental and has not been widely tested. However, because other areas of the Linux community have shown interest in the 100 Hz timer issue (the user-mode-linux developers, for example), the patch has a lot of potential.

Information: We had access to a beta build of SuSE Enterprise Server 7.2 for S/390 and zSeries, which was released on July 3, 2001 and is based on the 2.4.5 kernel. However, even this build, created well after the original availability of the patch, does not have the patch applied. This reflects how experimental the patch is.

At the time of writing, the members of the Linux-390 mailing list were discussing the merits of making it part of the IBM “s390/s390x” patches to the 2.4 kernel.

3.3.2 Large guest memory

The simplest configuration, and the one which most closely mirrors running Linux on discrete servers, is to simply allocate to each VM guest the same or similar amount of memory as the discrete server would have.

Due to the way in which Linux utilizes RAM, however, this is not efficient in a penguin colony. The Linux kernel allocates unused RAM as buffer cache to improve disk performance. To VM, there are no unused areas of memory, which makes it difficult to determine what can be paged out; refer to Figure 3-4.

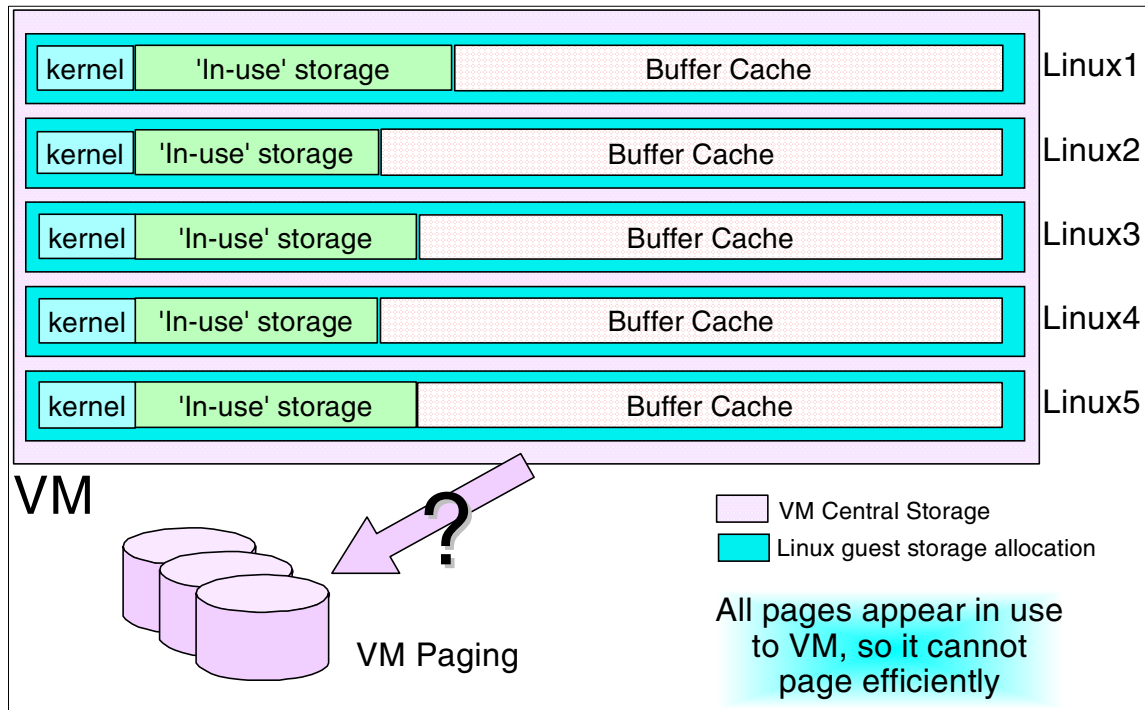


Figure 3-4 VM paging

Not shown in Figure 3-4 is the effect of Linux swap space, which further confuses the situation. Consider a memory-constrained Linux guest, with Linux swap space defined, which is operating in a memory-constrained VM system. VM will be paging Linux memory to relieve VM demands, but at the same time Linux will be swapping to relieve its own constraints. This can lead to a situation where pages of memory—marked as swapped-in by Linux but paged out by VM—are paged-in by VM simply to allow Linux to swap them out! This is generally referred to as *double-paging*, and is very undesirable.

An ideal solution allows VM and Linux to work together—or at least not fight each other.

3.3.3 Linux swap to VM virtual disk

An alternative way of allocating memory is to reduce the size of the Linux virtual machine, and allocate more swap space to it. In a discrete server environment, where disk access is orders of magnitude slower than RAM, this would be undesirable. Using VM Virtual Disk (VDISK) as Linux swap, however, we can greatly enhance the performance of Linux swap; refer to Figure 3-5.

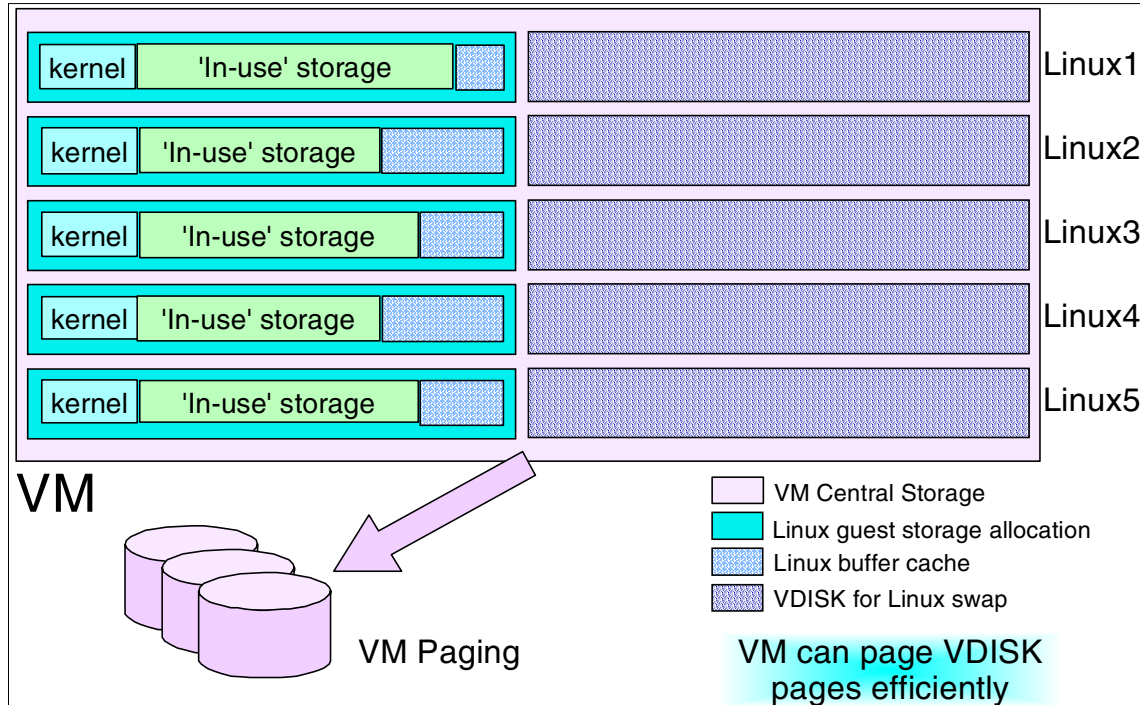


Figure 3-5 Linux swap using VDISK

In this case, even though in total we allocate a similar amount of memory to each Linux instance (once the amount of real memory and VDISK is added up), VM is able to manage the memory more efficiently.

However, there are limitations to this option:

- ▶ Applications which require a large amount of “real RAM” may not be able to obtain enough real memory to run².

² Empirical evidence suggests that WebSphere falls into this category, requiring a fairly large amount of core (in excess of 128 MB) just to start.

- ▶ Applications that are disk-intensive, and would benefit from Linux buffer cache, may experience slightly degraded performance (it is not as severe as forcing a disk I/O in every case if VM caching is in effect).
- ▶ Since Linux believes it is using a real disk, the DASD driver will still be used for swap I/O. This adds a slight performance overhead compared to memory access.

The VM paging constraint introduced by the Linux 100 Hz timer will still be an issue here, but because the size of the Linux guests' memory is reduced, the total amount of memory affected by the issue is reduced.

3.3.4 Linux swap files in memory

In order to further improve the performance of Linux's "virtual swap", we can take the somewhat radical step of having Linux swap into memory. There are two ways to achieve this, through using expanded storage or by using Linux RAMdisks.

Expanded storage

Linux can utilize expanded storage as a block device, using the XPRAM driver. This would allow an ext2 file system to be created in expanded storage, and Linux swap files to be defined there. This approach provides a fast swap facility, but there is little advantage between this approach and just allocating all of the memory to Linux as central. In addition, expanded storage is not supported on the z900 in 64-bit z/Architecture mode, so there would appear to be little future in this method.

Note: To clarify this point, z/VM supports expanded storage in 64-bit mode, but only for *its own use*. An operating system running as a guest in z/VM cannot use expanded storage. However, a guest OS could indirectly utilize z/VM's expanded storage through minidisk caching, VDISK or other z/VM services, if z/VM was configured to provide these services using expanded storage.

Linux RAMdisk

In this configuration, we increase the amount of memory allocated to the guest, but instead of using external swap devices, we allocate RAMdisks in Linux's memory and create swap files on these RAMdisks.

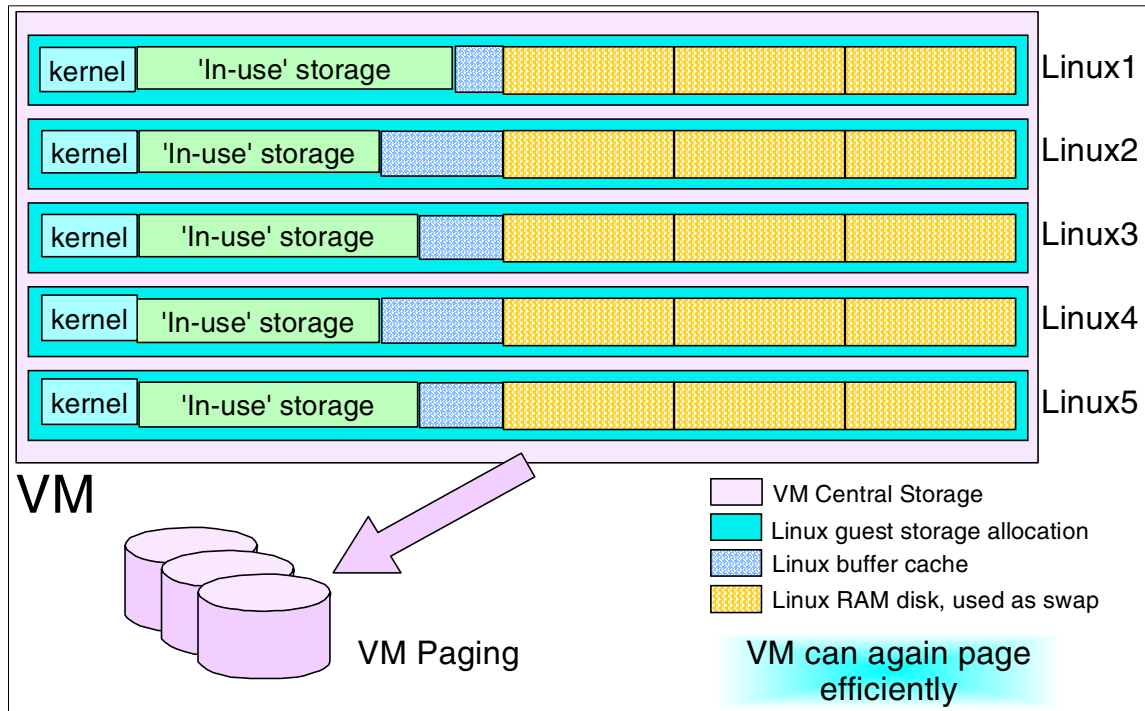


Figure 3-6 Linux swap to RAMdisk

This provides a number of advantages over the VDISK method:

- ▶ DASD driver is not used to access RAMdisk, thus reducing I/O overhead.
- ▶ Multiple RAMdisks can be set up, thus allowing good granularity in balancing memory available to Linux.
- ▶ VM can page the memory used by the RAMdisks more efficiently than if Linux was using it as buffer cache.

This configuration gives significant control over memory usage. For example, if an application requiring a large amount of “in-core” memory is started, RAMdisk swap files can be turned off and the RAMdisk deleted to return the memory to Linux (this is illustrated in Figure 3-7 on page 61).

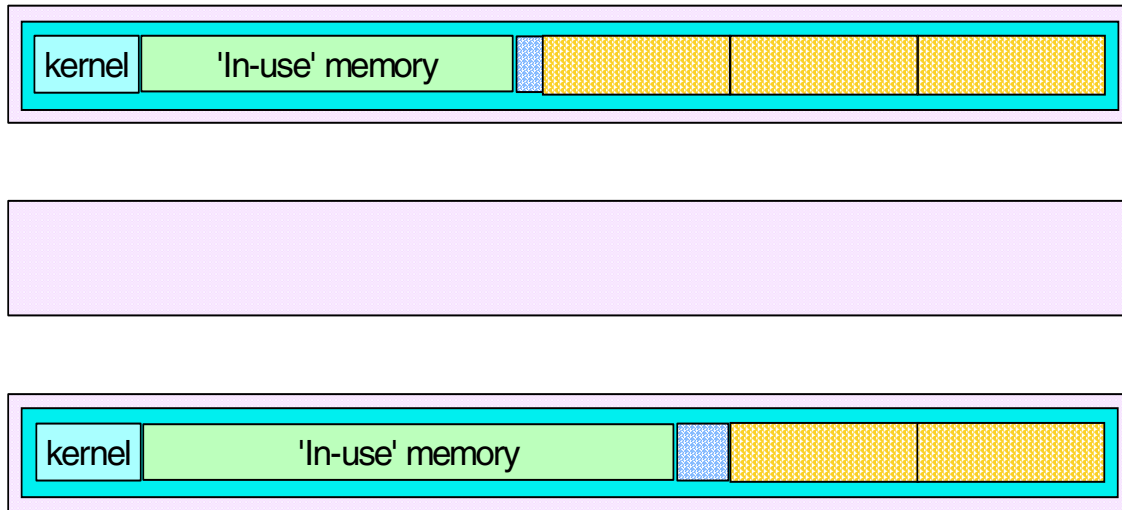


Figure 3-7 Relieving memory constraint in RAMdisk swap configuration

The reallocation of swap RAMdisk in this way would not improve the operation of a Linux image whose total memory requirement was close to or exceeded the size of the VM. If this is the case, removing the swap RAMdisk would actually cause severe problems, because *all* the available swap space would be in use. To perform the operation, additional DASD swap space would have to be brought on temporarily to manage the change.

The downside of this method is related to the `jiffies` timer: by increasing the size of the Linux guest, we've reintroduced the larger impact of the timer on VM paging. For this reason, this method would work best with a kernel modified to remove the timer.

3.3.5 “Hybrid” swap method

By using both Linux RAMdisk swap areas and VDISK swap space, it is possible to create an extremely tunable memory architecture which can provide all of the benefits of the methods discussed so far.

In this design, we allocate a number of Linux RAMdisks—but only build swap space in *one* of them. We fill the others with large files to force Linux to allocate the memory to the RAMdisk. In addition, we allocate a VDISK and make Linux swap to that. The swap space on the VDISK is marked as lower priority than the RAMdisk swap space. This is illustrated in Figure 3-8.

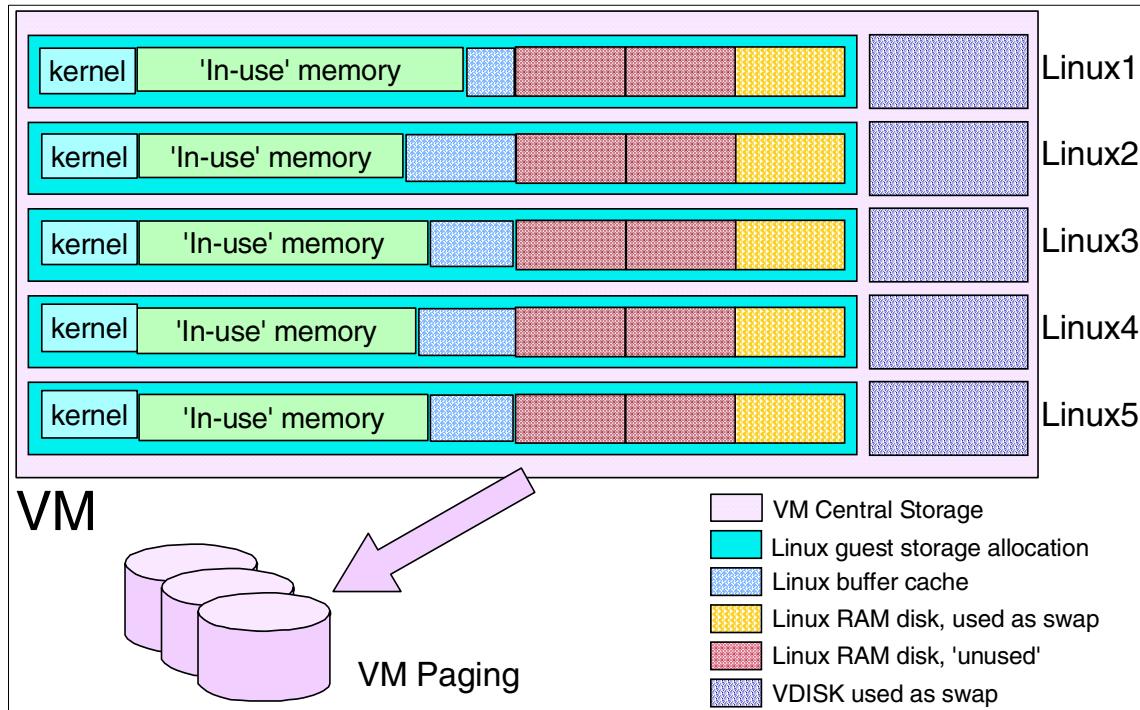


Figure 3-8 Hybrid Linux swap design

As the memory requirement of the Linux image grows, it will start swapping. Initially it will swap to the RAMdisk swap area, but if memory requirements continue to increase, it will fill the RAMdisk and start swapping to the VDISK. At this point, you delete one of the non-swap RAMdisks, releasing that memory back to core.

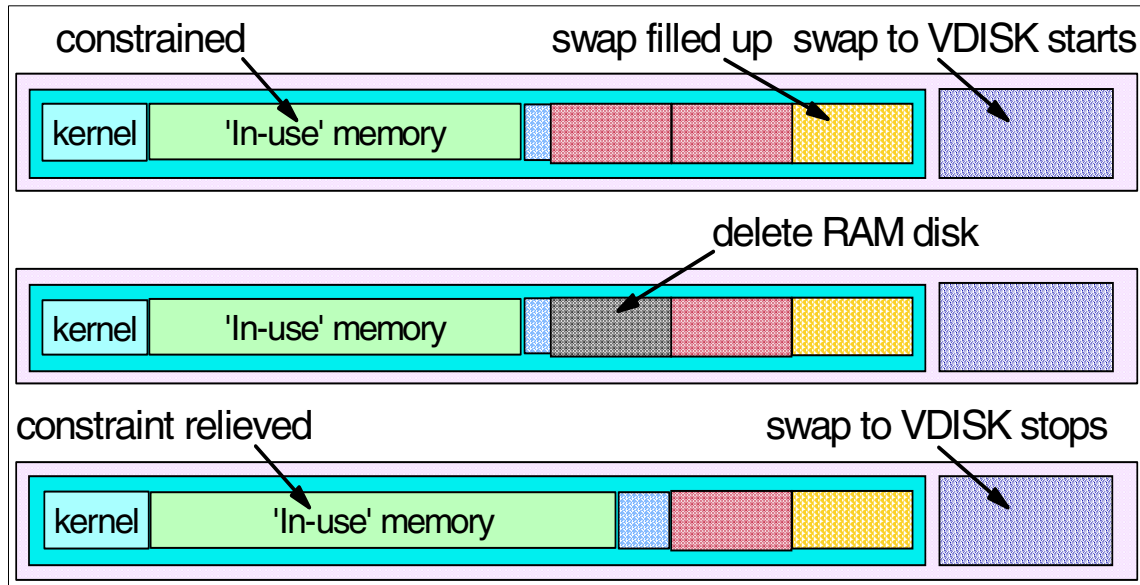


Figure 3-9 Relieving memory constraint in hybrid swap configuration

The process of deleting one of the filler swap disks can be automated. This automation could take place either in Linux (through monitoring of page I/O to the VDISK swap area), or through VM (by monitoring I/O to the VDISK itself). The memory that is used by the files held in RAMdisk will be paged out by VM, so there will be a slight overhead in paging space.

Possibly the best use of this approach is to determine the working set size of Linux virtual machines. By using this approach over time, the ideal configuration for your virtual machine size can be determined.

The process can be further enhanced to provide greater levels of granularity, by adding intelligence that recreates the filler disk and incrementally increases the size of its contents. This can give us a very good understanding of the memory requirement of the Linux instance. (However, there is a risk that a peak in memory utilization would skew the results obtained in this method.)

3.3.6 Sharing Linux memory

In a large penguin colony, many instances of common code such as the kernel will be in memory. VM can provide a shared, read-only storage area that can be used by many instances simultaneously, using a feature known as Named Shared Storage (NSS). This can greatly reduce the memory used by the penguin colony overall.

This shared area can also be used to boot a uniform build of Linux across a large number of instances. Refer to 10.7, “Linux IPL from NSS” on page 228 for more details.

3.4 Network topology

In this section we introduce methods for providing network connectivity to Linux instances in a penguin colony. We look at the connection types available to Linux on zSeries, and the use of these connection methods in connecting Linux to the network. We discuss further details of how the connection types are used in Chapter 4, “Networking a penguin colony”.

3.4.1 Network devices for Linux

There are several connection devices available under Linux. Generally, they fall into two categories, external interfaces and internal interfaces.

External interfaces

External interfaces allow Linux instances to communicate outside the boundary of the S/390 or zSeries machine the instance runs in. These are connections to network devices such as Ethernet switches and routers. The interfaces that fall into this category are the Open Systems Adapter (OSA) and Common Link Access to Workstation (CLAW) via ESCON.

Internal interfaces

As the name suggests, internal interfaces are within our S/390 or VM system, and are used mainly to connect between members of our penguin colony. Internal interfaces at this time are point-to-point connections which do not support broadcast. The two internal device types are Virtual Channel-To-Channel (VCTC), and Inter-User Communications Vehicle (IUCV).

Using internal interfaces, members of the penguin colony can communicate with each other, without needing connectivity to a physical LAN. Internal interfaces are the building blocks for providing the virtual networking used in the penguin colony.

In addition to VCTC and IUCV, there are now *HiperSockets* on zSeries machines which were added as this book was being completed. Additionally, z/VM V4R2 allows for virtual HiperSockets and the concept of a Guest LAN. For some details, see 16.2, “VM LAN support” on page 401.

Note: As discussed in “Channel-to-channel (CTC)” on page 74, CTC also operates over physical parallel or ESCON channels, and can be used to link Linux systems which exist in different S/390 processors. Unfortunately, this confuses our classification of CTC a little. The main use of CTC in a penguin colony will be the virtual kind, however, so for the purpose of this discussion, we retain the classification.

3.4.2 Network structure

Since it is not practical to connect all members of the penguin colony to the network using an external interface, it is necessary to create a “virtual” network environment within the mainframe. This will involve having certain parts of the environment set up to do no more than route network traffic. While this takes away some resources from our worker Linux instances, it is a better approach than providing dedicated network access to each individual member of the penguin colony.

Restriction: Be aware that the CTC and IUCV drivers have restrictions on the number of links that can be supported. Refer to “CTC/IUCV device restrictions” on page 102 for more details.

There are many ways that network connectivity can be configured inside the penguin colony. We will cover two of these: a *basic* design providing simple connectivity, and an *advanced* design using multiple interfaces and multipath routing. We then develop the second approach by introducing virtual IP addressing to provide enhanced connection availability.

Simple network design

Figure 3-10 on page 66 shows a basic virtual routing structure.

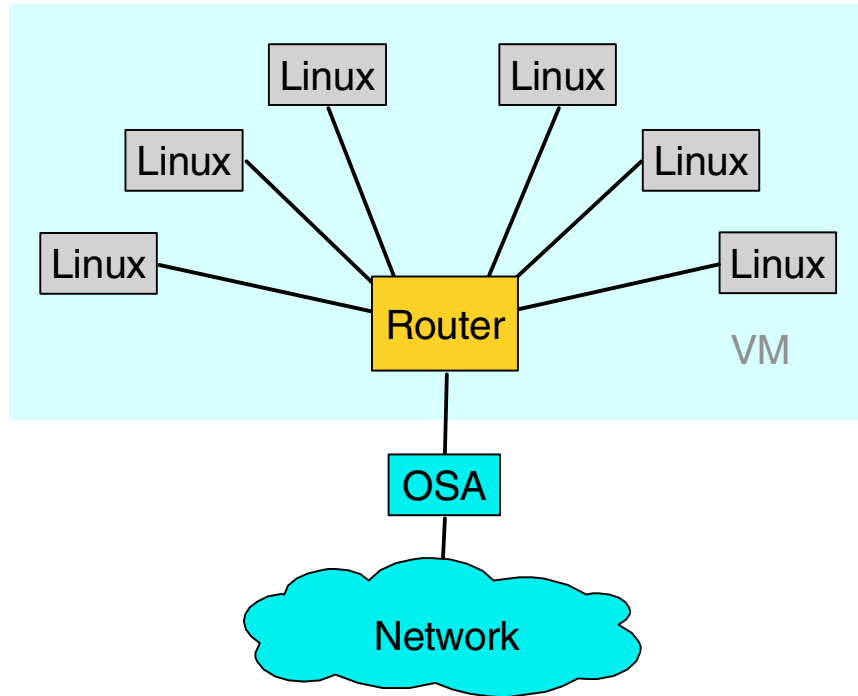


Figure 3-10 Virtual router in a penguin colony

This configuration provides for efficient connectivity, and satisfies the requirement to support many instances using a limited number of network connections.

Resilient network design

For situations requiring a high degree of availability and redundancy, the structure shown in Figure 3-11 on page 67 provides redundant virtual and physical connections.

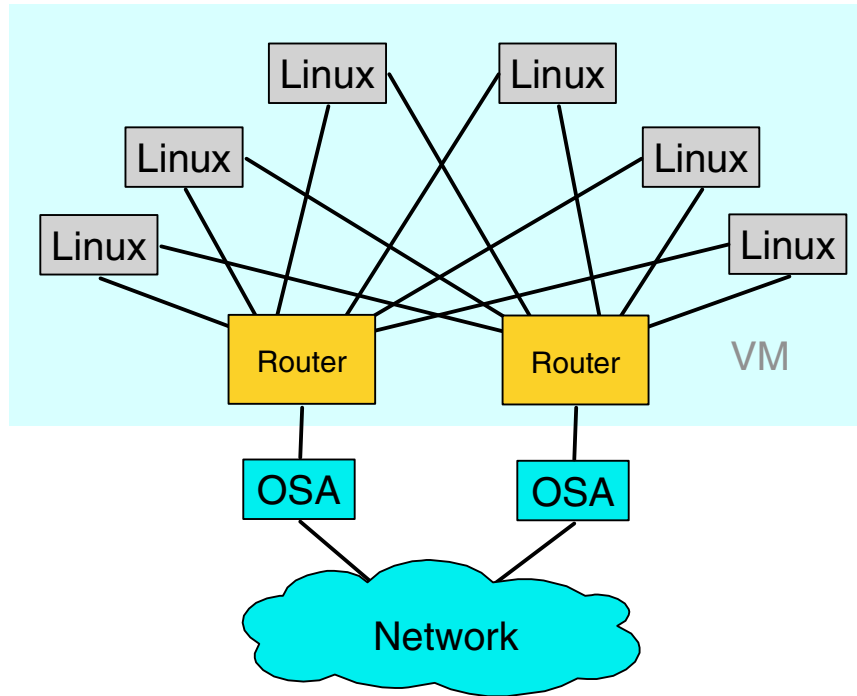


Figure 3-11 Dual router configuration

To properly exploit this design, some kind of dynamic routing is required between the virtual routers and the first line of routers in the network cloud. It may also suit requirements to extend this dynamic routing to include the Linux instances as well.

Tip: Take care in extending dynamic routing too far into the penguin colony, because running `gated` on your Linux instances will result in them having to process and generate routing updates. This will mean that they will wake up (and be awakened) at regular intervals, dramatically increasing the idle overhead of the penguin colony. This situation is discussed further in Chapter 8, “Performance analysis” on page 147.

The role of the router in these examples could be performed by any TCP/IP-capable S/390 or zSeries operating system, depending upon your connectivity method. In this discussion we will consider only Linux and VM TCP/IP.

Linux as a virtual router

The use of Linux in the virtual router role has the following benefits.

- ▶ Efficient routing engine
- ▶ Linux's advanced routing capabilities, including Netfilter
- ▶ Ability to build a thin, "routing only" kernel to maximize performance

When adding new members to the penguin colony, however, the CTC and IUCV drivers must be unloaded in order to add the new connections. This is disruptive to existing connections (and in the case of the CTC driver, may cause other servers to be restarted). In addition, the scalability of the CTC and IUCV drivers is not extensively proven.

The new HiperSockets support, which was announced as this book was being completed, will allow another networking option for zSeries machines. z/VM V4R2 provides support for both physical HiperSockets, which have a prerequisite of z900 hardware, and virtual HiperSockets. As well as running on zSeries machines, virtual HiperSockets will also run on machines that z/VM V4R2 supports: G5, G6 and Multiprise 3000 models.

VM TCP/IP as a virtual router

Using a VM TCP/IP user as the virtual router addresses the availability issue of the Linux router, because new devices can be added on the fly using OBEYFILE.

Note: The TCPIP OBEY command in VM allows the dynamic alteration of a running TCP/IP stack using configuration statements contained in a file known as an OBEYFILE. Refer to VM TCP/IP publications for more details on how this works.

Also, using VM as a "first-level" router can provide a more proven method of providing connectivity to Linux using unsupported or experimental network hardware (for example, Cisco CIP, for which the Linux driver is still considered experimental).

There are still scalability issues, however, as there are hard restrictions in VM as to the number of devices available (see "VM TCP/IP restriction" on page 104). Also, while it is generally considered that the performance of the Linux kernel would be superior to VM TCP/IP in routing IP packets, it is probably not significantly better. As it is necessary to set up at least one VM TCP/IP stack simply to support the VM system itself, the VM TCP/IP stack is a good choice for general routing use.

Combining Linux and VM TCP/IP for routing

In certain high-demand applications, it may be advantageous to combine Linux and VM TCP/IP routing in a layered approach. This may be necessary as a result of limitations in the network devices being used, or because the network interfaces available do not support Linux and you still wish to use Linux routing features such as Netfilter.

Note: Netfilter becomes available in the Linux 2.4 kernel, and provides firewall and traffic monitoring functions. It is the replacement for ipchains from the 2.2 kernel, and provides more functionality, including stateful firewalling.

Figure 3-12 illustrates a possible configuration (note that this is only an example; a real configuration may vary from this).

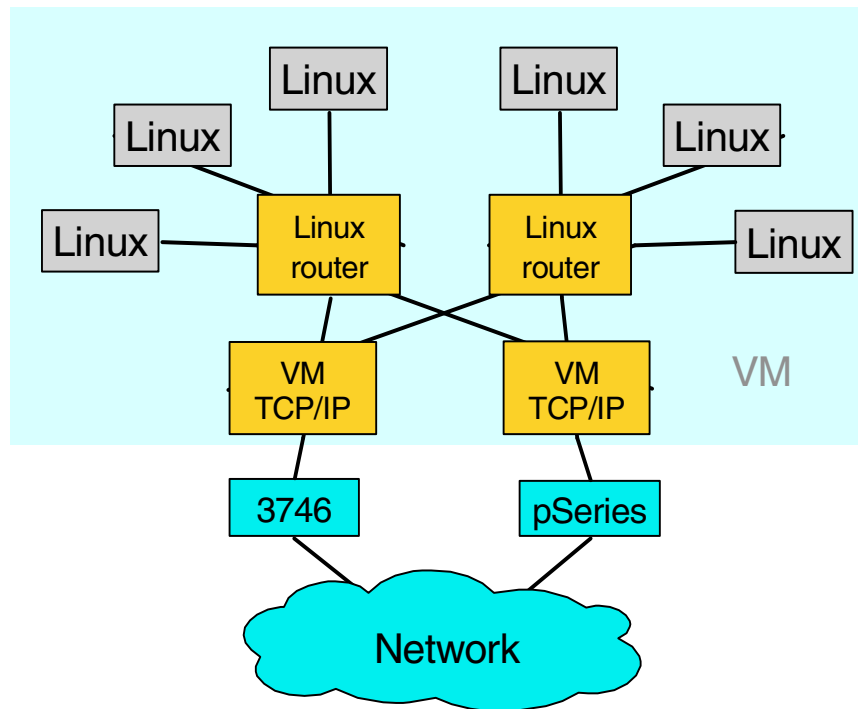


Figure 3-12 Combining VM TCP/IP and Linux virtual routing

Refer to Chapter 4, “Networking a penguin colony” on page 73, for further discussion about hierarchical routing structures.

3.4.3 General network considerations

In this section, we discuss general considerations you should keep in mind regarding the design of the network.

Talk to your friendly network staff

The success of this kind of design is dependent upon good communication and discussion with the network design team at your installation. While we've drawn "The Network" in our figures as the ubiquitous cloud, there is a lot of complexity in that cloud that must be considered by the penguin colony designers, just as there is complexity in the penguin colony that must be considered by the network team.

One of the most difficult concepts for many network staff to grasp is that we are building entire routing domains inside a single physical host. We design links, routers, and hosts that exist virtually inside VM—but in spite of their being virtual, they behave the same as physical links, routers, and hosts.

Tip: You may have noticed that the figures in this chapter have the surrounding VM system shown in very light shading. This is done for a reason, and we recommend you do the same with your own diagrams.

We suggest that, when presenting your designs to the network staff, you photocopy the diagrams with a light setting on the photocopier so that the box representing VM does not show up. In this way, the "virtual" network appears just like a network with real devices. At a later time, show the real diagrams which include VM. You can use this method to introduce people to the concept of virtual networking in a penguin colony.

Spend some time with your network team, discussing the interaction between the network and the penguin colony. Issues including physical connectivity and dynamic routing will have to be clarified as part of the design of the entire solution. Refer to Chapter 4, "Networking a penguin colony" on page 73 to find more about the issues that you will need to cover.

3.5 Workload tuning

In a penguin colony of any size, it is important to remember that all of the Linux instances are sharing the resources of the S/390 processor complex. In practice, this means that processor capacity used by unnecessary tasks should be minimized. Some tips would include the following.

- ▶ Run only the services you need to do the work you want. This ensures that the server is working at one task only. (You will see this recommendation from a security perspective, also.)
- ▶ Minimize the number of services that “poll”, or at least be aware of their impact. Examples of this include **nmbd**, the Samba NetBIOS name server daemon, and **gated**, the dynamic routing daemon.
- ▶ Where possible and convenient, use a service to back itself up rather than running additional services. In this way, the overhead of the server is reduced by not having to provide a backup service specifically. This is discussed further in “In-service backup” on page 129.



Networking a penguin colony

This chapter expands some of the issues introduced in Chapter 3, “Virtual server architecture”, related to connecting your penguin colony to your network.

4.1 Network devices

Networking on Linux for zSeries and S/390 offers many physical and virtual options.

4.1.1 Open Systems Adapter (OSA)

The OSA is a network card for S/390 and zSeries mainframes. Linux can utilize Ethernet, Fast Ethernet, Gigabit Ethernet and Token Ring OSAs. The OSA-Express is the newest version of the OSA, providing a number of improvements and new features, as follows:

- ▶ New Gigabit Ethernet interface, and improved Fast Ethernet interface
- ▶ Queued Direct I/O (QDIO) mode, which uses the Self-Timed Interconnect (STI) bus to provide memory-to-memory transfer of network traffic, dramatically increasing throughput
- ▶ IP Assist feature (available in QDIO mode), which offloads adapter processing from the host TCP/IP stack
- ▶ Dynamic OSA Address Table (also in QDIO mode only), which eliminates the need to predefine IP addresses
- ▶ Increased number of IP addresses per port available in OAT: from 16 to 512 on G5/G6 to 2048 on zSeries, and a maximum of 240 entries per OAT
- ▶ Increased number of subchannels available
- ▶ LCS support for TCP/IP when configured in non-QDIO mode

OSA provides a direct path to the “outside world” where required, but not all members of a large penguin colony can connect to it. To provide network connectivity to the entire penguin colony, virtual networking is required. These issues are discussed in 3.4.2, “Network structure” on page 65.

4.1.2 Channel-to-channel (CTC)

CTC is a point-to-point connection, using either a real S/390 channel interface or a virtual channel provided by VM. Multiple CTCs can be configured, allowing connectivity to multiple points in the environment.

The link-layer protocol used by S/390 systems for TCP/IP over CTC is very similar to Serial Line Interface Protocol (SLIP), an early TCP/IP connection method used over dial-up communications lines. Because all S/390 operating systems use the same link protocol, it is possible to connect a Linux instance not only to another Linux, but also to a VM or OS/390 TCP/IP stack.

VM Virtual CTC (VCTC) allows a device pair to be configured that links devices on two guests together. These linked devices then act exactly like a physical CTC, with VM carrying the network traffic internally between the two guests. Because no physical I/O is performed, VCTC often performs far better than a physical CTC. However, VCTC can only be used between guests in the same VM system (that is, a VCTC cannot link two guests in different VM systems, even on the same physical machine).

CTC support is provided in Linux using code contributed to the kernel tree by IBM. The CTC driver can be compiled as part of the kernel, or a module (`ctc.o`). CTC has shown high reliability in operation under Linux. However, it cannot be dynamically configured (you cannot add a new CTC device without unloading and reloading the CTC module, thus terminating any existing CTC connections). Also, the CTC driver becomes unstable if the channel is interrupted for any reason (that is, if the other end of the link is brought down).

Note: The latest versions of the IBM kernel patches include support for emulation of a raw serial connection over CTC. It can be used to provide a Linux-to-Linux connection without TCP/IP, which could be used for a console device.

4.1.3 Inter-User Communications Vehicle (IUCV)

IUCV provides a virtual communication link which is similar in function to VCTC, but does not involve channel I/O. It is available only under VM, and can be used only to connect between Linux images or VM TCP/IP stacks in a single VM system. Some prior planning is required to ensure that guests have the right directory authorization to make IUCV connections.

Like CTC support, IUCV support is IBM code that has been contributed to the kernel, and can be compiled either monolithically or as a module. From the Linux configuration perspective, it is still a point-to-point TCP/IP connection, so configuration issues are almost identical to CTC.

Because IUCV does not have to emulate channel I/O, performance is better than VCTC. While the driver has the same lack of dynamic configuration as the CTC driver, it has shown more resiliency against interruption of the communications path than CTC.

4.1.4 Other devices

In the following sections, we discuss other networking options.

Common Link Access to Workstation (CLAW)

This device type was originally used to provide TCP/IP connectivity to channel-attached RS/6000 machines, but is now also used to connect to the Cisco Channel Interface Processor (CIP) card. This would allow installations using channel- or ESCON-attached Cisco routers to link Linux systems directly, in order to decrease the routing path between the network and the Linux instance.

An experimental driver for CLAW has been written by UTS Global, but has not been extensively proven. An alternative to using this driver would be to use TCP/IP on VM to link to the hardware, then use IUCV to communicate between Linux and VM (this is illustrated in “VM TCP/IP as a virtual router” on page 68).

Other S/390 TCP/IP devices

There are a number of other connection types supported by S/390 hardware, including:

- ▶ Channel Data Link Control (CDLC), used to connect to the IBM 3746 Multiprotocol Controller
- ▶ Multi Path Channel (MPC), enhanced protocol used for OSA (non-QDIO) and host-host links
- ▶ ATM and FDDI OSAs
- ▶ HYPERchannel A220 devices
- ▶ SNALINK (TCP/IP access over SNA to 3745)

While Linux does not natively support these connections, it is possible to use a z/VM or OS/390 system to connect to them, and then use CTC or IUCV to link to the Linux system.

In some cases it may be possible to reconfigure the hardware to provide direct connectivity to Linux. For example, if you are using the MPCOSA device type for communication with an OSA in OS/390, you could create an LCS definition for use by Linux.

4.1.5 HiperSockets

HiperSockets were announced after this book was nearly completed, so they are only mentioned. They were developed to provide a highly available, high-speed network connection among mainframe operating systems, such as z/OS V1R2 or later, z/VM V4R2 or later, and Linux distributions with such support. This integrated any-to-any TCP/IP network provides a list of benefits not achievable by grouping servers around a z900 interconnected by external networking technology. See the following Web site for details:

4.2 Resilient IP addressing

In 3.4.2, “Network structure” on page 65, we show ways to create reliable physical connectivity for the members of our penguin colony. However, more work has to be done to provide resilient connectivity to applications.

Analogy: A person who wants high availability telephone service might have a second phone line installed. However, without listing both phone numbers in the telephone book, or installing an automatic diversion from one line to the other, the second line would be useless since no one would know to dial it.

In the same way, multiple IP addresses have to be tied to a single external interface in Linux.

Many methods are available to do this, including:

- ▶ DNS manipulation (round-robin, dynamic)
- ▶ Connection balancing
- ▶ Virtual IP addressing

4.2.1 DNS manipulation

Round-robin

The DNS can be altered to resolve name requests to multiple IP addresses. This is known as round-robin DNS. The DNS zone file lists multiple DNS A records for the one host name, and DNS will issue each address in sequence in response to requests for that name. The following is an example:

```
vmlinuxa IN A 9.12.6.69
          IN A 9.12.6.67
```

The A records can be specified in different orders, or multiple times, in order to weight the different adapters and bias traffic towards or away from certain adapters. For example, in the preceding example, if the first IP address was associated with a Gigabit Ethernet interface and the second was a Fast Ethernet interface, more connections could be directed to the Gigabit Ethernet interface by listing that address more often than the other, as follows:

```
vmlinuxa IN A 9.12.6.69
          IN A 9.12.6.69
          IN A 9.12.6.69
          IN A 9.12.6.69
```

This would result in the address of the Gigabit Ethernet adapter being given out four times more often than the Fast Ethernet interface.

Note that this is *not* a solution for high availability, since in this model the DNS server cannot respond to the failure of an interface, and will continue to serve the address of a failed interface in response to requests. In the example, if the interface at 9.12.6.69 failed, four out of five connection requests to vmlinuxa would also fail.

Another issue to consider is DNS caching at the client. In order to balance connections properly, these installations rely on specifying a short Time-To-Live (TTL) value in the returned DNS record. This is intended to stop clients from caching the DNS information, so that repeated requests from clients are distributed across all servers. It has been found, however, that some client applications and platforms ignore the TTL value returned from the DNS server. This might cause a particular client to continually request service from a non-preferred interface simply because its first request happened to return that address.

Dynamic DNS

Making the DNS server respond to the state of interfaces and applications is done using dynamic DNS. The DNS zone is updated by adding and deleting records on the fly according to factors including load, available applications and interface state.

Many solutions are possible using dynamic DNS, and it can also be a prerequisite to some designs based on virtual IP addressing (see “Virtual IP addressing” on page 79). One example would be to update the DNS to add an address record when a new interface is added or removed. It is also possible to integrate with network management utilities to add or remove DNS records relative to the amount of traffic being carried by the available interfaces.

Like the round-robin DNS option, dynamic DNS is subject to caching of DNS records in the client. This means that a client may have an extended outage due to caching of DNS data, even though the DNS server has been correctly updated to reflect the outage of an application server. Also, dynamic DNS is not yet standardized across DNS server implementations. For example, the Berkeley BIND-based DNS server generally distributed with Linux has implemented a method for dynamic updating which differs from the method used by the VM TCP/IP DNS server. While these differences can be designed around, it adds to the complexity of the design.

4.2.2 Connection balancing

In a connection-balancing scenario, an external (to the Linux hosts) service fields incoming connection requests and serves them to the real servers. This approach is used in clustering solutions, and is beyond what is required to provide network redundancy for a single host. However, a simplified version of this method could be used at a virtual router to share traffic across interfaces even on a single machine.

One of the great advantages of this approach is that it eliminates the problems associated with caching of connection and DNS information, since the IP address of the distribution server is used for all requests. Unfortunately, this same feature is the greatest downfall, because in a simplistic design the IP address of the distribution server becomes a single point-of-failure.

For more information on this kind of design, refer to Chapter 18 of *Linux for zSeries and S/390: Distributions, SG24-6264*.

4.2.3 Virtual IP addressing

z/OS and z/VM provide a facility called Virtual IP Addressing (VIPA), which creates an address within an IP stack that can be used to isolate applications from the failure of an IP interface. Applications bind to the virtual address instead of an interface address. If a physical interface fails, dynamic routing protocols can reroute traffic over other active interfaces, without operator intervention or outage.

Linux provides a dummy network interface that can be used to provide a similar feature. The dummy interface has an IP address, but is virtually associated with the physical interfaces of the Linux host.

Support for the dummy interface must be added to the kernel, either as a module or monolithically. You may need to add the dummy device support to your kernel before you can use it (our SuSE distribution did not have the driver available, so we had to build it).

Tip: If you have a kernel configuration file available, you can issue the following command to check the configuration status of the dummy device:

```
cat <config_file_name> | grep DUMMY
```

Remember that on the SuSE distribution, the pseudo-file `/proc/config.gz` contains the configuration of the current kernel. Copy this pseudo-file to another location and uncompress it to obtain the kernel configuration file.

A dummy interface can be added to the resilient design shown in Figure 4-1 on page 80, producing a design that allows almost transparent backup over the failure of any one of the network connections. This is shown with respect to a single Linux image in the figure.

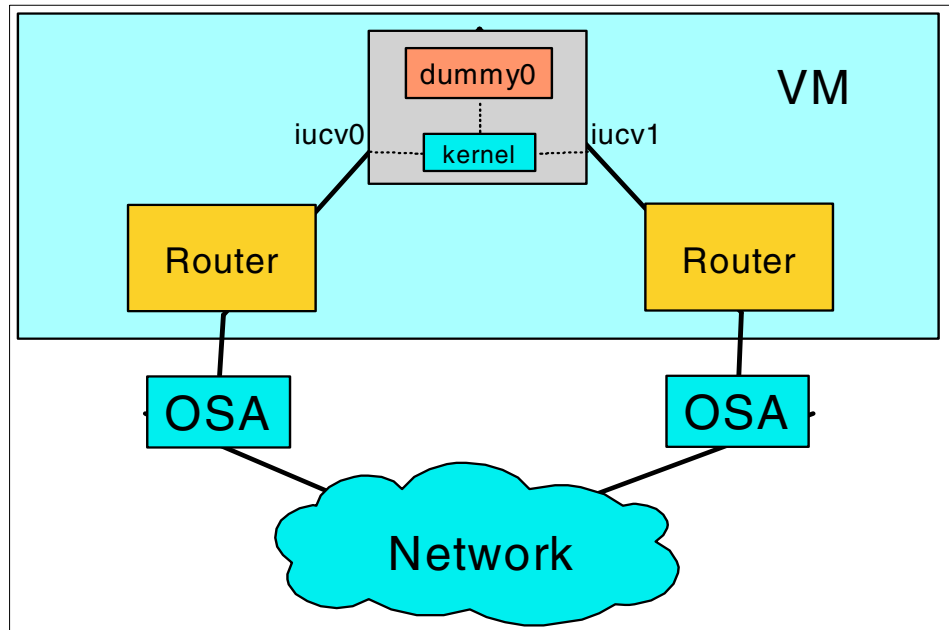


Figure 4-1 Virtual IP addressing using dummy interface

The IP address associated with the Linux image in DNS and other facilities is the IP address configured on the dummy0 interface. Using IP routing, incoming traffic is sent to this address using either network interface. If one interface fails, IP routing can forward the traffic via the other interface.

While this works for incoming traffic, any outbound traffic the Linux instance generates is associated with one or the other physical interface. In z/OS and z/VM, the SOURCEVIPA configuration support provides a means to source IP packets from the virtual address, removing the dependency on the physical device. When using SOURCEVIPA, traffic originates from a virtual interface rather than a real interface.

This support is not available under Linux at this time, so TCP/IP sessions that originate from a Linux instance using the dummy driver will fail if the interface those sessions originate on fails. Refer to 17.2, “SOURCEVIPA equivalence for Linux” on page 409 for a discussion about how this might be provided under Linux.

4.3 Packet filtering and Network Address Translation

Whenever you connect your computer to today’s Internet world, you are exposed to intruders from the outside. There are thousands of hackers just waiting to get into your computer to do damage or, perhaps, steal information. You need protection against them.

As already mentioned for general networking, our penguin colony is no different from separate servers. Therefore, our penguins need to be protected also. There are additional complications in a penguin colony, however, because you are likely to have many different networks—possibly for different customers—all sharing the same VM system. Not only does the penguin colony need to be protected from outside attack, but members of the penguin colony must be protected from each other as well.

This section introduces the IPTables utility, which controls the Netfilter features of the Linux 2.4 kernel. Netfilter provides very strong traffic filtering and translation capabilities, and is suitable for even some advanced firewalling roles inside your penguin colony.

Important: Although we provide an introduction to IPTables and Netfilter, and instructions on how to set up a capable firewall using them, we are not security experts. These chapters are not substitutes for a proper security assessment and implementation. Depending on your security requirements, you may still require dedicated firewall equipment outside your penguin colony.

4.3.1 What is packet filtering

As you can tell from the name, packet filtering is a method of filtering data coming to your computer, and is commonly used in firewall implementations.

Because everybody wants to communicate, sooner or later you need to connect your private network to the Internet; this has security considerations. With packet filtering, you can implement a firewall that will protect your computer from the outside world. You can also use a firewall on a single computer which, for example, is connected to the Internet through a dial-up line.

When you install a firewall to protect your internal network, every external computer that wants to talk to a computer on the internal network must ask the firewall for permission. If the permission is not granted, access is denied.

4.3.2 What you can do with Linux packet filtering

Using Linux packet filtering, you can do the following:

- ▶ Protect your internal network, that is connected to the Internet, from outside intruders
- ▶ Perform Network Address Translation (NAT), which allows internally-connected computers without a registered Internet address to reach Internet resources
- ▶ Filter the information going in or out of your internal network (or in or out of just one computer)
- ▶ Use your Linux server as a gateway between two different types of networks, for example, connecting Token-Ring and Ethernet worlds; this can be a cheap solution in comparison to buying an expensive router
- ▶ Share your dial-up Internet connection with others

4.3.3 Planning for packet filtering implementations

In this section, we show some of the possible designs for networking infrastructures used for packet filtering implementations. In our test environment, we always assume that the OSA card is configured to pass *all* traffic to a VM guest, not just the predefined addresses.

In the OSA card, only one IP address can be defined as the primary route (using OSA/SF), which means that this address will pass through *all* packets—not just the packet with the destination address for this IP address. When the OSA card is defined to have the primary route, we always use a VM Linux guest as the entry point for all traffic.

In the following examples, we show a few different approaches for setting up an internal network for your VM Linux guests.

Single router/firewall with one server

In the scenario shown in Figure 4-2, we have a single Linux image which acts as a router and firewall for the Linux server on an internal network. In this case we have only one server.

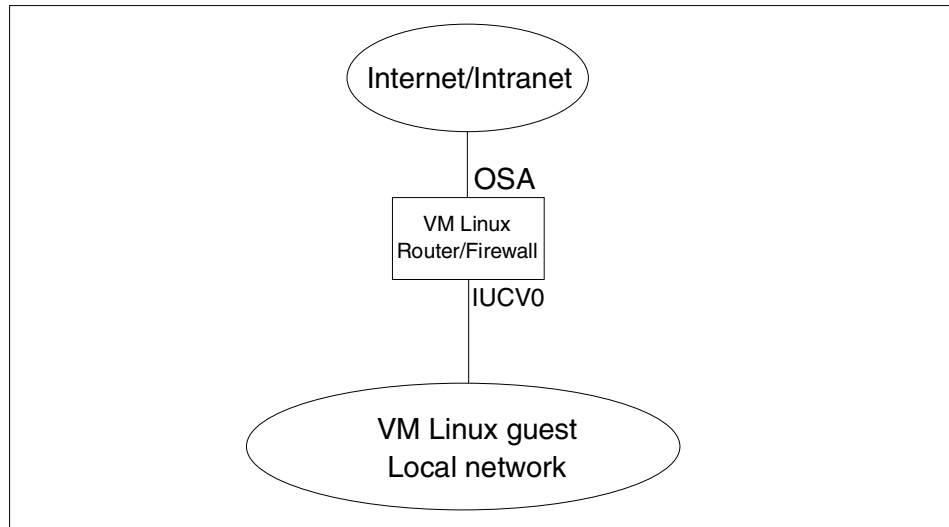


Figure 4-2 Single router/firewall implementation

Single router/firewall with DMZ on one server

In the scenario shown in Figure 4-3, we use a DeMilitarized Zone (DMZ) to host our Web and mail server. Because the DMZ is hosted on a separate interface, the other server can be protected from external traffic.

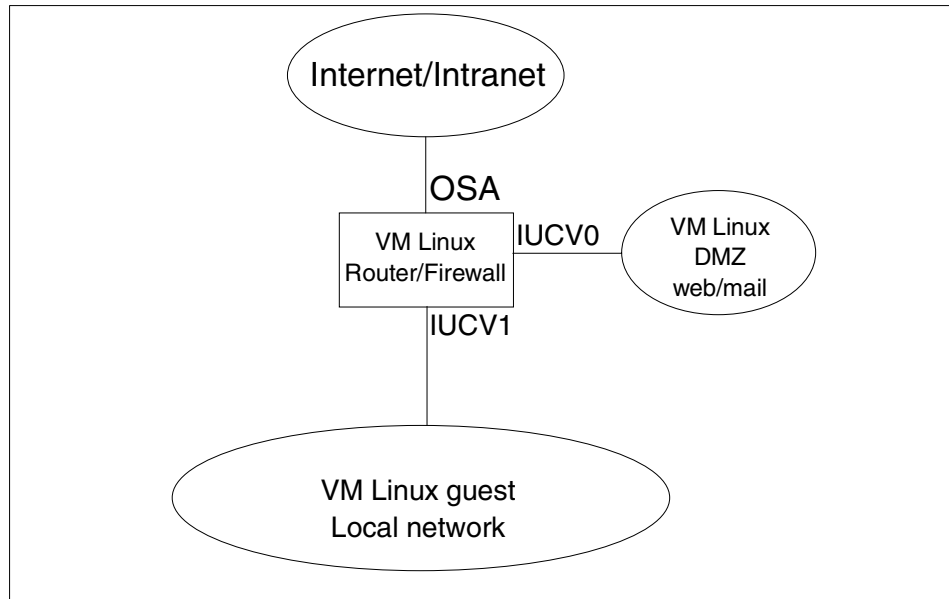


Figure 4-3 Single router/firewall with DMZ

Single router/firewall with more servers

In Figure 4-4, we use a single Linux image as a router/firewall for more servers.

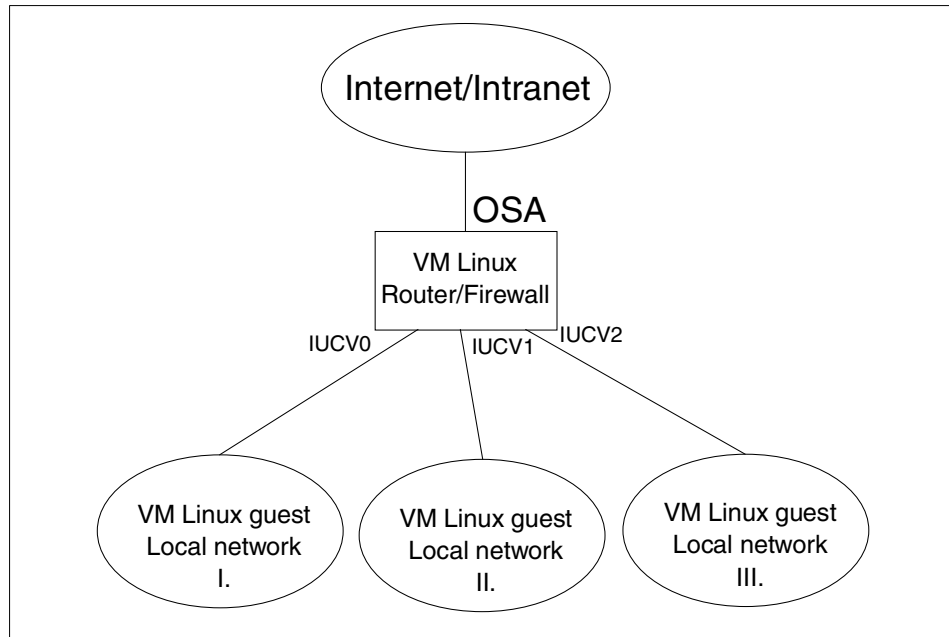


Figure 4-4 Single router/firewall with more subnets

In this scenario, each server is connected to the router/firewall via an independent interface. This means that we can control packet routing in such a way that only the correct packets are being forwarded to the servers. For example, we can block all broadcast traffic to our servers.

This kind of solution is often used when all servers belong to the same owner. In this scenario, we could also implement the DMZ for services that must be separated from the others.

Two-layer router/firewall implementation

In the scenario shown in Figure 4-5, we use two layers of routers/firewalls. The first-layer router/firewall is for the up-front Internet connection that serves the second layer of routers/firewalls. The second-layer firewalls are used to isolate each local network from the others.

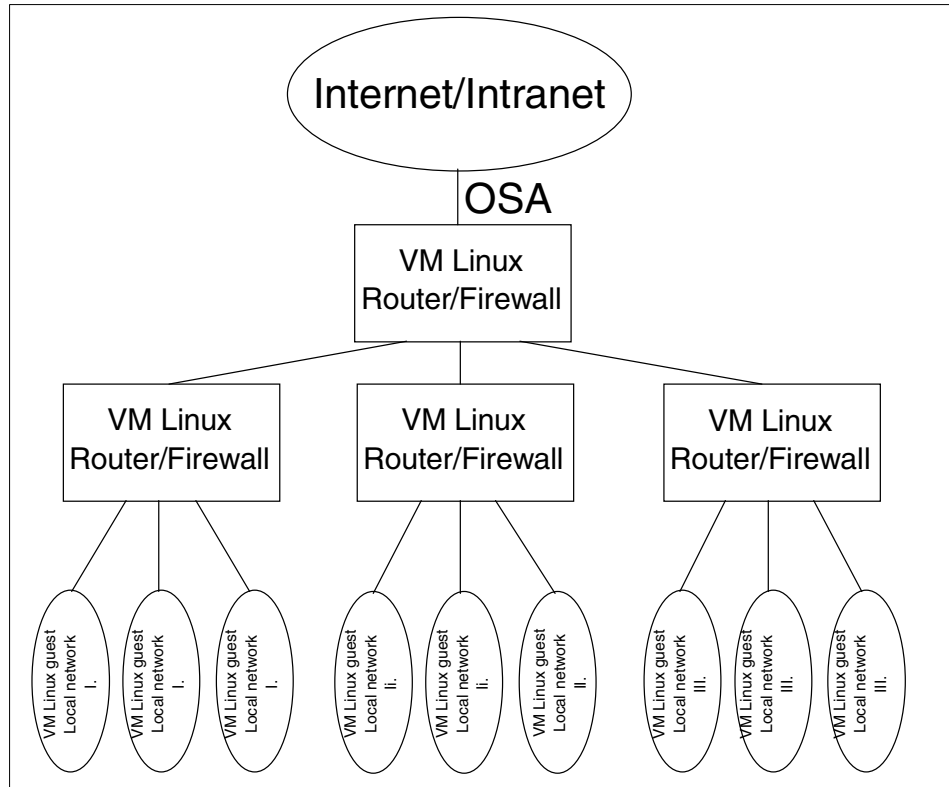


Figure 4-5 Two-layer router/firewall implementation

With such a solution, you provide an independent network for each small colony of Linux servers that's connected to the outside world over its dedicated router/firewall. Also, none of the servers in the local network interferes with the others, because it is connected to the router on its own interface.

This approach is also needed to implement a big colony of servers in the single-box solution. On the first router/firewall, we accept all traffic from the Internet. By implementing packet filtering, we can get rid of unnecessary packets (i.e., broadcasts) and pass only packets that are going to the services on our servers.

On the second layer, we separate servers among different owners for security reasons. On this layer, we also take care that the packet traveling to a particular server does not hit the interface of other servers in the local network. With this implementation, we ensure that we do not wake up any idle servers without good reason.

This approach also simplifies the management of the Linux colony inside the box, because the connections have to be predefined on VM and also in Linux. By separating each server in small local networks with its own router/firewall, you have a manageable number of servers (for example, defining connections for ten servers in the router is much simpler than defining connections for hundreds of them).

4.3.4 How packets travel through a gateway

In this section we will explain how IP Tables work. You can see the path of a packet coming into your server in Figure 4-6.

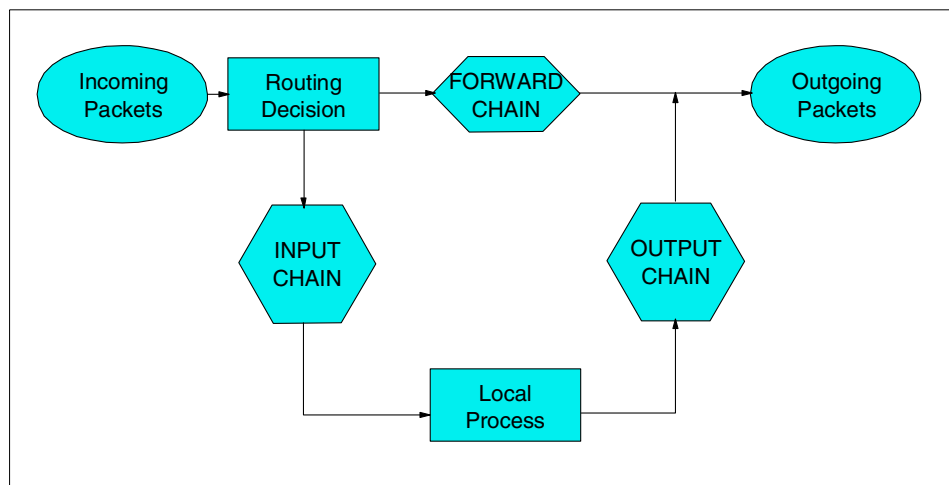


Figure 4-6 How a packet travels

The following list gives brief descriptions of each stage:

- Incoming packets** This is where the packet enters the gateway.
- Routing decision** At this point, the kernel decides whether the packet will be forwarded to another interface, or if it will be sent to a local process.
- Input chain** Before the packet gets to the local process, the kernel checks the rules in this chain against the packet.

Local process	These are applications or processes running on the server.
Output chain	Before a packet from the local process is sent to the output interface, the kernel checks the rules in this chain against the packet.
Forward chain	Packets that only travel through the router from one interface to another are checked against the rules in this chain.
Outgoing packets	This is where the packet exits the server.

As you can see from Figure 4-6 on page 87, there are three places where you can check the packets in your server:

- ▶ Input chain
- ▶ Forward chain
- ▶ Output chain

In 11.2.6, “Using IP Tables” on page 256, we explain how to use basic IPTables rules to filter traffic in your Linux instance.

4.3.5 Network Address Translation (NAT)

Network Address Translation (NAT) is used to translate the source or destination address of the IP packets traveling through the gateway. If you are using a Linux router/firewall to provide connection between the Internet and a local LAN, you can use NAT to conserve IP addresses. The router/firewall uses a real IP address, and to avoid wasting real IP addresses for the machines on the LAN, NAT is used on the router/firewall to allow the machines on the LAN to use *private IP addresses* (10.x.x.x or 192.168.x.x) to access the Internet transparently. The users see no difference between private and real IP addresses.

When packets are leaving the LAN, the router/firewall uses NAT to replace the source address of the originating computer with its source address. The destination computer replies back to the router/firewall, and the destination address is modified to send it to the computer on a private LAN.

There are three basic types of NAT:

1. SNAT (Source NAT) - in this process, the source address of the first packet is modified. SNAT is always done *after* the routing, just before a packet leaves the router/firewall.
2. DNAT (Destination NAT) - in this process, the destination address of the first packet is modified. DNAT is always done *before* routing, just after a packet enters the router/firewall.

3. Masquerading - this is a special form of SNAT.

We discuss each type in greater detail in the following sections.

Source NAT (SNAT)

With SNAT, you change the source address of the packet. This is done before packets are sent out. All other processes on the router (routing, packet filtering) will see the packet unchanged.

SNAT is used to change the address that a packet appears to come from. This is useful in applications like server clustering, where the IP traffic must appear to originate from the cluster IP address and not from the individual server that handled the request.

Destination NAT (DNAT)

Destination NAT involves altering the destination address of the packet. This can be used in Web proxy systems, where a firewall or router transparently redirects a request from a client to a cache system on your local network.

DNAT is done just after the packet comes in. All other processes (routing, packet filtering) will see the packet with the destination as you have altered it. This is important, because if the packet is routed prior to DNAT, it might be sent to the wrong interface for the altered destination address.

Masquerading

This is the specialized version of SNAT. It is usually used for dynamically assigned IP addresses (for example, dialups). With masquerading, you do not need to put in the source address; instead, the source address of the interface where the packet is going out will be used.

See 11.2.11, “Using IPTables for NAT” on page 263 for a description of how to use NAT and how to set it up using the iptables tool.

4.4 General network considerations

In “Talk to your friendly network staff” on page 70, we introduced the idea of discussing connectivity issues with the network team at your installation. This section introduces some of the issues that need to be addressed during these discussions.

4.4.1 How penguin herding is different

For newcomers to the concept of running Linux guests under VM, the idea of virtual networking connections between guests can be a bit difficult to grasp. While it is possible to give a number of your Linux instances access to the network directly, this would not be making efficient use of the hardware and the architecture. Through the use of *virtual* networking, a large amount of the network infrastructure traditionally present in a server farm can be eliminated.

Important: We do not suggest that *all* network tasks be brought into the penguin colony. Tasks such as high-throughput firewalling and management of modem banks do not benefit from being moved to S/390. However, savings can be realized by, for example, not having to provide a network port for each and every Linux instance.

In some ways, connectivity inside a penguin colony is somewhat easier than in a discrete server environment. This is because network connectivity is over non-broadcast point-to-point links, which helps make the environment more secure than using physical LANs. The routing infrastructure can become quite complex, however, because of the number of point-to-point links that must be managed.

Note: Refer to 13.7.1, “NetSaint” on page 323, for a discussion of the use of this open source tool, which we found useful in diagramming and graphically managing the network connectivity of a penguin colony.

When designing virtual routing systems for a penguin colony, remember to treat virtual routers the same way as real routers when it comes to designing your routing domains.

Virtual networking can be viewed from a network perspective, and from a processing perspective.

Virtual networking from a network perspective

In your virtual routing environment, the devices and links between virtual servers will be functionally identical to real ones. Virtual network devices behave just like real devices do, and the routing done in virtual routers is no different from routing in a physical machine.

In addition, because many basic packet-filtering tasks can be performed in your Linux routers, you can avoid the need for a lot of external firewall equipment except for high-security applications (as mentioned previously).

This is why, when designing the network connectivity, the choices to be made are no different from the choices that exist in the physical world.

Virtual networking from a processing perspective

In the world of physical routers and networks, the server people need give no consideration to the CPU consumed by routers and network equipment. The fact that a router actually contains a CPU and performs work is overlooked by everyone except the network people who support them. And even network people will disregard the fact that a modem also has processing function and requires CPU (or perhaps some analog equivalent).

In your penguin colony, however, routers and links are virtualized by VM, and share the CPU consumption of the penguin colony. This means that virtual network traffic will cost real cycles in your S/390 processor. Also, every virtual interface defined will require buffer space, and this will add up over all the machines in your penguin colony.

So this is perhaps the most important thing to remember in terms of networking your penguin colony: in return for what you save in terms of physical network equipment (routers, switches, network ports), you have to give up some S/390 processor and memory capacity to run your virtual network.

4.4.2 Dynamic routing

Almost all large TCP/IP networks today use some kind of dynamic routing to maintain connectivity. In our penguin colony, we must interact with the rest of the network and participate in the routing decisions that occur.

Dynamic routing involves routers exchanging information about the state of the network. Several protocols are used to provide dynamic routing, including the following:

- ▶ Routing Information Protocol (RIP)
This early dynamic routing protocol uses distance-vector calculations to build a view of the router network.
- ▶ Open Shortest Path First (OSPF)
This link-state protocol uses the status of the links between routers to calculate the shortest path between endpoints.
- ▶ Enhanced Interior Gateway Routing Protocol (EIRGP)
Developed by Cisco Systems, this protocol uses a proprietary algorithm which combines benefits of distance-vector and link-state algorithms.
- ▶ Border Gateway Protocol (BGP)
This routing protocol is used to provide reachability information between

separate networks, allowing connectivity between them without exchanging topology information.

Interior gateway protocols are usually used within an organization to provide high levels of connectivity between various parts of the internal network. RIP, OSPF and EIGRP are all interior gateway protocols. *Exterior gateway protocols* do not exchange topology information, and as such are used to connect networks belonging to different organizations (such as a company and their ISP). BGP is an exterior routing protocol.

Tip: For more information on dynamic routing protocols and their implementation, refer to the abundant documentation that available on this topic.

The exact method used to “Internetwork” your penguin colony with the rest of your corporate and/or customer network will vary, depending on a number of issues, including:

- ▶ The routing protocol currently in use in your network
- ▶ The addressing scheme used in the penguin colony
- ▶ How “open” the Linux instances will be
- ▶ The connectivity requirements of your customers
- ▶ Isolation from other customers

In the following sections, we present some alternatives and describe “tricks” to using these alternatives (without going into too much detail). The objective is to enable technical readers who do not have a background in networking or routing to discuss issues with the networking people at their installation.

Routing domain

A *routing domain* is a group of networks that share routing information. Generally, all routers in a given domain can reach all other networks in that domain. Within a routing domain, interior gateway protocols are used to provide this sharing of routing information.

In Figure 4-7 on page 93, three networks are shown. These may be separate organizations, or divisions within a single organization. Each of the clouds represents a routing domain.

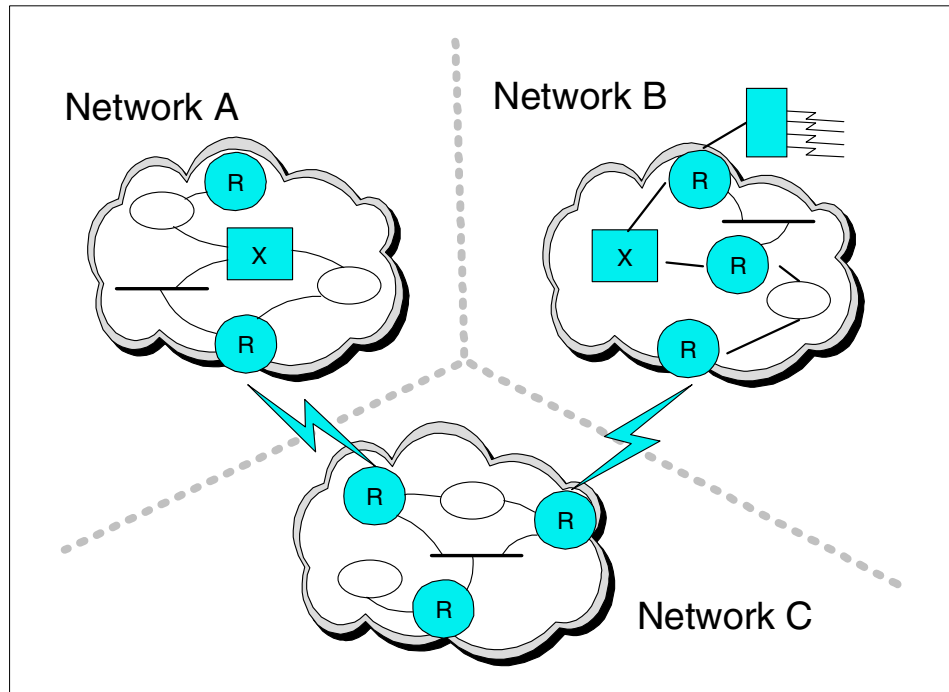


Figure 4-7 Routing domains

Routing domains can be used to control the flow of routing information in a large network, or between networks. For example, a large company with networks in cities all over the world does not need to have complete sharing of all routing information across all routers.

When devices in separate domains wish to communicate, some design work is required. Ideally, you want to allow connectivity as required, without exchanging topology information. An exterior gateway protocol is used in this role. Routers on either side inform each other about the networks that can be reached, without exchanging the detailed information provided in interior routing protocols.

In the example shown in Figure 4-7, control of the routing domains can allow communication between Network A and Network C, and between Network B and Network C, but prevent Network C from being used as a gateway between Networks A and B.

Choosing a routing protocol

This process cannot be done in isolation from the networking people (refer to “Talk to your friendly network staff” on page 70). As we have identified, for two-way communication to take place in a penguin colony, some interaction must take place between the penguin colony and the outside. If this boundary is not well planned, connectivity will be unreliable or may not even work at all. In the worst case, a poor design could cause problems in other, seemingly unrelated, parts of the network.

Example: Assume that a certain network chooses to use OSPF within its penguin colony. Because its router network also uses OSPF, it uses a different OSPF area number for the penguin colony. It sets up dynamic routing between virtual routers. The virtual routers that own the physical network connectivity for the penguin colony advertise themselves as the default routers for the area.

These outward-facing Linux systems would be the boundary between the penguin colony and the rest of the network. If they are not configured correctly, it could be possible for the Linux routers to become the default routers for the entire network! This could have catastrophic results on routing throughout the network.

One way of avoiding such problems is to use the *same routing protocol* inside the penguin colony as is used outside it. This has the benefit of uniformity across the whole network—and it does not necessarily mean that the penguin colony has to exchange complete network data with the rest of the environment (for example, OSPF allows different Autonomous Systems (AS) within a network, and this can be used to manage routing updates). This approach can be used in smaller networks, or where the consistency of using a single routing protocol is desirable.

Another method is to use a *different protocol* inside the penguin colony from the one being used outside, and have the first routers in the network manage the boundary processing.

Note: Most routers will maintain separate tables for each routing protocol. If a router running more than one routing protocol is properly configured, it can copy routing data from one routing domain into the other; this is known as “importing” routes.

For example, a router that has RIP configured on one interface and OSPF on another, can be configured to import the OSPF routes into RIP. This means that the routers in the RIP network will learn about networks in the OSPF domain through this router.

The third method is to use an *exterior gateway protocol* between the penguin colony and the rest of the network. This is an extension of the previous model, but has the effect of completely dividing the networks. It is useful when the penguin colony requires additional security, or if penguins from multiple customers are sharing infrastructure.

4.4.3 Using the OSA with Linux

Many different configurations are possible with OSA adapters, but basically one of two drivers will be used: For all OSA2 cards, the lcs.o driver is used; for OSA-Express fast Ethernet cards you have a choice: you can use the lcs.o driver in non-QDIO (or OSE) mode, or the combination of the qeth.o and qdio.o drivers in QDIO (or OSD) mode. For OSA-Express Gigabit Ethernet cards, the combination of the qeth and qdio drivers is used, because this card can only be run in QDIO mode.

Attention: In QDIO mode the OSA-Express card can communicate directly with the Central Processing Complex (CPC) using data queues in z900 memory and utilizing Direct Memory Access (DMA). This proves to be much faster than previous technologies. As this redbook was being completed it was noted that a small number of customers were concerned about performance of their OSA-Express Gigabit Ethernet cards while being driven by Linux z/VM guests. Through tuning, it was possible to significantly improve the performance of these cards. The tuning specifics will be incorporated as changes to the driver code in the near future. Before this driver code is available, the lcs.o driver with the OSA-Express fast Ethernet card in non-QDIO mode may give better performance on Linux guests under z/VM.

When installing distributions, the driver is chosen based on the type of networking you specify. We describe the files that are important for SuSE distribution in case you want to modify the networking driver implemented at install time:

- ▶ `/etc/modules.conf` - Here you associate the network drivers (e.g., `lcs` or `qeth`) with network interfaces (e.g., `eth0` or `tr0`) via the `alias` statement. When using the 2.2 kernel, you also add the parameters for the appropriate network driver.
- ▶ `/etc/chandev.conf` - Here you specify the parameters for the appropriate network driver when using the 2.4 kernel.
- ▶ `/etc/rc.config` - For the SuSE distribution only, this is where many important networking variables are set. Specifically, these are as follows:
 - `NETCONFIG` - The number of network cards: (e.g. `NETCONFIG="_0"` for one, `NETCONFIG="_0 _1"` for two)
 - `IPADDR_x` - The TCP/IP address for each interface corresponding to `_x` (e.g., `IPADDR_0="9.12.6.99"` is the IP address for the first interface.)
 - `NETDEV_x` - The device or interface name for each interface corresponding to `_x` (e.g., `NETDEV_1="eth0"` is the device for the second interface.)
 - `IFCONFIG_x` - The parameters to be passed to the `ifconfig` command (e.g., `IFCONFIG_0="9.12.6.99 broadcast 9.12.7.255 netmask 255.255.255.0 mtu 1492 up"` are the `ifconfig` parameters for the first interface.)

For details on the drivers' parameters, see *Device Drivers and Installation Commands*. For 2.2 kernels, the manual can be found on the Web at:

www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.2.shtml

For 2.4 kernels, it can be found on the Web at:

www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.4.shtml

OSA Address Table (OAT)

The OSA has a unique design that allows it to be accessed by multiple LPARs or VM guests simultaneously, each with a different IP address, but attaching to the same physical LAN. This operating mode of the OSA is known as *shared mode*.

In shared mode, the OSA Address Table (OAT) is required to provide a mapping between host device addresses and network IP addresses. This allows the OSA to send IP packets to the correct LPAR or guest.

Note: For more information about OSA-Express, refer to the product manuals or the IBM Redbook *OSA-Express Implementation Guide*, SG24-5948, on the Web at:

<http://www.ibm.com/redbooks/abstracts/sg245948.html>

If your OSA is accessed *only* by the first-level router on behalf of your penguin colony, the OAT is not used. In this case, everything will work as you expect, since the OSA will forward all traffic to the VM guest that activates it, in exactly the way that a normal Ethernet adapter operates. If you are in shared mode, however, the OAT is used to determine which VM guest (or LPAR) will receive the incoming packet.

To ease the definition work required—and to simplify the configuration, in many cases—a particular VM guest or LPAR can be defined to the OSA as the *primary router*. In this case, all IP packets arriving from the network that do not have a corresponding specific OAT entry will be sent to the LPAR or guest nominated as the primary router. A secondary router can be defined, to which packets will be sent if the primary system is not active.

Note: Primary router is often referred to as *primary default entry* in OSA documentation. *Primary router* is the term that appears in Linux device driver documentation.

Figure 4-8 shows the process used to dispatch a packet through the OSA.

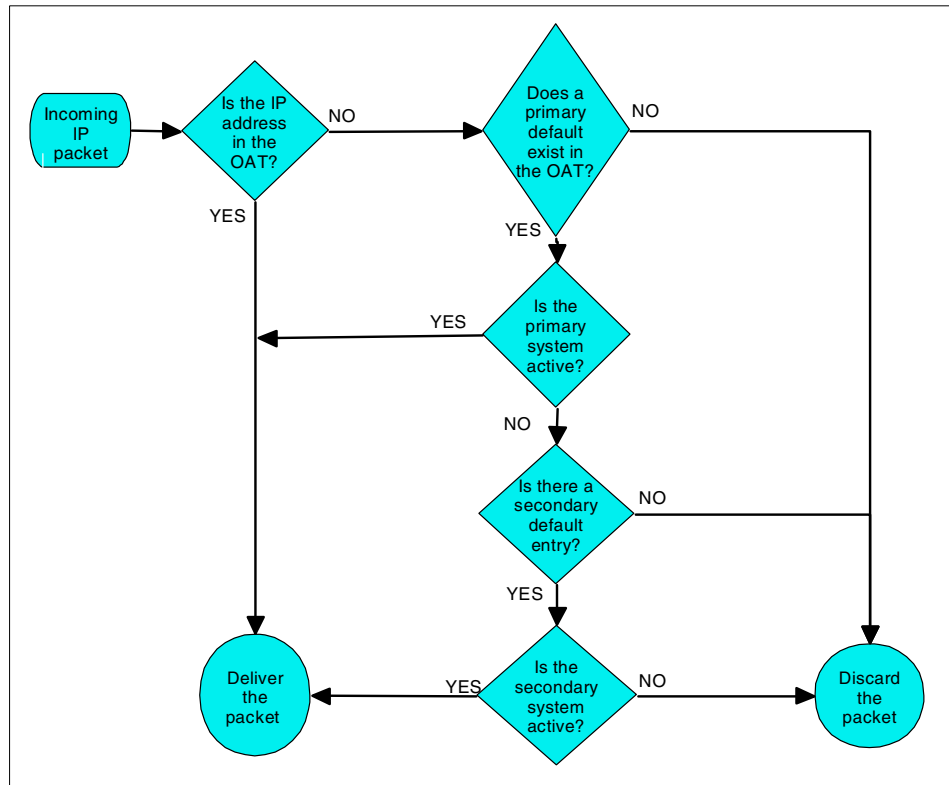


Figure 4-8 Packet path through the OSA

The IP address used in this process is the destination IP address of the packet. This is may not be the IP address of your router system. If the packet is destined for one of the Linux images in your penguin colony, then it will be the IP address of that system and not the router's address.

Note: A complete description of how TCP/IP routing works is outside the scope of this book. Almost any documentation on Linux TCP/IP will provide an introduction to routing and TCP/IP connectivity. Alternatively, you can go to:

<http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html>

The information at this URL provides a good insight into the way that TCP/IP is implemented in Linux.

To illustrate, in the following list we break down, step by step, the process involved in sending a packet from somewhere in the network to a Linux instance in a penguin colony, focusing on the steps that involve the OSA:

1. A packet leaves a client machine somewhere in the IP network. According to normal IP routing rules, the packet reaches the router immediately before the OSA in the path to the penguin.
2. The router, upon receiving a packet not intended for itself, has to route the packet onward through the network. It consults its routing table to determine the next router in the chain. It finds the correct route, which identifies your OSA-owning penguin as the next hop to the destination.
3. The routing table shows the IP address of the OSA-owning penguin. The router performs an Address Resolution Protocol (ARP) request to find the hardware address which matches the IP address in the routing table.
4. The OSA responds to the ARP request with its hardware address.
5. The router builds a new IP packet, with the same destination IP address field it has always had on its path through the rest of the network, and sends it over the Ethernet to the OSA.
6. The OSA receives the packet and consults the OAT to find out where to send it.

Note: This is the point at which the process can fail in the penguin colony environment. If the IP address of the Linux instance has not been added to the OAT, or if the OAT does not have a primary router defined, the OSA will throw the packet away, and the Linux router will not even be aware of that.

Important: Many network folk are surprised by this, since it is unusual to have an adapter make packet-forwarding decisions without input from the operating system. Due to the special capabilities of the OSA, however, this behavior is required.

The concept of the primary and secondary router in the OAT can be confusing. Primary router, in the OAT, does *not* mean that the OSA is routing packets. Think of the primary router as being the LPAR or guest that the OSA thinks is a router, and can handle packets unknown to the OSA.

Because each Linux instance in your penguin colony will have at least one IP address, the definition requirements for the OAT can be huge, or even impossible. OSA-2 could only have 16 entries in the OAT, with a maximum of 8 per LPAR or guest. This meant that only 7 Linux instances could be supported behind one router.

Even with OSA-Express, which raises these limits considerably, the definition workload would be enormous. Therefore, the current recommendation when using OSA adapters for penguin colony connectivity is to either define the router guest as the primary default entry in the OAT, or dedicate the OSA port to the router guest, using the OSA non-shared.

Using multiple OSAs

In order to support a high-availability configuration, you may choose to provide more than one OSA to your Linux router. This usually involves connecting each OSA to a different LAN, with redundant network connectivity.

You may also use multiple OSAs to provide greater bandwidth to your penguin colony, in which case you can have these OSAs connected to the same LAN. In this configuration, you need to be aware that the way the Linux kernel responds to ARP requests may cause connectivity to fail if you are not using primary routers in your OSAs.

When an ARP request is received by Linux, the kernel checks to see if the request is for *any* interface on that machine. If it is, then the adapter that received the request will respond to the ARP request with its hardware address, even if the IP address belongs to another interface.

Note: This behavior is generally an efficiency and availability advantage. The first adapter that receives the ARP request is possibly less busy than other adapters, and it makes sense to respond to the request as soon as it is received, rather than waiting until the request is received at the correct adapter. There may also be a physical cabling problem with the other adapter, in which case it would never respond.

This behavior will not work with the OSA, however, since only the OSA with the IP address defined will be able to pass traffic for that address. If you are not defining your Linux router as the primary default entry in the OAT, you must turn off this behavior in Linux with the following commands:

```
echo 1 > /proc/sys/net/ipv4/conf/all/hidden
echo 1 > /proc/sys/net/ipv4/conf/<device-name>/hidden
```

Note that the line specifying the device name must be repeated for *each* OSA interface. These commands must be executed when the system is started, so adding them to the network startup script is most efficient.

QDIO mode

OSA-Express adapters can be defined to support QDIO. This mode gives high-speed operation through direct use of the STI bus, but it also provides a facility which allows the operating system to dynamically update the adapter's OAT. This dynamic update happens using an "out-of-band" signalling path between the driver and the adapter.

Note: "Out-of-band" signalling is a telecommunications term which refers to a signalling or control channel that is separate from the path used for data flow. In the case of the OSA-Express, this path is an extra device address required when using QDIO mode instead of LCS mode.

In QDIO mode, the OAT does still exist, even though you no longer have to build it manually. Under Linux, the qeth module sends the required information to the OSA-Express to allow it to update the OAT. Setting primary router status is also controlled dynamically through the qeth driver.

OAT update from VM TCP/IP or CS/390

QDIO support in VM TCP/IP and Communications Server for OS/390 provides updates to the OAT for any configured interface. This means that not only is the IP address for the OSA adapter itself added, but any other interfaces, as well. This support is required particularly for OS/390, where the use of Virtual IP Addressing (VIPA) required the VIPA address for a stack be added to the OAT.

There are no updates for addresses that are not connected to the stack, however. This means that, in the case of an IUCV connection under VM TCP/IP, for example, the IP address of the local end of the connection is added, but the remote end is not.

OAT update from Linux

Under Linux, the qeth driver will only add the OSA's own address to the OAT. No other addresses on the Linux system are added.

However, the qeth driver supports an interface through the /proc file system which can be used to manually add an OAT entry against this instance of the qeth driver. This would allow logic to be added to the network configuration scripts for CTC and IUCV interfaces. When the interfaces are configured, the script would additionally invoke the qeth driver to add the peer IP address for the link to the OAT.

Sharing an OSA

For some time, it was considered unsafe to share an OSA adapter between OS/390 or VM systems and Linux. Now, with recent versions of the LCS driver, this precaution is not necessary. However, you may still choose to exercise caution when sharing with high-profile production systems.

The choice of whether to share or not to share will become part of your design process. For example, an existing z/OS installation using one port on an OSA-Express Fast Ethernet card may have set primary router on a z/OS LPAR to support the Application VIPA function in z/OS. In this configuration, the installation is unable to have a z/VM LPAR configured as primary router to support a penguin colony under z/VM, since an OSA port cannot have more than one primary router specified. In this situation, two possible solutions exist:

- ▶ Use the second OSA-Express Fast Ethernet port.

One port could be dedicated to Z/OS as per the current scenario, and the second port could then be dedicated to z/VM to be used for the penguin colony. Each port can have primary router specified independently.

- ▶ Use Queued Direct I/O (QDIO) mode in the OSA-Express.

In this mode, z/OS can update the internal OAT in the adapter, as required. This eliminates the need for primary router to be set on the z/OS LPAR, so that z/VM can be given the role on behalf of the penguin colony. A CTC connection could be set up between z/VM and z/OS to route any traffic intended for z/OS that goes to z/VM incorrectly.

There is also a third solution which is to route all traffic through the z/OS system, and use a CTC connection from there to the z/VM system. This solution does increase the hop count to the Linux systems. Also, many z/OS installations prefer to keep z/OS TCP/IP traffic separate, for security reasons. On the positive side, z/OS has a new feature called *HiperSockets Accelerator* which could prove useful in such a configuration.

4.5 Other issues

In this section, we address CTC/IUCV device restrictions and a VM TCP/IP restriction.

4.5.1 CTC/IUCV device restrictions

The following device restrictions exist.

Numbers of links

When building a large penguin colony and using a virtual routing scenario, it is desirable to have as many Linux instances as possible supported off one router. This increases the server-to-infrastructure ratio of your penguin colony. Unfortunately, there are currently some restrictions in place that limit the number of links that can be created.

Attention: The following points describe ways that the kernel support for CTC and IUCV can be modified to increase the number of devices supported. In order to make these changes, you *must* be familiar with kernel recompilation and C language syntax. Also, using a rebuilt kernel may have ramifications if you have a support agreement.

In the Linux 2.2 kernel, the CTC driver has a hard restriction of a maximum of 8 links. A `#define` directive in `ctcmain.c` sets this, and it could be increased to allow more CTC links to be set up.

In the 2.4 kernel, the restriction has changed: when CTC is compiled into the kernel, a restriction of 16 links applies, but when the driver is modularized there does not appear to be a restriction.

The IUCV driver appears more friendly to increasing the number of devices. While both the 2.2 and 2.4 kernel levels have `#define` directives limiting the maximum number of devices to 10, the later versions of the driver provide instructions in the code for how to increase the limit.

Attention: Modifications to these drivers should not be considered lightly; issues such as memory consumption must be taken into account. For example, the CTC driver allocates a 64 KB buffer for each connection it manages. With 256 connections, a total of 16 MB of memory would be locked in the CTC driver alone! Therefore, we reiterate: modify these drivers at your own risk.

Configurability

As mentioned in “Linux as a virtual router” on page 68, the CTC and IUCV drivers cannot currently be configured dynamically. This means that the modules must be unloaded and reloaded to support newly added devices, which requires all interfaces using that module to be shut down.

This is because the drivers currently only scan for valid devices when they are initialized. The drivers build control blocks and buffers at that time, and are not designed to support the dynamic addition of new devices.

A useful enhancement to the CTC and IUCV drivers would be the ability to perform this dynamic reconfiguration. The kernel already provides support to recognize newly configured devices, and the DASD driver is also being enhanced in this area. Reconfigurable CTC and IUCV drivers would allow a much more flexible penguin colony environment to be created, where less planning of future connectivity would be required—and adding a new penguin to the colony would be much easier and more dynamic.

4.5.2 VM TCP/IP restriction

Currently, performance of a VM TCP/IP stack degrades after approximately 100 DEVICES have been defined. The details on this are not clear, and we are not aware of the nature of the degradation in performance. We are also not aware of any active work to research this further.

We mention this as a prompt for you to find out more, so that you can factor this into the planning for your penguin colony.



Security architecture

In this chapter we discuss security topics related to running Linux on a S/390 or zSeries. Running a Linux system requires reliable security on any hardware platform. Using hardware other than S/390 or zSeries often leads to discrete servers with dedicated hardware resources.

With Linux on S/390 or zSeries, together with the partitioning of the hardware and the virtualization offered by VM available to the Linux systems, many new aspects of security arise. On one hand, you must consider if and how the physical resources of the S/390 or zSeries hardware will be isolated from the virtual Linux systems. On the other hand, the maintenance of multiple Linux systems with VM requires some additional security features in front of the penguins, but also offers clever methods of handling a colony of them.

5.1 Sharing isolation and reconfiguration of resources

As discussed in Chapter 1, “Introduction” on page 3, Linux can run on zSeries and S/390 hardware in three basic modes:

- ▶ In native mode
- ▶ In LPAR mode
- ▶ As VM guests

In the following sections, we discuss security as it pertains to running Linux in the three different modes.

5.1.1 Running Linux in native mode

The simplest security scenario is with Linux running in native mode. Here Linux is the only operating system on the hardware. Therefore, all internal resources (processors, memory, channels) are dedicated and there are no concerns about sharing these resources with other systems. It is still possible to share external resources such as DASD with other mainframe systems, but that would imply the same problems as if the Linux systems were running on a hardware platform other than S/390 or zSeries.

5.1.2 Running Linux in LPAR mode

The security scenario with Linux running in one or more LPARs gets more complicated because the PR/SM microcode is used to share the S/390 or zSeries resources.

An independent operating system can run in each LPAR (whether it's Linux, OS/390, VM or others). These LPARs and the various operating systems can share internal resources. From a logical point of view, the resources seem to be dedicated to the partition—but from a physical point of view, the hardware is virtualized and partitioned by PR/SM to be shared among up to 15 LPARs. PR/SM ensures that each LPAR uses only the resources that have been allocated to it. If total isolation of a partition is required, the system can be configured via the IOCP and IOCDS so that no resources are shared among another partition.

An ISP or ASP has to deal with various servers from several customers, who all have an essential interest in separating their data and applications from those of other customers. To make the principle of virtualization by PR/SM acceptable for these customers, some questions have to be answered:

1. Is there an official certification that the processes of the PR/SM microcode are secure, especially under the new zSeries architecture, which adds new features such as HiperSockets?
2. If CPs are shared between LPARs, is there any risk that one LPAR can influence the operations in another LPAR?
3. What is the risk if memory is reconfigured from one LPAR to another?
4. How can the dedicated and secured use of peripheral devices be assured, if shared channels and control units are used?
5. Should there be a secure administration access to Linux for zSeries and S/390 systems, especially to those that are logically placed in DMZs?

PR/SM certification

Question 1 asks for official statements and guarantees of the security provided by hardware and microcode. For S/390 servers, which means up to the 9672 G6, PR/SM has been evaluated according to the European ITSEC E4 (Information Technology Security Evaluation Criteria, evaluation level E4). This evaluation certifies that PR/SM can separate and isolate partitions as if they were running on physically separate systems. This includes topics such as the following:

- ▶ Identification and authentication
 - PR/SM will associate a unique identifier with each logical partition in the current configuration.
 - Each LPAR is uniquely identified, based on IOCDs definitions.
 - The identifier is used to mediate access control.
- ▶ Audit and accountability
 - All security relevant events are recorded in an audit log.
 - The audit log is protected from unauthorized deletions or modifications.
 - Applications in LPARs cannot read the audit log.
- ▶ Access control
 - LPAR security controls define a partition's access to IOCDs, performance data, crypto, and reconfigurable channels.
 - Access to control units and devices on shared channels can be restricted.
 - Dedicated channels, storage, and CPs are never shared.
 - PR/SM will prevent the transfer of a message between a logical partition and any resource not explicitly allocated to it.
- ▶ Object reuse
 - Storage will be cleared prior to allocation or re-allocation.
 - All information in physical processors or coprocessors will be reset before dispatching the processor to a new logical partition.

- Non-shared channel paths and attached I/O devices will be reset prior to allocation to a LPAR

For zSeries servers, a certification based on Common Criteria EAL5 is being worked on. This certification uses roughly the same criteria as the older ITSEC E4. For more details, see the URL:

<http://www.commoncriteria.org>

To answer questions 2 through 5, we have to take a closer look at the way PR/SM handles the sharing, isolation and reconfiguration of resources among LPARs.

Processors in a LPAR environment

LPARs running on S/390 or zSeries hardware can use either dedicated PUs (which only one LPAR is allowed to use), or shared PUs (which every LPAR defined for shared CPs can access).

CPs or IFLs (but not both, in the same LPAR) can be used to run Linux workloads. A CP is a general purpose processor that has the full ESA/390 and z/Architecture instruction sets, and it can run all operating systems available for S/390 or zSeries, including OS/390, z/OS, VM/ESA, z/VM, VSE, and Linux. The IFL can only be used to run native Linux, or z/VM V4 with Linux guest systems. While the number of CPs determines the model of the 9672 or 2064 (and with this the base measurement for the software licenses running on this server), an IFL is not counted in the pool of PUs that determine software pricing.

While an OS/390 or z/OS LPAR requires CPs, a Linux LPAR (which means native Linux, or z/VM V4 with Linux guest systems; VM/ESA LPARs still require CPs) can run either on CPs or on IFLs. Therefore, usually OS/390 or z/OS LPARs (which require CPs) do not share processors with native Linux or z/VM LPARs (which can use IFLs). But there still is the possibility, especially for small testing LPARs, to run Linux on a CP shared with a OS/390 or z/OS LPAR. And of course IFLs can be shared between two or more Linux LPARS, but not between Linux and z/OS LPARs.

When a processor is shared between two or more LPARs, each partition is given access to this processor for a certain amount of time. This is controlled by an algorithm based on the defined weight of the LPARs. Different partitions may use a single physical processor at different times, but the information held in the processor will be cleared before it is used by another partition.

If the time allocation for the LPAR is used up, or when the LPAR has no more work to do for the processor, or when a higher prioritized LPAR is demanding the processor, an interrupt occurs and the access to the processor is given to the other partition. When a processor is switched from use by one LPAR to another, its entire state is preserved and the new LPAR's state is restored. For the duration of its timeslice, only this LPAR has access to the processor.

Memory in a LPAR environment

Memory is not shared between LPARs, but the total amount of memory installed on a 9672 or 2064 is divided among the LPARs. Logical storage, both central and expanded, is represented by the same amount of contiguous physical storage of the same kind. This is done when the LPARs are activated, and PR/SM does not move logical partitions once they have been placed in real storage.

PR/SM also allows that physical storage can be deallocated from one logical partition and reallocated to another while these LPARs are running. If reconfigured, the storage is cleared as part of the reconfiguration process, so no data can be presented from one LPAR to another.

However, the operating system has to support this reconfiguration by being able to clear the areas of memory that are scheduled to be configured off this LPAR and given to another. VM or native Linux do not support dynamic storage reconfiguration, so there is no security impact of running Linux on S/390 at all.

Channels and external devices in a LPAR environment

The S/390 and zSeries architectures allow channels and the external devices connected to these channels to be shared between multiple LPARs.

It is also guaranteed that channels dedicated to one LPAR cannot be accessed by another LPAR, that access to control units and devices on shared channels can be restricted, and that non-shared channels will be reset before reallocation to other LPARs.

If a device that has to be accessed using a shared control unit and shared channels should be dedicated to one LPAR, this has to be defined in the I/O Configuration DataSet (IOCDs). PR/SM ensures that no LPAR can access a device that is not defined to this partition. It prevents the transfer of data between a LPAR and any resource not explicitly allocated to it.

In an environment with several Linux systems running in different LPARs, it is possible, and in most cases, sensible, to share channels and control units between the LPARs—and thus between the Linux systems. However, you need to be careful when DASD devices are shared. Unlike OS/390 or z/OS, the Linux operating system itself does not have any built-in procedures that prevent system images from interacting with each other's writing operations and destroying the

integrity of the data. Unless the applications using the DASD devices do provide special procedures to ensure data integrity, write operations to a shared DASD device should only be permitted on *one* Linux system, with the others only allowed to read the data.

Networking in a LPAR environment

Network devices are handled the same way as other external devices. They are either channel-attached control units, or located on Open Systems Adapter (OSA) cards. The ports on the OSA cards can be shared between LPARs, just as any other channel. The security in shared usage, isolation and reconfiguration of these devices is the same as with other channels and control units. Additionally, the network outside of the physical S/390 or zSeries server has to be designed and set up in a way that no unauthorized access to data or resources is possible.

For system administration tasks especially, secure network access has to be granted, and this administration network should be separated from the normal Intranet and Internet.

For the internal network communication between LPARs on the same zSeries server, two choices are provided which do not need to use any hardware outside. A TCP/IP connection between two LPARs can be set up by using a shared OSA port, or by using an in-memory communication vehicle called HiperSockets.

The first environment to support HiperSockets will be Linux for zSeries, z/VM, z/OS, and any combination of these operating systems running in LPARs on a zSeries server (there are no HiperSockets for 9672 G5/G6 and Multiprise 3000 servers). Because these connections are not physically attached to the outside network, they are as secure as the LPARs themselves.

5.1.3 Running Linux under VM

Similar security questions arise with the usage of z/VM or VM/ESA to virtualize the resources of an LPAR:

- ▶ How are VM resources and definitions protected against guest systems?
- ▶ What is the remaining risk if the resources of VM guest systems (memory, CPs) are reconfigured?
- ▶ How secure are the different kinds of communication among Linux images (for example, OSA, HiperSockets, Guest LAN, IUCV or VCTC)?
- ▶ How can the dedicated and secured use of peripheral devices be assured, if shared channels and control units are used?
- ▶ How can it be proven that VM guest systems are isolated from each other?

The System Integrity Statement for VM

The last question is asking again for official statements and guarantees for the security provided by VM. There is no official certification of VM/ESA or z/VM comparable to the ITSEC E4 certification of PR/SM. However, the z/VM guest machine separation uses the very same machine facilities that were created for, and are used by, PR/SM. So the same level of trust can be placed in z/VM and VM/ESA guest machine separation as in the PR/SM microcode.

Furthermore, IBM gives a System Integrity Statement for z/VM (in the publication *z/VM General Information*, GC24-5991), which is cited here:

System integrity is an important characteristic of z/VM. This statement extends IBM's previous statements on system integrity to the z/VM environment.

IBM has implemented specific design and coding guidelines for maintaining system integrity in the development of z/VM. Procedures have also been established to make the application of these design and coding guidelines a formal part of the design and development process.

However, because it is not possible to certify that any system has perfect integrity, IBM will accept APARs that describe exposures to the system integrity of z/VM or that describe problems encountered when a program running in a virtual machine not authorized by a mechanism under the customer's control introduces an exposure to the system integrity of z/VM, as defined in the following "z/VM System Integrity Definition" section.

IBM will continue its efforts to enhance the integrity of z/VM and to respond promptly when exposures are identified.

After this statement, in which IBM guarantees fixing every exposure of the system integrity of z/VM, there follows the "z/VM System Integrity Definition":

The z/VM control program system integrity is the inability of any program running in a virtual machine not authorized by a z/VM control program mechanism under the customer's control or a guest operating system mechanism under the customer's control to:

- Circumvent or disable the control program real or auxiliary storage protection.
- Access a resource protected by RACF. Resources protected by RACF include virtual machines, minidisks, and terminals.
- Access a control program password-protected resource.
- Obtain control in real supervisor state or with privilege class authority or directory capabilities greater than those it was assigned.
- Circumvent the system integrity of any guest operating system that itself has system integrity as the result of an operation by any z/VM control program facility.

Real storage protection refers to the isolation of one virtual machine from another. CP accomplishes this by hardware dynamic address translation, start interpretive-execution guest storage extent limitation, and the Set Address Limit facility.

Auxiliary storage protection refers to the disk extent isolation implemented for minidisks/virtual disks through channel program translation.

Password-protected resource refers to a resource protected by CP logon passwords and minidisk passwords.

Guest operating system refers to a control program that operates under the z/VM control program.

Directory capabilities refer to those directory options that control functions intended to be restricted by specific assignment, such as those that permit system integrity controls to be bypassed or those not intended to be generally granted to users.”

This definition, together with the preceding statement, is a guarantee that VM is able to provide full system integrity to the VM guest systems, and that IBM will fix any exposure to this. However, because the CP program and the guest systems are under the control of the customer, the achieved level of system integrity depends on the way the VM environment is set up and maintained. This also is made very clear by having customer responsibilities being defined as follows:

While protection of the customer’s data remains the customer’s responsibility, data security continues to be an area of vital importance to IBM. IBM’s commitment to the system integrity of the z/VM environment, as described in this statement, represents a further significant step to help customers protect their data.

Product documentation, subject to change, describes the actions that must be taken and the facilities that must be restricted to complement the system integrity support provided by z/VM. Such actions and restrictions may vary depending on the system, configuration, or environment. The customer is responsible for the selection, application, adequacy, and implementation of these actions and restrictions, and for appropriate application controls.

So, to give a short answer to the question: there is no external proof or certification (like E4 from the ITSEC for PR/SM) that VM systems are isolated from each other. But IBM warrants the integrity of the virtual machine interface and will accept integrity APARs and fix any problem exposed.

To answer the other questions, again we take a closer look at the way VM handles the sharing, isolation, reconfiguration, and management of resources between guest systems. A comprehensive summarization for this is given in Appendix B of *Linux for S/390*, SG24-4987.

Definition and management of guest systems

Simply put, z/VM transforms the principles of partitioning—which on the hardware level are provided by the PR/SM microcode—to the LPAR environment, and enriches them with virtualization. The Control Program (CP) of VM is able to virtualize hardware resources, either by sharing or partitioning real hardware resources, or by emulating their behavior. The definition of the virtual guest systems and of the resources available to them, as well as the management of this environment, is also provided by the CP.

Operating system failures that occur in virtual machines do not usually affect the z/VM operating system running on the real processor, and or the other guests. If the error is isolated to a virtual machine, only that virtual machine fails and the user can re-IPL without affecting the testing and production work running in other virtual machines.

VM resources and definitions are protected through *privilege levels*. A guest can, in general, manipulate its own environment; but without special privileges, which are under customer control, one guest cannot manipulate another's environment. Users of the virtual machines are unaware of the virtualization done by CP, just as an LPAR is unaware of the virtualization done by PR/SM.

User access to the VM system and its virtual machines can be controlled by the Resource Access Control Facility (RACF) licensed program, the strategic security facility for VM/ESA and z/VM. RACF also checks the authorization for the use of system resources like minidisks and data in spool files, and audits the use of them.

However, the RACF database cannot be shared with OS/390. In addition, in a complex VM environment, we recommend that you use the Directory Maintenance (DirMaint) product to maintain the user directory.

Processors in a VM environment

The VM Control Program defines and assigns logical processors to the guest systems, the virtual machines. These logical processors are matched to the logical processors of an LPAR (or to the physical processors, with VM running on the native hardware), which PR/SM maps to shared or dedicated physical processors.

A virtual machine can have up to 64 virtual processors defined, although a zSeries server can physically only have 16 processors (CPs or IFLs). If the operating system running in the virtual machine is capable of using multiple processors, it will dispatch its workload on its virtual processors as if it were running in a dedicated hardware environment.

The VM Control Program handles dispatching the virtual processors on the real processors available to that virtual machine. A real processor can either be dedicated to a virtual machine, or shared among virtual machines. (Keep in mind that the VM CP only handles the processors it controls, which means that if VM is running in an LPAR, a real processor to the VM Control Program can also be a shared physical processor dispatched by PR/SM.)

There is no security risk if the resources of VM guest systems are reconfigured, or if the virtual processors of different guest systems are dispatched to the same physical CPs or IFLs. The state of a processor is preserved for one guest and restored for another by the VM Control Program (just as PR/SM does for LPARs). Therefore, no information can be accidentally passed from one VM guest system to another in this way.

Memory in a VM environment

Each virtual machine has its own defined virtual memory. The physical residency of the guest system's memory pages in real storage is managed by the VM Control Program's paging mechanism. Pages that have not been referenced can be moved out of real storage, either into expanded storage or onto a paging device.

When a virtual machine touches a page that is no longer in real storage, a page fault occurs and the Control Program will bring the missing virtual page back into real storage. The memory addresses used within a virtual machine are also virtual, and they have no meaning outside the virtual machine.

The VM Control Program also allows the sharing of virtual pages by a number of virtual machines. A shared virtual page requires just one page of real storage, no matter how many virtual machines are sharing it. This can be used for sharing the Linux kernel, which is read-only to the guests, enforced by the hardware.

Virtual disks (VDISK) can also be shared by several virtual machines—and data from shared minidisk (MDISK) caches can be copied to private virtual pages. This can have a profound effect on the productive use of multiple cloned Linux guest systems in a z/VM environment. Refer to Chapter 3, “Virtual server architecture” on page 45 and Chapter 10, “Cloning Linux images” on page 209 for a more detailed discussion.

There also is no security risk if the memory of VM guest systems is reconfigured, or if portions of the virtual memory of one guest are located by the CP in physical memory regions where the data of another guest resided earlier. Memory is cleared when it changes hands, and there is no vestigial information that can “leak” from one guest to another.

Channels and external devices in a VM environment

Each operating system running in its own virtual machine communicates with virtual devices. The mapping of virtual to real devices and resources is handled transparently by the VM Control Program.

The virtual DASD devices used by virtual machines are called VM minidisks. They are implemented by partitioning a real S/390 volume into cylinder ranges that appear as separate disk volumes to the virtual machine. A minidisk can span a whole real disk volume. A real disk can also be dedicated to a virtual machine.

Minidisks (MDISK) can be shared or non-shared. If authorized, one virtual machine can link to a minidisk belonging to another virtual machine to access the data on it. Links can either be read-only or read-write.

When a minidisk is write-shared, some software is needed to manage access to the data. CP is able to cache the contents of minidisks in real or expanded storage to improve application response times, and to share this minidisk cache between several virtual machines.

It is also possible to define virtual minidisks (VDISK), which are mapped into real storage by the VM Control Program, instead of residing on real DASD volumes, and to share them. The principles of using shared minidisks, shared minidisk caches and shared virtual minidisks between multiple Linux guest systems is also very important for running multiple cloned Linux guest systems under z/VM; refer to Chapter 3, “Virtual server architecture” on page 45 and Chapter 10, “Cloning Linux images” on page 209 for a more detailed discussion.

If devices such as an OSA port are dedicated to a VM guest, the VM operating system does not influence the use of this device by the guest operating system. Also, the dedicated and secure use of peripheral devices such as VM minidisks is assured, even if shared channels and control units are used.

From a VM perspective, physically shared but logically distinct devices (e.g. minidisks) are, for all intents and purposes, separate. One guest cannot access another's data (e.g., by seeking beyond the end of their disk). Interference by one guest with another from a performance viewpoint can occur, but it is controlled by VM scheduling mechanisms. Where devices are logically shared (e.g., a shared minidisk), authorization must be given by a system administrator to establish the sharing.

Networking in a VM environment

Network communication between a VM guest system and the outside world is established over the same physical hardware devices (OSA, channel-attached control units) as previously described, but the VM Control Program manages access to them. Of course, VM only manages the devices when they are defined

as shared (rather than dedicated to only one virtual machine). The network has to be designed and set up so that no unauthorized access to data or resources is possible—and for system administration tasks, a separate network with secure access is especially recommended.

For network communication between a virtual machine in a VM LPAR and another LPAR on the same zSeries server, a shared OSA port can also be used. If both operating systems support this connection, even HiperSockets can be used (at the time writing, however, no Linux driver for HiperSockets was available).

For communicating between two virtual machines running in the same VM system (with VM running in an LPAR or on the native hardware), three additional communication vehicles are available:

- ▶ The virtual Channel-To-Channel (VCTC) device uses virtual I/O instructions. These are intercepted by the VM Control Program, which moves the data between memory buffers.
- ▶ The Inter-User Communications Vehicle (IUCV) provides a high-speed pipe for communications between virtual machines and the VM TCP/IP stack. IUCV connections can be established between pairs of virtual machines on the same VM system, or even on different VM systems.
- ▶ While VCTC and IUCV offer point-to-point connections, the VM Guest LAN, introduced with z/VM 4.2, provides multipoint virtual connections between guests, using virtual HiperSockets adapters within a z/VM system.

The VCTC and the even faster IUCV and Guest LAN connections are essential for the network design of multiple Linux virtual machines running in one VM system, with VM running both in LPAR or in basic mode.

All the different kinds of communication between guest systems and LPARs (such as shared OSA, HiperSockets, IUCV or VCTC) are completely secure in that an unauthorized third party cannot eavesdrop on them. However, keep in mind that access to these connections is only as secure as the connected operating systems using them.

5.2 Linux security

It is beyond the scope of this redbook to describe all the methods and tools available to increase security on Linux. In this section we discuss general recommendations for keeping your Linux server secure. In “Additional security documentation” on page 120, we list other sources you can refer to get a more comprehensive overview of the various security issues surrounding a Linux installation.

5.2.1 What kind of protection do you need

The way a Linux server should be protected is highly dependent upon the server's purpose. Therefore, consider what kind of access to the server is required, what exposures have to be taken into account, what kind of security attacks can be expected, and which tools to use in order to ensure the security of the system.

Based on the security basics summaries offered in *Linux for S/390, SG24-4987*, and in *Linux for zSeries and S/390: Distributions, SG24-6264*, we make the following general recommendations for protecting your Linux server:

- ▶ Disable unneeded services

Depending on the Linux distribution used, different services are activated by default. Many of the network-related services are handled by **inetd** (or **xinetd**). You can deactivate many of these services by editing the `/etc/inetd.conf` or `etc/xinetd.conf` files (but you should consider carefully if the services are really unneeded before you remove them).

- ▶ Use the tcp wrapper

To protect and log the remaining services, the tcp wrappers daemon (**tcpd**) should be used. When a communications service request is received, **inetd** invokes **tcpd** first, which then can invoke the requested service. The **tcpd** daemon performs some verification, logs the request according to the settings in the `/etc/syslog.conf` file, enforces access control and then, if everything is in order, passes the request on to the requested service.

- ▶ Use Secure Shell for remote access

Simply stopping a service like telnet is not a good solution for a Linux server that needs to be accessed remotely. To allow remote access and to prevent the password sent to telnet from being “sniffed”, replace the telnet service with the Secure Shell (SSH).

SSH connections are encrypted and protected against IP or DNS spoofing. Similarly, the secure copy (**scp**) command can be used instead of FTP, and secure login (**slogin**) can be used instead of rlogin.

For additional security, remote login for root can be forbidden. Then root access will be limited to the Linux console which is a VM session. In this scenario, both the VM and the Linux passwords would have to be cracked.

- ▶ Use shadow password utilities

The `/etc/passwd` file often contains encrypted passwords that can be read by all users. This creates the possibility that weak passwords can be cracked via dictionary attacks. To avoid this vulnerability, we recommend that you use the shadow password utility, where passwords are stored in the `/etc/shadow` file (which does not have read access).

Additionally, this file contains information about expiration and renewal, so maintenance of passwords is eased. Use of shadow passwords is the default in current versions of all major distributions.

- ▶ Use the Pluggable Authentication Module (PAM)

The PAM provides a library of authentication modules, all located in the `/etc/security` directory. These modules offer standard procedures for authentication purposes and can be used by various services, which configuration files are listed in the directory `/etc/pam.d`.

Without these modules, every application would have to contain its own authentication code, if anything more than the standard user authentication by password is required. And if the authentication requirements of an application change, the whole application would have to be recompiled. By using PAM, only the configuration file in the `/etc/pam.d` directory has to be changed.

- ▶ Monitor security news and alerts

In order to keep abreast of news concerning vulnerabilities or bugs in software running on the Linux server, the system administrator should check the Web sites related to these products often, and also check general Linux security-related URLs.

In addition, the sysadmin should frequently monitor the log files of the applications (usually located in the directory `/var/log/`) for any problems.

- ▶ Use LDAP servers for authentication

For directory services over TCP/IP, the Lightweight Directory Access Protocol (LDAP) should be used. One or more LDAP servers contain the data making up the LDAP directory tree. Information about objects (such as people, organizational units, printers, documents, etc.) are stored as entries in a hierarchical structure. LDAP provides a mechanism for clients to authenticate to a directory server, in order to get access to the information provided there.

The implementation usually provided with Linux is OpenLDAP; refer to the following URL for more information:

<http://www.openldap.org>

Setup and usage of OpenLDAP is described in Linux for zSeries and S/390: Distributions, SG24-6264. OpenLDAP provides a directory service called **s1apd**, which handles access control to, and management of, the databases containing the directory trees. It also contains the **s1urpd** daemon, which helps **s1apd** provide replicated service.

- ▶ Use firewall tools to secure your network

To control the traffic of TCP and UDP packets by using the IP firewall rules on the Linux kernel, IPTables should be used; see Chapter 11, “Network infrastructure design” on page 235 for details. Thorough evaluation and planning of the network infrastructure is required regarding the setup of Virtual Private Networks (VPN), including the arrangement and use of proxies, reverse proxies, and firewalls.

- ▶ Protect against viruses and trojan horses

Because of its software architecture (in particular, memory management and file/user permission design), Linux is not susceptible to the traditional viruses that plague more elementary operating systems like Windows.

But this does not mean that Linux is entirely safe from mischief or external threats, especially “trojan horse” programs. The distinction between a virus and a trojan horse is critical, and illustrates why Linux is relatively immune to viruses but not to trojan horse programs—so let’s explain these terms here.

What is a virus

A *virus* is a program that actively operates, independently of a user, to attack various system resources. On most Windows systems, a user is also the administrator. Therefore, all system resources (disk, memory, applications, files, logs, devices, etc) are accessible by anyone or anything, including the virus program.

It is impossible for a Linux operating system to suffer system-level damage from a virus, because it cannot get access to low-level system functions. However, just because Linux is relatively safe from viruses doesn't preclude it from spreading mail-based viruses when it is being used as a mail server.

For this reason, there are antivirus programs that can be used for detection and disinfection of viruses and malicious code passing through Linux firewalls; for example, refer to the following URL:

<http://www.f-secure.com>

What is a trojan horse

By contrast, a *trojan horse* is a program that cannot operate unless it is invoked (unwittingly) by a user. Generally speaking, Linux systems don't execute trojan horses on their own; they must be executed explicitly by the user, and are especially dangerous if the user is the root or superuser of a Linux system.

Therefore, to help prevent the introduction and execution of a trojan horse, a system administrator should, at a minimum, avoid logging in as root or otherwise assuming superuser capability unless it is absolutely necessary for some sysadmin task, and should furthermore ensure proper access permissions on files (which is particularly important for system utilities, devices and log files).

A very useful hardening tool is tripwire, which is able to detect and report file and directory modifications. This can help to detect trojan horses and modified software left by hackers, for example for sniffing passwords.

- ▶ Use tools for testing network security

You can use tools such as the **scanlog** daemon to test network security. This daemon is able to recognize if someone is requesting more than a specific number of ports per second, which can indicate that someone is scanning the Linux server for insecure ports.

5.2.2 Additional security documentation

The following documentation provides detailed information about Linux-related security topics. The paper “Addressing Security Issues in Linux”, by Mark Chapman, presents a broad overview of the various security issues regarding Linux installations, and what you can do to keep these subjects under control. It is available on the Web at:

<http://oss.software.ibm.com/developer/opensource/linux/papers.php>

A very useful paper, it discusses the most common tools and utilities for increasing the level of security, and refers you to various URLs for further information.

Another excellent paper entitled “Linux Security State of the Union”, by Robb Romans and Emily Ratliff, is also available at the same URL. The authors discuss the main prejudices regarding open source software in general and Linux in particular, which often inhibit the use of these products in production environments. The contention is that an open source operating system need not be insecure. On the contrary, the availability of the source code availability implies the advantage of a very stable product (because everyone is able to run and test it), whose bugs are fixed with extraordinary speed.

The paper lists and describes projects that are underway to improve overall acceptance of Linux as a secure operating system, ready for productive usage in an enterprise environment.

5.3 Cryptography on z/Series

S/390 and zSeries servers offer specialized processors for cryptographic operations. The IBM 2064 zSeries 900 supports up to eight PCI-CC cards. Each PCI-CC card contains two engines and is assigned two CHPID numbers. Using these cards for cryptographic operations, the CPs or IFLs are released from these processor-absorbing instructions.

In an ISP or ASP environment, cryptographic procedures are generally used for secure TCP/IP connections between the server and the user somewhere in the Internet. Applications for firewalls, Web serving and mail serving have the requirement to protect data, as shown in Figure 5-1.

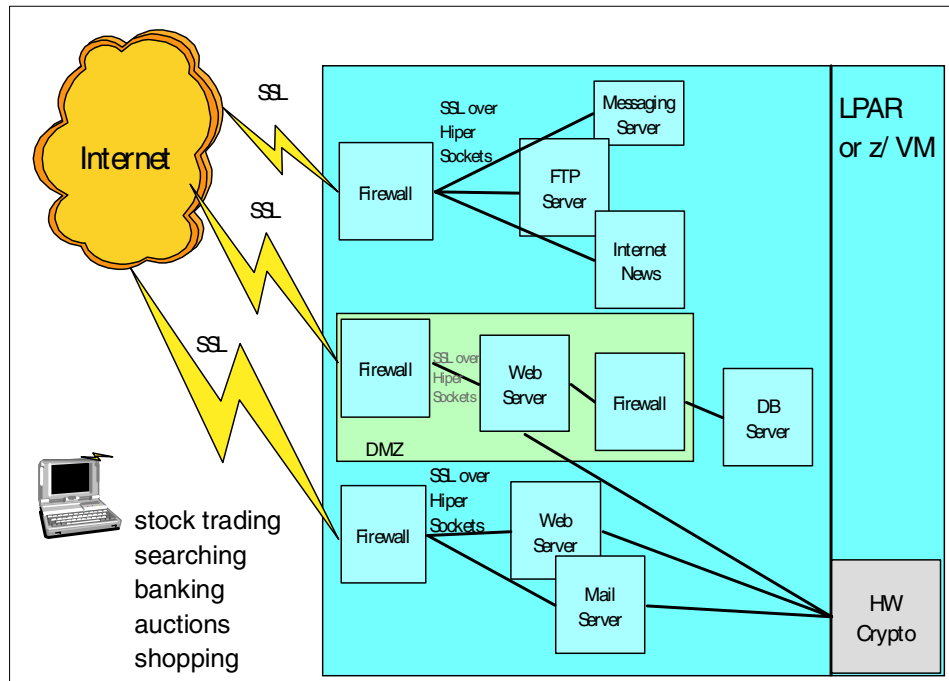


Figure 5-1 Usage of zSeries 900 PCI-CC card

The operating system has to be able to recognize cryptographic instructions, pass them to the PCI-CC card to be executed, and return the result to the application.

Linux on S/390 and zSeries will provide this functionality for hardware encryption in the first step for those asymmetric algorithms used by SSL, which will result in remarkable performance enhancements for SSL transactions. The future directions will also include hardware cryptography for symmetric algorithms (such as DES, 3DES, RC4, etc.), support for cryptographic file system, and digital signatures.

From a security point of view, there are no system integrity exposures if the PCI-CC card is shared and used both with OS/390 and Linux workload. Aside from potential performance concerns if there are not enough crypto features, there are no security considerations. Each operation is discrete and independent

of those that precede or follow it. VM manages the queue of requests to ensure that a guest can see only its own request, as with a shared processor or a shared channel. Of course, if necessary, a PCI-CC processor can also be dedicated to a Linux guest system in a virtual machine.



Migration planning

In this chapter we discuss high-level considerations for planning a migration from distributed servers to Linux on S/390 or zSeries.

6.1 Where to start

The reality of any large organization's IT infrastructure is that it defies easy analysis, so how do you select which servers are the best candidates for migration? By solving the biggest problem first; focus on the area where there is the most "pain", and fix it.

6.2 What to look for

Consider the following when selecting which servers to migrate.

6.2.1 Small scope

The most successful very early migration customers we've been involved with were faced with these small-scope requirements:

- ▶ Needing to upgrade multiple servers running on older hardware
- ▶ Needing to upgrade multiple servers running on older operating systems
- ▶ Needing to increase disk capacity
- ▶ Being unable to justify underutilized single servers

Normal attrition presents many low-risk opportunities to consolidate single-use servers. This builds a track record of success and customer satisfaction. The financial comparison is made using real data.

This type of migration will not justify the purchase of a new S/390 or zSeries; instead, it assumes there is extra capacity on an already installed machine. However, the incremental cost of running the migrated servers can be very accurately measured in terms of the sunk costs in the data center. When compared to the costs of many upgrades to many small servers, this incremental approach is very attractive financially, especially in terms of technical support costs. File and print servers are the typical examples of this migration, and have been very successful.

Medium scope

Other customers have taken a more comprehensive approach. Rather than wait for servers to reach end-of-life before migrating them, a more global and proactive plan is used. Some indicators for using this approach are:

- ▶ Consolidating a new acquisition
- ▶ Building, expanding, or moving to a new data center
- ▶ Implementing a plan for increased capacity
- ▶ Redesigning or upgrading a network

This type of migration depends on having strong skills already in place in the S/390 datacenter. It assumes there is plenty of capacity on the machine—or a budget to buy more capacity. It further assumes end-to-end cooperation from all members of the technical support staff. The people who will support applications on Linux need to see this as an opportunity to enhance their skills and marketability. That applies equally to the NT administrator, the MVS system programmer, and the VM system programmer.

Large scope

Another type of migration happens in companies that perceive a strategic advantage in exploiting open source code. These companies chose to bet their business on this model and do not want their competitors to be aware of it, so there are few references. This approach can take the form of a new, “from the ground up” deployment, or be a corporate-wide directive to embrace Linux and migrate to it. The success of such an implementation depends on:

- ▶ Strategic, core-business application deployed on Linux
- ▶ Strong development, testing, and support staff
- ▶ Executive sponsorship
- ▶ Competitive advantage

6.3 Total cost comparison framework

In this section we discuss the components you should use to calculate the total cost comparison.

6.3.1 Staffing

There is a current and growing shortage of skilled IT technicians. Leveraging the effectiveness of employees you currently have, and attracting the skill set coming out of college, is a significant financial advantage. Some components of staffing costs are compared in Table 6-1:

Table 6-1 Staffing cost components

Penguin colony	Separate servers
One location	Many locations
One staff	Much staff, or downtime for travel
Centralized skill redundancy; provides overlap and backfill	Isolated experts; expensive, hard to duplicate for 24x7 coverage
Mass, automated customization	Manual customization

6.3.2 Hardware

Hardware price comparisons of single-use servers to a mainframe might seem to be an easy win for the single server. A production environment includes more than just a box. When the cost of hardware for failover, backup, peak capacity, and networking is included, the mainframe can become very competitive. A calculation of hardware costs should include these items:

- ▶ New servers: CPUs, keyboards, monitors
- ▶ Disk storage
- ▶ Networking hardware: switches, routers, racks and cabling
- ▶ Uplift for failover, backup, and peak capacity

6.3.3 Occupancy

Data centers represent large sunk costs, and server consolidation on Linux zSeries presents a huge opportunity to get the maximum return on that investment. Using the Linux Community Development System example, 600 servers exist in the floor space of a refrigerator. The cost per square foot for single-use servers, no matter how efficiently stacked, is going to be a significant financial burden. Occupancy costs include:

- ▶ Floor space
- ▶ Heating and cooling
- ▶ Electrical consumption
- ▶ Uninterrupted power supply

6.3.4 Other factors

The costs of staffing, hardware, software, and occupancy can be measured and projected with some accuracy. Those costs do not represent the whole picture, and it is important to consider the following:

- ▶ Cost of an outage
- ▶ Cost of migration
- ▶ Cost of exploiting emerging technology
- ▶ Cost of multiple databases
- ▶ Cost of multiple copies of one database
- ▶ Cost of a proprietary architecture
 - Non-portable code
 - Restricted interfaces
 - Removal of function

It may not be possible to put a cost figure on each of these, but one of them may be the determining factor that makes Linux on a mainframe the best choice in a particular situation.



Backup and restore

In this chapter we discuss how to effectively back up and restore Linux systems. We describe a number of backup methods, and focus on the use of VM and Linux tools.

7.1 Backup methodologies

Backup and restoration, particularly in large environments, involves more than simply making a copy of data and keeping it in a safe location (although that is often a good start). Instead, each different type of data loss scenario requires a different approach to the restoration of service.

7.1.1 Disaster recovery

In order to recover from a disaster such as loss of a computing center or disk subsystem, you must consider hardware considerations as well as data management considerations. Once you solve the hardware issues, you need to make all of your data available again as quickly as possible.

Taking device-level backups is the easiest way to do this. However, this tends to be the most disruptive and costly backup method. As an example, Point-to-Point Remote Copy (PPRC) is an example of device-level backup that is extremely effective in a disaster recovery role, but it requires that you duplicate your entire disk storage facility, which may be cost-prohibitive (this is discussed in more detail in 7.2.2, “Point-to-Point Remote Copy (PPRC)” on page 131).

Using VM DASD Dump Restore (DDR), another method of device-level backup, requires that you shut down the system using the disk being backed-up, which is also disruptive (as discussed in 7.3.3, “VM DASD Dump Restore (DDR)” on page 138).

7.1.2 Logical backup

It is often not appropriate to restore entire file systems at a time. For example, if a single file is accidentally deleted from a server, the most efficient method of getting that file back would not be to have to restore an entire file system.

Device-level backup processes are not appropriate for this purpose. Consider PPRC in this scenario: if the file is deleted on the main disk, within an instant the deletion is repeated on the mirrored disk.

The same thinking applies in the case of data corruption. An external backup solution must also be available so that the file can be restored to a time before the data corruption took place.

7.1.3 Backup types

There are generally considered to be two types of backup: *full* and *incremental*.

A full backup is a copy of the entire file system. This backup could be used to restore that file system if it was completely destroyed. In a device backup, this would be an image copy of the entire device. In a logical backup, every file in the file system would be copied.

An incremental backup only copies *changes from the time of the last backup*. On its own, an incremental backup could not restore an entire file system. Incremental backups are used as a way to keep full backups up-to-date, without having to repeat the entire full backup every time. Incremental backups are usually associated with logical backup methods, where directory information provides an easy way to find out which parts of the file system changed since the last backup.

7.1.4 Complex application backup

Applications such as databases, which generally keep very large files with complex internal structures, pose problems for backup processes due to this internal complexity. A backup program, looking at the file system level, only sees a large file, and must handle it as such. Even if only a single 1 KB record in a database file of 1 GB has been changed since the last backup, an incremental backup of the file system would still back up the entire 1 GB file.

One approach to avoiding this is to use application-specific tools to back up the application data internally. That way, the file system backup facility can be told to ignore the application files. For example, there are a number of tools for DB2 that function within DB2 to perform backups.

Another approach gives the file system backup tool the intelligence to look inside the application file format and treat it almost like another file system type to be backed up. The TSM agent for Lotus Domino is an example of this, allowing TSM to view documents inside .NSF databases and back them up individually.

7.1.5 In-service backup

One method of providing backup for application-specific data is to use a client of that application to back it up. This can reduce the overhead of running a backup client in addition to the application server. We refer to this strategy as “in-service” backup, since the backup happens within the service being provided.

For example, if you have a number of virtual servers providing DNS, the DNS zone data can be backed up by having another DNS server elsewhere in the environment configured as a secondary DNS. This secondary DNS will perform zone transfers to maintain its copy of the configuration, thereby creating a backup of the data. A similar approach could be taken with HTTP or FTP servers, using site mirroring programs such as **wget**.

In many cases, the overhead of a backup client on an application server will not be significant. However, from a security perspective, a highly secure server will have the minimum number of services running in order to reduce security exposure. In this case, an in-service backup strategy may be a suitable option.

7.2 Hardware possibilities

You have the following hardware possibilities for backup and restore processes.

7.2.1 FlashCopy

FlashCopy is a feature of the IBM Enterprise Storage Server that can create an identical copy of disk volumes without interrupting operating system access to the volume.

Note: Other storage systems have similar capabilities, often called by different names. Refer to the documentation for your hardware to check whether your storage systems support such a feature, or refer to your vendor.

This feature can assist the backup process in environments where it is not convenient to take systems down or to take applications offline. Using FlashCopy, data can be duplicated to alternate disk volumes, and then brought online to another system and backed up using traditional methods (tape, etc).

FlashCopy can provide the same support to Linux systems, but it is important to consider the impact of the buffer cache. If a FlashCopy is performed while there are buffered writes in cache, data integrity is lost. In a Linux environment, the following steps would have to be taken:

1. Suspend updates to applications and databases.
2. Flush the Linux buffer cache with the **sync** command.
3. If VM minidisk caching is used, ensure it is flushed also.
4. Initiate the FlashCopy.
5. When the copy is done, reopen applications and databases.

FlashCopy is designed to copy entire disks at once. This means that copying a single minidisk will generally involve copying the entire disk on which the minidisk resides.

7.2.2 Point-to-Point Remote Copy (PPRC)

Point-to-Point Remote Copy (PPRC) is a feature of S/390 and zSeries disk controller hardware that allows a remote disk unit to maintain a synchronized copy of the data held on a production disk. The remote disk unit does not have to be in the same location as the production unit (but distance restrictions do apply).

PPRC is used extensively in “traditional” S/390 installations to provide disaster recovery for entire disk enclosures. Since the data is synchronized at the time of write I/O, the remote disk is virtually identical to the local disk. If the local disk subsystem is lost, the channel paths to the remote disk can be brought online and processing resumes at the point of failure.

PPRC incurs a slight overhead in I/O duration, due to its synchronous nature. The operating system does not see the I/O complete until after the remote unit has successfully completed.

In a Linux scenario, PPRC can be used as part of a highly redundant and available permanent storage design. An example of this design is shown in Figure 7-1.

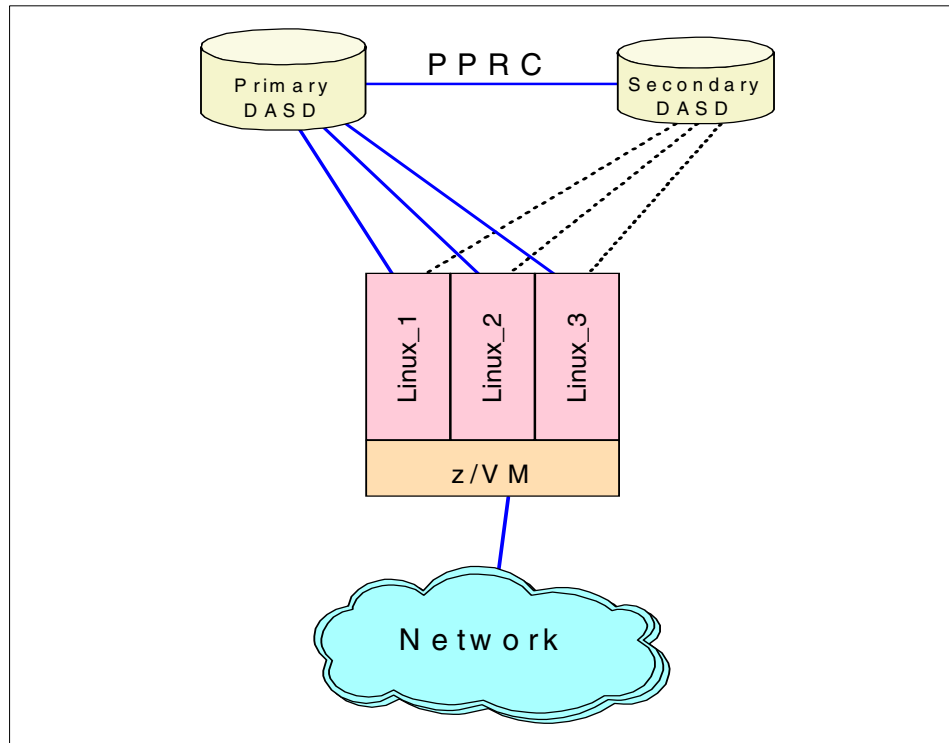


Figure 7-1 PPRC disk configuration for Linux

In this diagram, PPRC is used to synchronously mirror writes to a second disk subsystem. In the event of a failure of the disk, the paths to the secondary disk unit are brought online and processing can resume. However, while VM is able to handle these device switching non-disruptively in many cases, the Linux guest systems and the applications running in them should be shut down if the primary DASD is lost, and restarted with using the secondary DASD.

This design is enhanced and discussed further in 7.4, “High availability choices with zSeries” on page 139.

Note: In “Network block device” on page 137, we discuss a way to produce a similar mirroring method in Linux, which does not require PPRC.

7.2.3 IBM 3480/3490 tapes

IBM has written a driver for IBM Model 3480/3490 tape drives. This driver allows a mainframe tape device to be used by Linux applications and utilities. As with other device support code, it can be either compiled into the kernel or built as a module (tape390.o).

Important: The tape390 driver was written after the SuSE GA distribution was released, so in order to use the driver on this distribution, you need either to get an update, or to build the module (and a new kernel) using an updated source tree.

We used a beta of SuSE Enterprise Server for S/390 7.2, dated June 21 2001, which was built on a 2.2.19 kernel and did contain the tape390.o module.

Configuration

The driver takes a single parameter, which allows you to specify the devices to be used by the driver. For example, the following command would load the tape390 module, defining any tape devices between device addresses 0180 and 0183 and one at 0189:

```
insmod tape390 tape=0180-0183,0189
```

If you want to have the module loaded automatically when it is required, add the following lines to /etc/modules.conf:

```
alias char-major-254 tape390
alias block-major-254 tape390
options tape390 <module-options> # if you want to pass options to it
```

The specifics quoted here seem to be common in the documentation *Device Drivers and Installation Commands*, for both the 2.2.16 and 2.4 kernels.

The driver may search for all tape devices attached to the LINUX machine, or it may be given a list of device addresses to use. If it is not given a list the numbers allocated are volatile - the number allocated to any particular physical device may change if the system is rebooted or the device driver is reloaded. In particular a device brought online during a LINUX session will be allocated the next available number at the time it comes online, but at the next reboot it will be given a number according to the sequence of device addresses. If a tape= parameter is present at system startup or module load, all tape devices in the ranges of the specified parameter list will be used. The devices are then numbered (sequentially from zero) according to the order in which their subchannel numbers appear in the list.

In both cases the associations between subchannel numbers and device numbers are listed in the file /proc/tapedevices.

Operation

The driver interfaces with programs via /dev nodes, with the same name format expected by standard Linux tape utilities:

- ▶ Character mode:
 - /dev/ntibm n (non-rewinding)
 - /dev/rtibm n (rewinding)
- ▶ Block mode:
 - /dev/btibm n (for read-only)

Currently, a device major node number has not been formally allocated to this driver, so major node number 254 is being used until a formal allocation is made. To use the tape device driver, you must create the /dev nodes manually (unless your distribution has already created them).

The minor numbers are allocated in pairs starting from 0, with the even number representing the rewinding device and the odd number for the non-rewinding. The even number is used for the block device. The driver allocates drive number 0 to the first device found, 1 to the second, and so on.

For example, to create the /dev nodes for the first two tape drives present in the system, the following commands are used:

```
# mknod /dev/rtibm0 c 254 0
# mknod /dev/ntibm0 c 254 1
# mknod /dev/btibm0 b 254 0
# mknod /dev/rtibm1 c 254 2
# mknod /dev/ntibm1 c 254 3
# mknod /dev/btibm1 b 254 2
```

When a major number is formally allocated and the driver modified accordingly, all that will be needed is for the /dev nodes to be recreated (and the /etc/modules.conf entries, if present, to be edited) using the right major number. No changes to the programs that read or write tapes will be necessary.

Important: If you use devfs on your system, the entries in the /dev tree will be managed automatically, and will have a format derived from the device address of the tape drive being used.

For example, the rewinding device for the tape drive on device address 0181 will be referred to as: /dev/tape/0181/char/rewinding.

For more information on the tape driver, refer to *Linux for S/390 Device Drivers and Installation Commands*.

7.3 Software tools

You have the following software tool possibilities for backup and restore processes.

7.3.1 Software RAID

Using kernel support in Linux, multiple disk devices can be assembled into a disk array using Redundant Array of Inexpensive Disks (I/O RAID) principles. RAID provides many options for aggregating physical devices into contiguous storage, simultaneously providing fault-tolerance and a degree of disaster recovery capability.

Note: RAID is usually implemented in hardware, with specialized controller devices doing the disk aggregation and presenting a single disk volume to the operating system. For this reason, we have to refer to software RAID as a special case.

On S/390, Linux can utilize software RAID by creating a single RAID volume across disk devices on separate disk controllers.

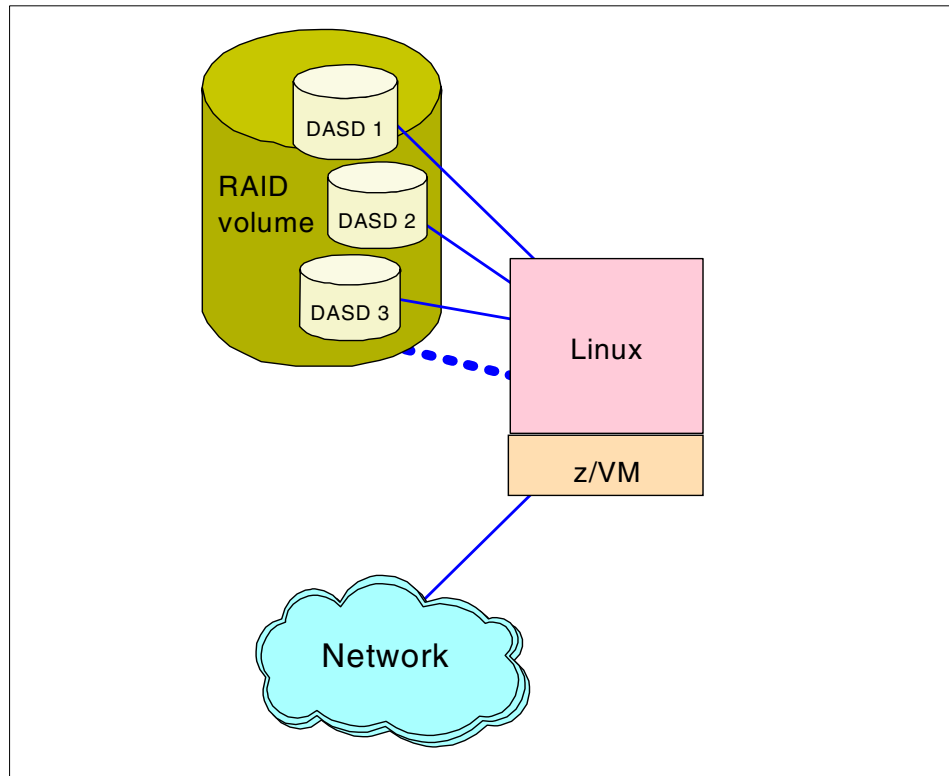


Figure 7-2 Software RAID example

In Figure 7-2, the solid lines represent the real I/O path to the physical devices involved, and the dashed line shows the path to the RAID volume created. Since the file system is created on the logical volume, file system I/O appears to be to the logical device. The RAID driver maps this to an operation to the physical disks.

The greatest benefit in an S/390 environment is gained when the physical DASD is distributed across multiple controllers. In this case, depending upon the RAID configuration used, the logical volume can continue to be available if physical disk is lost.

7.3.2 Network block device

The network block device is a Linux device driver which can provide access to a physical disk across a TCP/IP network. On its own, it does not offer much to Linux on S/390, but combined with a mirroring RAID configuration, it can create an off-site duplicate of a disk volume in a similar way to what PPRC does at a hardware level.

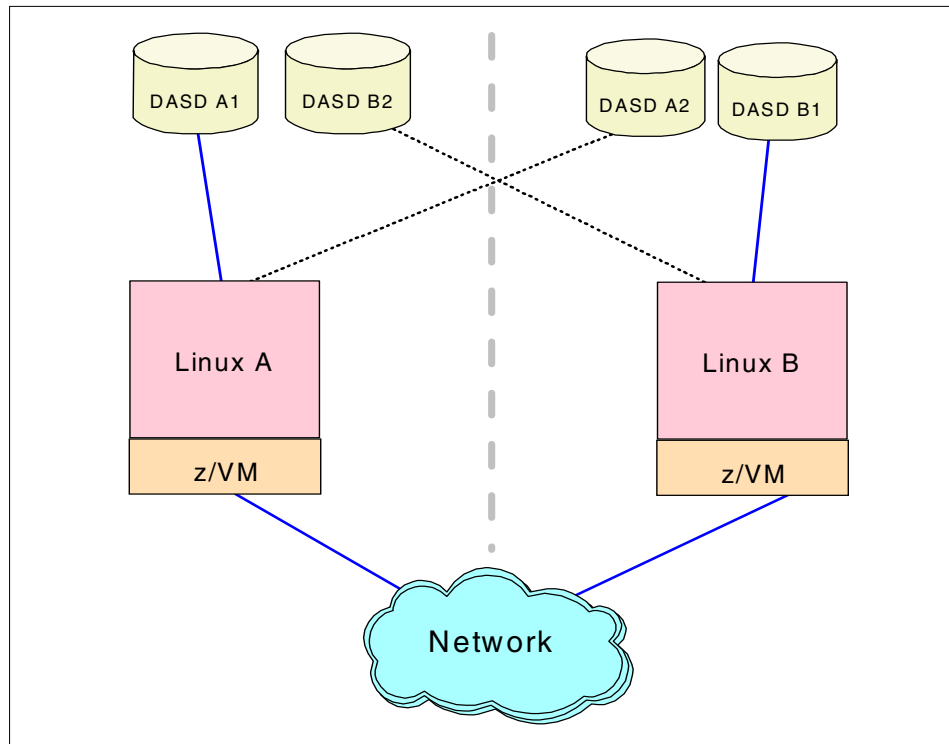


Figure 7-3 Network block device cross-site

In the example shown in Figure 7.3, Linux systems operate on two separate processors in two sites. Disk devices at both locations are accessed only by the Linux systems local to the site. Some of the disk, however, is linked via the network block device to the Linux system in the other site, creating the connectivity shown in the diagram (where logically, each system has access to disk physically at the local site, and via the network to the other site). By creating a mirroring-RAID volume over these devices, a cross-site mirrored device is created.

The network block device can be used in conjunction with distributed file systems (see “Global File System (GFS)” on page 51 for a brief description of how GFS can utilize the network block device).

Using this kind of configuration would provide a cheaper alternative to PPRC for sites that do not currently have the infrastructure to support PPRC. However, to maintain disk response times, high speed network connectivity is needed between the two locations. In certain high-demand applications, the cost of providing the support infrastructure for PPRC might be justified.

7.3.3 VM DASD Dump Restore (DDR)

DDR is a full-volume disk backup utility that can copy VM minidisks in their entirety. These backups do not provide any awareness of the contents of the volume being copied, they simply treat the volume to be copied as a disk file. DDR can be used as a means of providing full-volume backup and restore capability to Linux systems running under VM. Using DDR, entire systems can be dumped to tape, disks at a time.

Important: This operation is best performed when the Linux system is shut down. If you take a DDR copy of a Linux minidisk, any pending writes that Linux has in buffer cache will not be present on the backup. This will lead to data integrity problems. Also, since the Linux file system is mounted when the copy takes place, if a restoration is required a file system check will take place when it is first mounted. For ext2 file systems in particular, this may be undesirable.

DDR can be used to take a backup of a Linux system immediately after installation. In the event of a disaster, this backup could be restored and used as a starter system for subsequent file-level restoration using another tool.

7.3.4 Amanda

Amanda is an acronym for the “Advanced Maryland Automatic Network Disk Archiver”; an Open Source backup scheduler. Amanda uses a client-server arrangement to facilitate the backup of network-attached servers. Using Amanda, it is possible to have a single tape-equipped server back up an entire network of servers and desktops. We provide more detail about its use in Chapter 12, “Backup using Amanda” on page 269.

7.3.5 Tivoli Storage Manager (TSM)

TSM provides data-level, multiplatform, consolidated backup capabilities. TSM agents exist for a variety of server and desktop operating systems, including z/OS, z/VM, Linux for zSeries and S/390, Windows, and others. TSM backs up remote file systems over the network to a central point which copies the data to tape.

TSM offers a number of benefits:

- ▶ On supported platforms, TSM provides agents to allow data-level backup of file structures like DB2, Lotus Domino, and others. This means that TSM can back up individual documents within a Notes NSF database, for example.

Note: At this time, Linux for zSeries and S/390 is *not* one of the platforms supported for TSM agents, so TSM can only provide file-level backups.

- ▶ Integration with hierarchical storage management facilities, such as those included with DFDSS on z/OS.
- ▶ A single backup platform and scheduling point for the entire data center.

TSM is also described in the IBM Redbook *Linux for zSeries and S/390: Distributions*, SG24-6264, which is available on the Web at:

<http://www.ibm.com/redbooks/abstracts/sg246264.html>

7.4 High availability choices with zSeries

In the preceding sections, we describe the procedures to backup and restore VM and Linux data. But whether whole minidisks are backed up using DDR, or incremental copies of single Linux datasets are produced by TSM or Amanda, these backup copies are all out of date the moment after they have been created. And if data has to be restored by using these backup copies, then updates that were issued to the data—after the last backup copy was produced—are usually lost.

In cases where data is destroyed in a logical way (e.g. by some erroneous program code), then going back to an old but safe copy of the data is probably acceptable. But what happens if all data belonging to several Linux guests—or even to the entire VM LPAR—is destroyed?

In such a case, a disaster recovery effort is required. As pointed out previously, restoring data from backup copies created by DDR and TSM is very time-consuming. New hardware has to be provided, installed and defined; the data from the backup copies has to be restored; and changes to the data after the last backup copy was made will still be missing. Similar problems arise if the server itself is damaged in a disaster.

Traditional architectures provide a reliable environment

With many Linux servers under VM, the traditional S/390 and zSeries configuration for setting up high availability computing can be exploited to provide a reliable Linux operating environment.

S/390 and zSeries architecture is designed for continuous availability, which means that the services have to provide both high availability (the avoidance of unscheduled outages) as well as continuous operations (the avoidance of scheduled outages). In an OS/390 or z/OS environment, this goal is usually reached by exploiting the design of a Parallel Sysplex, enriched by remotely mirroring DASD devices to build a Geographical Dispersed Parallel Sysplex (GDPS).

While Linux is not able to participate in a Parallel Sysplex, all other architectural design points a GDPS is based on can be used to design a highly available Linux production environment on zSeries. This includes the use of remote DASD mirroring to provide consistent copies of all data, the use of Capacity BackUp (CBU) to enable required computing power at the surviving server, and the use of automation to handle all procedures for bringing up the backup systems and redefining network and channels.

Figure 7-4 on page 141 shows a sample scenario, which we discuss in detail.

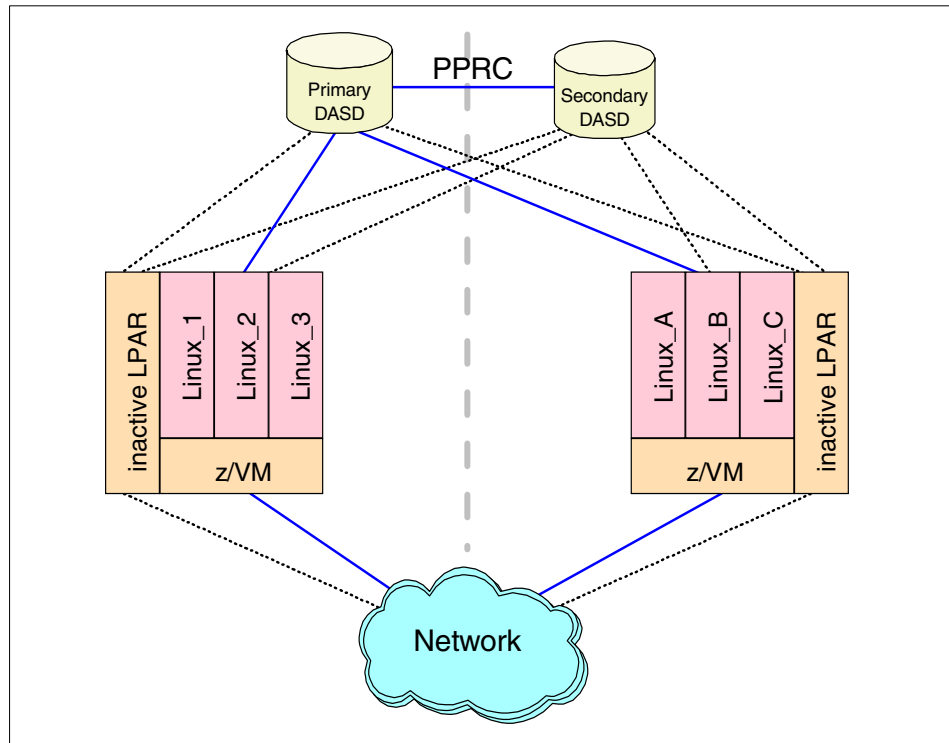


Figure 7-4 High availability Linux scenario, normal operations

In this example, the Linux servers operate in two z/VM LPARs on two zSeries servers, which are located in two physically separated computing centers (indicated by the dotted grey line between the servers). Each of these zSeries servers also contains a defined, but inactive, LPAR, which is supposed to take over the workload of the other server in case of a disaster.

In order to offer sufficient computing power to this LPAR without hampering the work of the Linux guest systems in the other productive LPAR, additional redundant processors are available that can be concurrently activated by using the Capacity BackUp (CBU) feature. CBU is available on 9672 and zSeries z900 servers; refer to the IBM Redbook *IBM @server zSeries 900 Technical Guide*, SG24-5975, for more details. It is available on the Web at:

<http://www.ibm.com/redbooks/abstracts/sg245975.html>

The data belonging to the z/VM operating systems of both zSeries servers, as well as the minidisks of all the Linux virtual machines, are located on a DASD control unit (CU) attached to both servers, using Fibre channel CONnections (FICON) or Enterprise System CONnections (ESCON). The DASD devices are remotely mirrored to another DASD CU, by using the synchronous Peer-to-Peer Remote Copy (PPRC) abilities, provided, for example, by the 2105 Enterprise System Server (ESS).

The operating systems (z/VM and Linux) are *not* aware of this mirroring; only the connections to the primary DASD CU are activated. Nevertheless, connections to the secondary DASD CU are defined and cabled (indicated by dotted lines), ready to be used in the case of a disaster.

Also, the network connections are defined to both LPARs in each zSeries server, but only the connections to the productive LPAR are active.

7.4.1 Loss of a DASD control unit

The following scenario discusses what happens if the primary DASD control unit (CU) fails, as illustrated in Figure 7-5 on page 143.

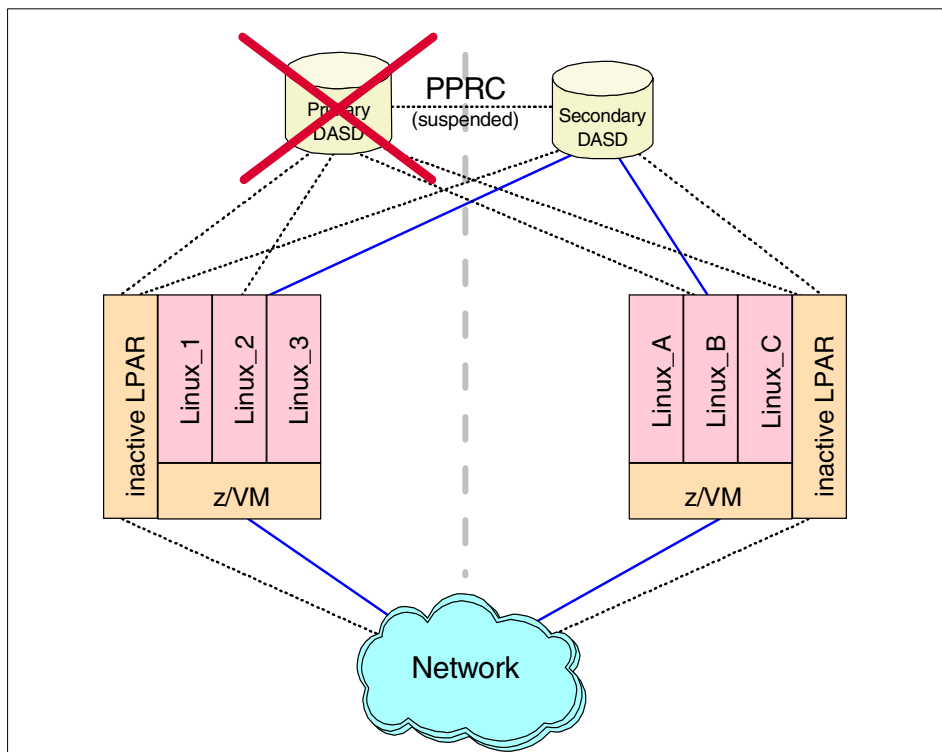


Figure 7-5 High availability Linux scenario, after failure of DASD control unit

If the primary DASD CU becomes unavailable for some reason, the mirrored data on the secondary CU has to be used. In order for this to happen, the devices of the primary CU have to be configured offline to all attached operating systems, and the devices on the secondary CU have to be configured online.

Because the secondary devices are simply mirrored images of the primary ones, they contain the same data and even have the same VOLSER, and the failed I/O operations of the attached systems can be set up again on the secondary DASD hardware.

Depending upon how fast the switch is performed, and on the time the applications and operating systems accept a delay in I/O operations, the applications may continue to work without being disrupted. However, with Linux running as a VM guest operating system, all Linux images affected by the failing primary DASD should be shut down and rebooted from the secondary DASD.

It may be possible, depending upon how often the Linux system accesses the minidisks on the failing DASD, that the secondary DASD can be configured online by CP, without Linux having noticed the temporary absence of the minidisks—or it can at least be possible to repair these minidisks with Linux still running. But this will require manual intervention in the Linux system to ensure that the restored minidisks are working correctly and the data is not corrupted.

Therefore, for ease of use and to ensure data integrity, in the case of the loss of one or more DASD volumes, we recommend the following:

- ▶ Configure the failing devices offline to VM.
- ▶ Shut down the affected Linux systems.
- ▶ Configure the secondary devices online.
- ▶ Restart the Linux systems with using the secondary devices.

With the loss of the primary DASD CU, mirroring with PPRC is of course suspended, and further operations have to continue without mirroring until the CU is available again. The failure of the secondary DASD CU, the failure of the PPRC connection between the both CUs, and the loss of the access to one or several devices also need to be considered.

For example, if the zSeries server is able to issue a write I/O operation to the primary DASD device, but the primary CU is not able to operate mirroring to the secondary CU because the connection between the CUs units has failed, the I/O by default will not be completed. In this case you have to decide if the I/O operations to the primary CU should be resumed without mirroring, or if the operations have to be stopped.

The primary DASD CU does not know why there is no response from the secondary CU; possibly the entire second computing center has been destroyed, or it may only be the result of a weak cabling connection. For this reason it will probably be necessary to get more information regarding the state of the secondary CU before deciding how to continue with the I/O operation.

The procedures to gather the necessary information, the rules for making the appropriate decisions, and the execution of the required commands have to be coded and established by using automation utilities. Besides the basic hardware functionality, this is one of the most complex topics of a GDPS.

7.4.2 Loss of a S/390 or zSeries server

The next scenario involves the failure of a whole zSeries server. If a disaster strikes the computing center, and the server is not able to continue operations, the workload has to be transferred to the surviving server in the other computing center. This is illustrated in Figure 7-6 on page 145.

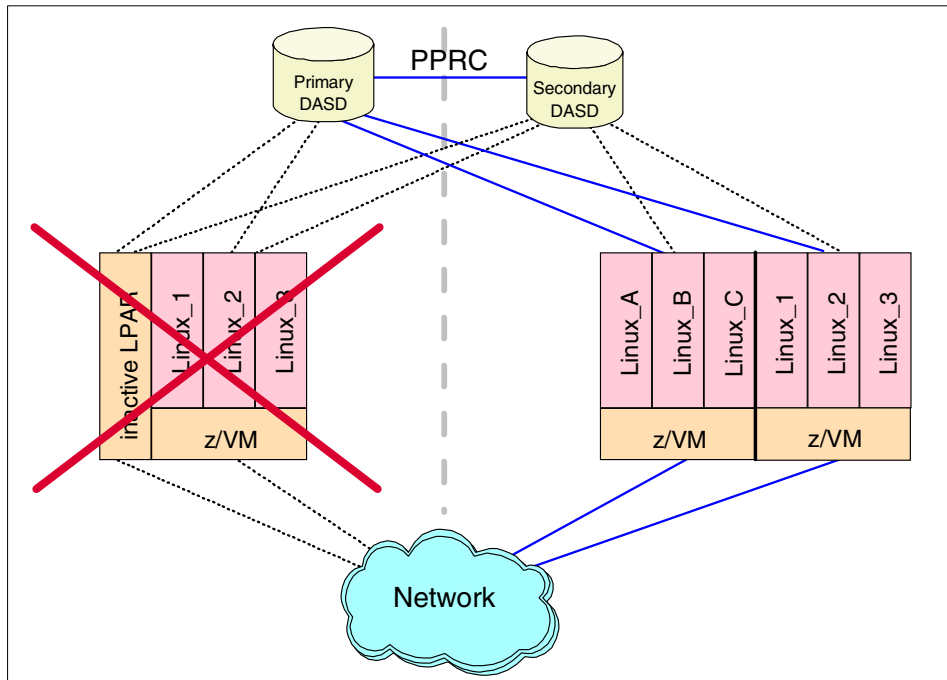


Figure 7-6 High availability Linux scenario, after failure of server system

On the surviving server, an additional LPAR has already been defined for disaster recovery purposes. This LPAR is now activated by loading the operating system from the same DASD devices as the failing system used, using the already installed connections to the primary DASD CU. The PPRC mirroring operations between the primary and secondary DASD CU continue without any changes.

While the LPAR activation on the server can take place immediately after the failure of the original server, the processor capacity of the surviving server has to be adjusted as soon as possible, in order to meet the needs of the combined workloads of both servers. This can be done by exploiting the 9672 or 2064 z900 CBU feature, which allows you to activate physically available but unused processors non-disruptively.

Resuming network connections to the restored LPAR is also no problem, since the LPAR activation is done with exactly the same data as the failed original LPAR, and the same network addresses as before are used. The OSA of the new LPAR on the surviving server makes itself known to the network with these addresses, and network operations resume as before; the fact that the physical hardware has changed is transparent to the routers in the network.

In summary, the hardware capabilities of PPRC and CBU allow you to achieve a high availability computing environment based on Linux under VM on S/390 or zSeries. But the scenarios we presented require a thorough understanding of the reasons for failures, the actions that have to be taken to recover from failures, and the consequences of these actions. Although it is possible to handle such recovery tasks manually, as computing environments become increasingly complex, there is a greater need for automated processes.



Performance analysis

In this chapter we discuss the performance analysis and tuning of Linux images running under VM.

Performance of a large system is critical to its success, only one step away from functionality. Once a system has been developed and is functional, the next question is—what is the price performance? If price performance is very low, a system and service is more likely to succeed. The intent of this chapter is to help you realize an optimal price performance.

8.1 Performance considerations

The aspects of performance measurement and tuning that are needed in environments serving many users are extensive. This chapter reviews server resource analysis, subsystem analysis, server delay analysis and some level of response time analysis. The methods of storing performance data is also a consideration; is report format sufficient, or is a performance database necessary to allow detailed analysis—and even accounting functions—to be performed? In an environment where service is associated to charge-back, the performance database becomes more important.

S/390 and zSeries systems are large systems with many potential points of contention, and many places of potential optimization. Each subsystem can be complex and have many options. Any subsystem, when overutilized, can be a global system bottleneck. As your workload changes or grows, utilization of one resource can reach a threshold that causes a significant *global* resource shortage. On a system where changes on one server can impact performance of another, you need to monitor the environment carefully.

Many early Linux for S/390 installations did not have VM performance skills—and ran into problems which easily could've been avoided. For example, the most common complaints often were “things stop” or “this system is slow”. Both of these issues can result from very simple configuration problems.

VM has a very sophisticated scheduling mechanism that is designed to efficiently support thousands of users, even when resources are overcommitted. Linux has some features that impact the view of resource utilization. Understanding storage and page space requirements when designing a system allows you to avoid configuration and utilization problems.

8.2 Why measure performance

Performance measurement and tuning has a cost in terms of staffing and time. In environments where the cost of purchasing a new system is less than the cost of analyzing the performance of a system, it is common to not spend much time on performance analysis. However, in a zSeries and S/390 environment, most installations will require both real time analysis as well as a structured methodology for capacity planning. This section describes methodologies for both real time performance analysis and capacity planning.

Projecting requirements when moving applications from an NT or Linux/Intel environment to zSeries and S/390 is important, in order to ensure a high level of success. If the resource requirements of an application exceed what is available or what makes economic sense, then that needs to be understood prior to moving the application to zSeries and S/390. Choosing applications to run effectively on zSeries and S/390 will greatly increase the chance of success.

8.2.1 Cost of running applications

In environments such as ISP and ASP, where charge-back is reasonable, the cost of running each application is a consideration. Each application should be evaluated to determine which platform is most cost effective. Some applications will be more cost effective on dedicated RISC processors, while others will be more cost effective on zSeries and S/390.

Applications that require resource for only short periods of time are very effective in an S/390 time-sharing environment, because when the applications are idle, they do not consume processor or storage resource. As the applications become active, they are brought back into memory. The cost of storing the application on disk (page space) can be measured in cents-per-megabyte and is usually insignificant.

However, applications that require significant dedicated resource have a very different cost model—instead of using an amount of storage for a small percentage of time, these applications require storage most of the time. For these applications, there is no choice other than to ensure the resource is provided all the time. Because the cost of dedicating resources is much higher, the cost of running this application is also higher and potentially should be put on a platform suitable to the requirements. This may be an LPAR, or a non-S/390 server; the platform decision should be based on what is economically justified.

8.2.2 Controlling costs

Once the cost of running an application is known, you'll want to monitor those costs. If an application or server suddenly increases its resource requirements, either due to programming errors or increased workload, the cost of running that application will rise. Given a charge-back environment, there will be issues if the additional costs are not recognized as they occur; sudden increases in monthly costs can cause problems. By monitoring costs as they are incurred, both customers and service providers will have a better understanding of the provided service.

To ensure the cost of running an application does not increase without management awareness, you will need to have a performance monitoring methodology. This methodology should include thresholds for resource use and should be monitored programmatically. Mechanisms for alerting management of unexpected increases in resource requirements are required components of managing this type of service.

8.2.3 Controlling the impact of one application on another

In environments where there may be several different customers and where Service Level Agreements are required, there's a need to monitor each application and server on a real time basis to ensure that problems with one customer do not impact other customers. It is common practice to provide an automated alert system to check operational values against threshold settings. Details on implementing this function are discussed in 8.13, "Alerts" on page 175.

8.3 Measurement tools

Following is a list of some of the measurement tools available on VM:

- ▶ ESALPS from Velocity Software
See 8.3.1, "Measurement tool used" on page 151.
- ▶ FCON/ESA from IBM
FCON/ESA provides performance monitoring capabilities with system console operation in full screen mode. FCON/ESA can give you an immediate view of system performance, or post-process its own history files. Threshold monitoring and user loop detection is also provided, as well as the ability to monitor remote systems. The most recent enhancements include support for remote access via APPC, virtual disk in storage reports, and enhanced minidisk cache reports.
- ▶ Real Time Monitor (RTM) VM/ESA from IBM
With RTM VM/ESA, you can get an immediate view of current system performance. Use RTM VM/ESA for short-term monitoring, analysis and problem-solving. It can simplify performance analysis and the installation management of VM/ESA environments. The latest RTM VM/ESA service includes support for the RAMAC array family, and support for RTM VM/ESA to run in 370 Accommodation mode on VM/ESA Version 2.
- ▶ VM Performance Analysis Facility (VMPAF) from IBM

Use VMPAF to statistically correlate and chart system performance problems, tuning information, and trend analysis. VMPAF does this by analyzing the relationships between variables from VM MAP, VM PRF, monitor and other data sources in order to determine which subsystems are most impacting your current system performance. Then, using interactive graphics, VMPAF gives you a quick, clear picture of these relationships.

- ▶ VM Performance Reporting Facility (VM PRF) from IBM

VM PRF uses your system monitor data to analyze system performance, and to detect and diagnose performance problems. VM PRF provides interim reports, plus summary and trend data that show resource use by user ID or user class, response time, DASD activity, channel utilization and system throughput.

8.3.1 Measurement tool used

The measurements in this chapter were performed using the Linux Performance Suite (ESALPS), a commercial product from Velocity Software that is available on the Web at:

<http://VelocitySoftware.com>

The ESALPS components we used were: ESATCP, for gathering Linux and Network performance data; ESAMON, for processing the VM performance monitor records and data gathered by ESATCP; ESAMAP, to provide reports for long-term analysis. ESAMON and ESAMAP are based on the CP monitor providing current performance data and long-term historical data. (ESAWEB, the fourth component of ESALPS and a VM-based Webserver, was not used for these measurements.)

ESAMON creates a Performance Data Base (PDB) to store performance data. This data is then used as input for reporting in many forms. There are two forms of the PDB, referred to in the ESALPS documentation as “History files”. The first form has a one-minute granularity and allows a detailed analysis. The second form has a 15 minute granularity and is long term, with each day’s data being summarized.

In this chapter, we provide examples and give recommendations about how to use these effectively. The reports and real time displays provide the performance information in the same format. Each report and display has a name (such as ESAUSRC) which provides user configuration data. Menus and tables of content help users find the needed reports and displays. Performance reporting is performed for:

- ▶ User data, showing resource requirements by user, user class, accounting codes.

- ▶ Response time data, showing response times based on the CP definition of transactions.
- ▶ Processor subsystem, showing details of all processors and LPARs.
- ▶ DASD and DASD Cache, showing DASD response times, by I/O component, cache controller data showing cache hit information, read/write data, etc. MDC hits (I/O satisfied by the CP minidisk cache function) by device are provided, as well as MDC hits by user. Data is provided both by device and by control unit. Channels are measured, and seek analysis is provided.
- ▶ Storage subsystem showing user storage, MDC storage, CP storage and available storage.
- ▶ Paging/Spooling subsystems, showing device and system activity, as well as utilization.
- ▶ Non-DASD I/O showing tapes, network controllers, channel-to-channel adapters and any other attached device, showing both by device and by control unit.
- ▶ TCP/IP data showing traffic at each layer of the IP stack (Transport layer, IP Layer, Interface/Hardware layer), and for the local VM stack, traffic and network response times by subnet and by application.
- ▶ Linux data showing by resource utilization by application (processor and storage), disk utilization, storage use for cache and buffer. Data is provided for any Linux being monitored.

8.3.2 Screen display

Screens can be displayed by from a CMS virtual machine executing “ESAMON screen”. The examples in this chapter can be displayed in this manner. For example, issuing the command **ESAMON SMART** gives you the screen shown in Example 8-1. This screen is automatically updated every 60 seconds.

Example 8-1 Output of ESAMON SMART command

```

TUNER: SMART      ITS0                                08/09 13:51-13:52
Seconds until next interval: 57                        2064 40ECB
  <-----Top Users-----> <-----Servers----->
  Userid:   CPU%   IO/Sec Pg/Sec  Userid:   CPU%   IO/Sec Pg/Sec
1) ESAWRITE 0.33    0.80    0   VMLINUX9  1.8   27.67  0.02
2) VMRTM    0.23     0      0   VMLINUX7  1.7   0.75   0
3) ESASERVE 0.00     0      0   VMLINUX2  1.1   0.45   0
4) ESATCP   0.00     0      0   TUX8MSTR  0.3   0.37   0
5) HUB6     0.05     0      0   VMLINUX6  0.3   0.02   0
6) SNMPD    0.00     0      0   TUX60002  0.2     0     0

<-----CPU Statistics-----> <---In Queue User statistics---> <-Page->
%cpu %usr %prb %sys %ovr %idl  InQ  Q0  Q1  Q2  Q3 Eli Ldng <-rate->

```

8.4 Measurement data sources

With environments including Linux servers and VM hosts all linked in a network, there is a need to measure each environment to understand the full system. However, each environment of the system has different measurement requirements and data sources, so combining the different data sources to understand your capacity and performance will be the challenge. Following is a list of the common data sources for each environment.

- ▶ Network - the most common method of evaluating network performance is by using a Simple Network Management Protocol (SNMP) data source. This is well defined and includes many network performance metrics. It has been extended by many vendors and open source groups to include information found useful for specific network components. SNMP Version 2C is the most common. This includes password protection of the data (community names), and performance enhancements in the protocol.
- ▶ z/VM - measuring VM is typically done using the CP Monitor as a data source. This technology scales very well and is widely used. The monitor reports on almost everything that has been thought to be useful on reporting subsystem and user performance for VM systems. As new releases of z/VM come out, and new releases of TCP/IP appear, the monitor is enhanced to provide instrumentation for new features.

Most performance analysis products suitable for a large environment will be based on the CP monitor. An alternative to using the CP monitor is to use a diagnose interface to look at internal VM control blocks. This provides access to most data. The downside is that every release of VM requires the data interface to be updated, whereas the CP Monitor is automatically updated with each data source, allowing users of the CP monitor to run without impact on new releases of VM.

- ▶ Linux - how to measure Linux from a global perspective is new technology. A good open source performance data source is NETSNMP. Linux is being enhanced significantly and the associated data source must be enhanced as well. NETSNMP provides performance data accessible to network monitors using SNMP. This performance data includes the network traffic, a set of private MIBS from University of California/Davis (UCD MIBS), and HOST MIBS that are defined in RFC 1157. The HOST MIBS provide data on applications for processor and storage requirements, as well as data on each

device of the Linux system. There is active development of NETSNMP, with new releases regularly.

NETSNMP is included in the three major Linux distributions (Red Hat, SuSE, Turbolinux) providing a common data source for analyzing performance. It can be found on sourceforge at:

<http://net-snmp.sourceforge.net/>

See 13.6.2, “SNMP installation” on page 306 for more detailed information on installing NETSNMP.

- ▶ CP Monitor - there are many options as to what data to have the monitor produce. For real-time monitoring, an interval of 60 seconds is common, balancing the overhead of collecting the data with the problem of a very long interval that hides performance spikes.

Measuring Linux wait states seems to be much more useful with a state sampling of .1 (10 times per second). The overhead of this method seems to be immeasurable, but it's a reasonable place to start. If you are using ESALPS, then all domains should be enabled except for SEEK and SCHEDULE (these should be enabled when you are performing specific analysis and require SEEK or SCHEDULE data).

8.5 Local vs. global performance tuning

In an environment with many virtual servers, having the technology and personnel to optimize the environment is a necessity. Optimizing a large, single, expensive system has significant payback, while optimizing small minicomputers often takes only a few minutes (or consists only of paying for inexpensive upgrades). However, tuning a large system that shares resources entails different requirements, ones that may be surprising to installations that are new to the zSeries and S/390 environment.

Virtual machines (Linux servers) should be tailored to the applications. Allocating resources to one virtual machine takes resources away from other applications—the difference between thinking globally and thinking locally. Minimizing resource requirements for one application means more resources are available for other applications; this reduces your overall costs.

Recognizing there are performance people from two radically different environments (local and global), tuning and performance analysis must be designed for the appropriate environment.

8.5.1 Local environment

Local environments use dedicated servers and tend to have one application on a server; they then tune that server to meet the application requirements. For some applications (Web serving, Lotus Notes, print serving), multiple servers might be dedicated to one application. The benefit of this approach is that work occurring inside one server does not impact work on other servers. Because the cost of each server is small, the cost of incremental growth is not necessarily a capital expenditure.

8.5.2 Global environment

The traditional performance methodology is two to three decades old and is based on global optimization of a system, evaluating systems with many applications sharing resources. In this global type of environment, resources are typically more expensive and therefore sharing those resources is a primary objective.

When one application or user on the system consumes large amounts of a resource, it impacts other applications or users on the system. Thus, global optimization requires you to evaluate all aspects of performance—from both a resource subsystem perspective and from an application perspective. Most large installations have dedicated personnel just for performance analysis, capacity planning and system optimization.

With current directions, and with cost justifications for moving many smaller servers to fewer and larger zSeries and S/390 systems, the optimization perspectives must be global—one server in a virtual machine can impact other servers on that system.

8.6 Linux operational choice

There are two methodologies for operating Linux; probably the most efficient is to use a mix of both. One is the typical VM/CMS methodology, where one virtual machine runs one (and only one) application. The other methodology is more like a typical VM/VSE or centralized Linux server environment, where one server runs many applications. The advantage of running small servers with only one application is that the server can be tuned for the application. The advantage of the larger server running many applications is a reduction in overall overhead.

Security considerations also influence the operational choice; having many applications running on only one server increases the risk an application falling prey to a hacker allowing access to data from multiple applications. Using single application servers greatly reduces your security risks.

See Chapter 5, “Security architecture” on page 105 for a discussion on security.

8.7 The CP scheduler

The multiprogramming level of z/VM is probably higher than that of any other platform. With possibly tens of thousands of users, each running their own programs and environment, there is a requirement for sophisticated task management and scheduling. The CP scheduler provides this function. The scheduler determines when users run, how often they run, which users to restrain when a resource is constrained, and many other functions. There are several controls that installations can use to tune the scheduler. Generally, the default values for these controls are appropriate.

Note that the scheduler has been tuned to run thousands of CMS (single tasking) users, as well as 10 to 20 large multitasking operating systems such as VSE, TPF or OS/390. Some installations may run even 100 larger guests. The operational considerations of running thousands of Linux servers on the z/VM system are not completely known. Education and new methods of tuning will likely be required.

8.7.1 Queue definitions

The scheduler categorizes each user by how long it has been running. Short tasks fall into queue 1, intermediate tasks are in queue 2, and long-running tasks are in queue 3. There is also a queue 0 for high priority work.

Linux servers that run a timer to do some small amount of work every 10 mS break this model. With the timer interrupt every 10 mS, CP classifies any Linux server as a long-running task, and will put it in queue 3. An implementation of the timer routines in Linux without using the 10 mS interrupt has been proposed, but is not yet available in the mainstream kernel sources.

However, measurements of a preliminary implementation showed the expected reduction of CPU resource usage for idle Linux servers. The measurements also showed that CP again was able to distinguish transactions, and did not classify every Linux server as a queue 3 user all the time.

When the Linux server is dispatched less frequently, you will have more control; long-running jobs are likely more resource-intensive, and you can reduce the number of queue 3 servers allowed to compete for resource. Reducing the concurrent number of tasks competing for resource then reduces the contention felt by the shorter tasks.

8.7.2 Global controls

The two most useful SRM controls are the DSPBUF and LDUBUF.

DSPBUF is used to control the number of users accessing the processor. Generally, the DSPBUF is not needed; however, when your processor is constrained, you can use the DSPBUF to limit the number of queue 3 users allowed to compete for the processor resource, which will in turn reduce the processor utilization. Thus, even in a very processor-constrained environment, short tasks will run very fast.

LDUBUF is used to control the number of users allowed to page. The default value of LDUBUF allows a system to thrash (the point where pages are being paged out for one working server to allow another working server to resume work). If you get to this point, the only solution is to reduce the number of servers competing for the paging resource. Lowering the LDUBUF from its default value does that. Linux servers have working sets that are variable and typically very large. Because of this, the scheduler may not react as fast as you'd like to current storage demands, so you may need to use STORBUF and XSTOR to achieve the desired effect.

STORBUF limits the amount of storage in use by dispatchable users. The STORBUF control is more often a hindrance than an assist to performance. Most performance people recommend raising the STORBUF control in order to make it non-operational. The XSTOR operand tells the scheduler to treat some percent of expanded storage as main storage (the usual recommendation is 50% for this value).

Virtual machines that are being held back due to resource constraint are kept on a list called the "eligible list". If you never have any users on the eligible list, the scheduler is not detecting a shortage of resources that could be alleviated by holding some users back. Thus, if LDUBUF is holding users back, then users would otherwise be loading in working sets which the paging subsystem may not be able to support, the DSPBUF reduces the number of dispatchable users, and the STORBUF limits the amount of storage in use by dispatchable users.

Following are the shortcut recommendations for scheduler controls. The numbers following the set command are for: a) all queues, b) queues 2 and 3, and c) queue 3. This allows you to set the amount of contention for each resource by queue.

Note: You should raise the queue 3 value of DSPBUF if processor utilization never exceeds 90%, and lower it if processor utilization often is at 100% for long periods of time.

```
SET SRM STORBUF 300 250 200
SET SRM DSPBUF 32000 32000 30
```

```
SET SRM LDUBUF 80 60 40
SET SRM XSTOR 50%
```

One method for measuring the impact of the scheduler on users is to look at the ESAUSRQ display, which provides most of the needed information. Knowing when there are users that are being held back by the scheduler because of the settings for LDUBUF or DSPBUF tells you which resource is constrained.

8.7.3 Local controls

There is a local control for each server that should be used sparingly. Setting a virtual machine to QUICKDSP tells the scheduler to never hold this user back, regardless of resource constraints. Virtual machines such as your TCP/IP gateways, security monitors, and the operator, should have this option.

Use QUICKDSP only to protect specific virtual machines when resources are very constrained. Using it too often will disable one of the most important features of the scheduler: the ability to survive serious resource shortages.

Priority between virtual machines is provided by the use of SHARE settings. There are two options for SHARE settings, relative and absolute. Users with a *relative* share will get a smaller overall share as more users logon. Users with an *absolute* share maintain their share regardless of the number of other virtual machines.

Thus, users such as TCP/IP or security managers should be given absolute shares as their workload increases when more users logon. Relative shares are used to control how the remaining resource is divided up between the rest of the virtual machines. The following recommendations can be given.

- ▶ Set the shares to absolute for all service machines that you expect will need to provide more work as more virtual machines are created, and set the shares to relative for all others.
- ▶ Set the ABSOLUTE shares to the peak required value (for example, 8% for TCP/IP if that is TCP/IP's requirement at peak load).
- ▶ Using the default of RELATIVE 100 is recommended unless you have a need to prioritize work.

Do not use very high relative shares, because using high relative shares for one or more users reduces your ability to prioritize your production work. For example, if there are 10 servers, and all servers are relative 100, then each server is scheduled to obtain about 10% of the system.

The arithmetic is quite simple: the “Normalized Share” is the relative share divided by the total of the relative shares. If one of the servers is then given a relative share of 200, that user gets a significant increase of about 9% (from 100/1100 to 200/1100).

Giving one user (TCP/IP, for example) a relative share of 10,000 means that each default user has a share of 100/11000, or less than 1%. There is no need to confuse the scheduler by reserving 90% of the resources for a guest that only needs 5%. Proper tuning of your system means allocating absolute shares to your key service machines.

A second part of the local controls is setting a cap on how much processing power a user is allowed to absorb. There are two reasons to do this: for accounting (to ensure users do not get more than what is paid for), and when users loop, or run very CPU-intensive jobs (to minimize their impact on other servers).

The following sets a target average share, but limits the server to 5% of the processing resource. The effects of the LIMITHARD are measurable on the ESAXACT (transaction analysis) report.

```
SET SHARE REL 100 ABS 5% LIMITHARD
```

8.8 Processor subsystem

Knowing how much processor is used, and by which servers, is information you need to know for efficient capacity planning. Controlling the rate at which your servers access the processor is done by setting Shares. Share settings have a minimum value and a maximum value, with several options and variations of each.

Using the CP monitor, you can capture over 99.9% of the processing power used. Building a processor map showing how much processor is used by LPAR, VM, Linux servers, VM servers, and CMS allows you to project future requirements based on new users or customers.

One of the issues seen in the past was the following: an important Linux application was ported in a fashion guaranteed to produce poor performance. As the Processor Local Dispatch Vector Report (ESAPLDV) in Example 8-2 shows, there were about 70,000 dispatches per second on each processor—this should be about 1000 on systems that are running well. This overhead was very costly; running anything 211,000 times per second would intuitively have a very high cost!

This is the kind of potential problem that's extremely hard to diagnose without the proper tools. An installation might perceive that S/390 performance is bad—when, in reality, a simple correction to the application might eliminate 200,000 calls to a function that does not need to be called.

On a dedicated processor, this might not be an issue. However, on a zSeries or S/390 system where most resources are shared, this application would be inappropriate to run as it performs in this example.

Other items to examine in the report are the number of steals and moves per second (low numbers are desirable). The Moves To Master value indicates how many calls are made to functions that must be single-threaded on the master processor. High numbers indicate use of functions that may not be appropriate for a high performance application.

The PLDV Lengths values show the number of virtual machines waiting on each processor queue, indicating the current level of multiprogramming. This is a good example of the need for VM performance analysis *and* Linux performance analysis.

Example 8-2 Sample Processor Local Dispatch Vector Report (ESAPLDV)

<VMDBK Moves/sec>		<-----PLDV Lengths----->						Dispatcher	
CPU	Steals	To Master	Avg	Max	Mstr	MstrMax	%Empty	Long	Paths
-	-----	-----	-----	-----	-----	-----	-----	-----	-----
0	823.8	0.7	0.2	1.0	.	.	83.3	70489.8	
1	111.1	0	0.4	1.0	.	.	55.0	70454.2	
2	196.4	0	0.4	2.0	.	.	61.7	70056.8	
	-----	-----	-----	-----	-----	-----	-----	-----	-----
	1131.3	0.7	1.0	4.0	.	.	200.0	211000.7	

8.9 Storage subsystem

Lack of storage to meet the requirement results in paging, and paging causes delays in service. Monitoring the storage requirements and the impacts of applications provides necessary feedback for capacity planning. There are many ways to reduce storage, and on VM there are different types of storage. For storage capacity planning purposes, you should maintain a map of your storage to understand the requirements for VM, Minidisk Cache, Linux user storage (by customer), VM Servers (TCP/IP, management service machines), and CMS users, if any. This map should be maintained for both Expanded Storage and Real Storage.

8.9.1 Storage options

For storage (memory), there are several options, each with different impacts on Linux, applications, and your global resources. Coming from a minicomputer or microcomputer environment, administrators have been taught that swapping is undesirable. When swapping to slow SCSI devices, this may be true, but on zSeries and S/390, there are many other options—and these options can reduce your overall (global) resource requirements. For swapping, the alternative options are:

- ▶ Use Virtual disk as a swap device. The benefit is a much smaller page space requirement, as well as a smaller requirement for real storage.
- ▶ Use RAMdisk as a swap device. The benefit is a smaller requirement for real storage. When sharing storage between many servers, this is important.

8.10 DASD subsystem

For DASD (disk) storage, there are options to share some amount of disk between servers, read only. Using VM's minidisk cache to cache shared data once is significantly more effective than having each Linux cache the same data.

There are currently¹ three different ways to format the disks to be used by Linux.

<code>dasdfmt</code>	The DASD driver in Linux for zSeries and S/390 comes with the <code>dasdfmt</code> utility to format the disks. It formats all tracks on the disk with a fixed block size. There is no support for this particular format in existing S/390 software.
CMS FORMAT	The FORMAT program in CMS also formats the disk with fixed block size, but adds a special eye catcher in R3. This format is recognized by CMS and by CP.
RESERVE	With the CMS RESERVE command, a single big file is created to fill the entire (CMS-formatted) minidisk. The Linux file system is then built into this single big file such that the original CMS formatting of the disk is retained.

There is a small penalty for using the CMS RESERVE format in that some of the blocks on the disk are not available for use by Linux. These blocks are used for CMS housekeeping, as shown in Figure 8-1.

¹ The patches that were made available on June 29, 2001 appear to change several things in this area. We have not yet investigated what the impact of these changes is.

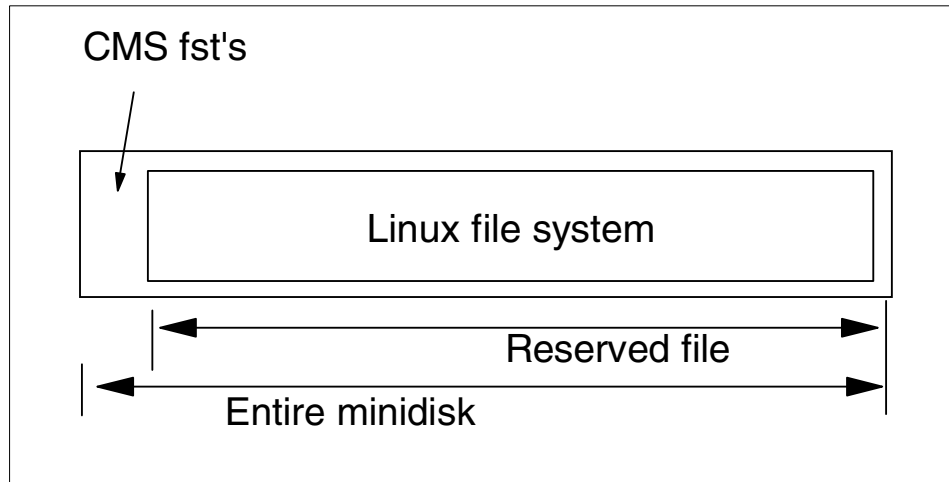


Figure 8-1 Minidisk prepared with RESERVE

However, the advantage of this approach is that the disk can be accessed by a CMS user ID and is clearly identified as in-use to everyone. CMS applications can even read and write the blocks in the file (for example, with the **diskupdate** stage in *CMS Pipelines*). An extra bonus may be the fact that the big file on the disk has a file name and file type which gives you 16 more characters to guide systems management processes (like writing with a marker on CD-R disks that you created).

Linux can also use a disk that was only formatted by CMS. In this case Linux will use all the blocks on the disk such that CMS ACCESS will fail on this disk afterwards. Even when you do not need the ability to access the blocks from CMS, there may still be a good reason to prefer this format over Linux **dasdfmt**. VM directory management products like DirMaint can format the disk before making it available to the user ID.

Tip: If you use IBM RAMAC Virtual Array (RVA), there would be a benefit if you use the “Instant format” function. The “Instant format” is part of the VM SnapShot function such that an instant copy of a formatted (empty) disk is made on the extent to be formatted using the SnapShot feature of RVA. This copy is instantaneous and does not occupy back-end storage in the RVA.

We believe there is no good reason to use the `dasdfmt` command for Linux images on VM, except for the situation where you have a virtual machine running Linux and you forgot to format the disks. Since Linux does not yet tolerate DETACH and LINK of minidisks very well, you’d have no option otherwise but to shut down the Linux system and get back to CMS to format it (but if you do an automatic format with DirMaint, that would not happen anyway).

Note: There used to be a bug in the DASD driver that prevented Linux from booting from a CMS RESERVED minidisk. This resulted in the recommendation to avoid that format when you wanted to boot from disk.

This bug was fixed long ago. You can make a CMS RESERVED mini disk bootable with `si10`. Whether you want to do that on VM, or use a NSS to IPL from, is another matter.

8.10.1 VM Diagnose I/O

The biggest advantage of the CMS RESERVE style of format, however, is that it is the only disk format for which the current Linux for S/390 DASD driver can use VM Diagnose I/O. Diagnose I/O is a high-level protocol that allows the user ID to access blocks on its minidisks with less overhead than pure S/390 channel programs with Start Subchannel (SSCH).

To enable VM diagnose I/O in the DASD driver, you must configure the kernel to enable the “Support for DIAG access to CMS formatted Disks” which is not done in the default SuSE kernel. To enable the option, you first need to disable the “Support for VM minidisk (VM only)” configuration option (also known as the old mdisk driver).

Note: The configuration options in the kernel are slightly confusing in that “CMS formatted disk” really means a disk prepared with the RESERVE command. There is no technical reason why this should be like that. When the kernel is configured without the DIAG option, the DASD driver will use SSCH for the I/O. The VM Diagnose interface does not require the disk to be RESERVED, so as long as it is fixed-block formatted; Diagnose I/O could have been used for both types.

The DASD driver with the May 14, 2001 SuSE distribution appears to be broken. When configured to use the DIAG support, it refused to use the diagnose interface for the disk that was prepared with the CMS RESERVE command. After fixing the dia250() function in dasd_diag.c to return the correct return code, the minidisk was recognized by the driver as such, but Linux then appeared to hang after it started to scan the partition table. Both these problems have been fixed in the 2.2.18 patches from Linux for S/390, but the fix apparently was not ported back to 2.2.16.

Showing the benefits of VM Diagnose I/O

To quantify the effects of VM Mini Disk Cache (MDC) and Diagnose I/O, we did a simple test using the Linux 2.2.18 kernel with the linux-2.2.18-s390 patch. Each Linux image in the test booted with a RAMdisk and then ran a script as shown in Example 8-3. The loop in the script creates a 64 MB file and then reads it four times to allow some of the I/O be satisfied using the cache. The file is large enough to completely flush the buffer cache of the Linux image.

Example 8-3 Sample script for testing diagnose I/O

```
mke2fs /dev/dasda1 -b 4096
mount /dev/dasda1 /mnt

while [ true ]; do
    dd if=/dev/zero of=/mnt/temp bs=1024 count=65536
    cp /mnt/temp /dev/null
    cp /mnt/temp /dev/null
    cp /mnt/temp /dev/null
    cp /mnt/temp /dev/null
    rm /mnt/temp
done
```

We ran a number of Linux images with this script (on a VM system that turned out of be more I/O-constrained than we expected).

Note: When the MDC design was changed to cache full tracks of data rather than just the 4 K formatted CMS minidisks, this caused problems for database applications that do fairly random access to the blocks on their disk, or at least do not follow a track-related reference pattern. The full track cache was then enhanced with the “Record MDC” (sometimes referred to as “Classic MDC”). Since Linux does not have a track-based reference pattern either, it was assumed that Record MDC would make a difference.

Three different ways to format a disk for Linux, and three different styles of MDC, gives nine combinations, but some of these do not need to be measured; see Table 8-1. Only Diagnose I/O is eligible for record MDC. This means that the Linux DASD driver specifying record MDC for the other two styles of formatting disables MDC.

Table 8-1 Impact of MDC on response time

	No MDC	Track MDC	Record MDC
dasdfmt	30.5 s	17.0 s	N/A
FORMAT	30.3 s	17.1 s	N/A
RESERVE	36.3 s	8.8 s	8.9 s

The difference between track and record MDC is very small in this experiment, because the I/O was mainly sequential and involved a relatively small amount of data. With more random access to the data, one should expect record MDC to waste less storage for reading in unwanted data, and thus be more effective.

Table 8-2 Comparison showing the benefits of MDC

Format	MDC	CPU s	I/O	Elapsed times
dasdfmt	no	1.87	3.09	30.5
	track	1.96	2.44	17.0
FORMAT	off	1.90	3.25	30.3
	track	1.41	1.68	17.1
RESERVE	off	2.37	11.3	36.3
	track	1.91	2.65	8.9
	record	1.93	2.28	8.9

The comparison in Table 8-2 on page 165 clearly shows improved response times when using Diagnose I/O combined with MDC. The channel programs used by the DASD driver appear to be “MDC unfriendly” in that they do not exploit MDC very well. We have not been able yet to understand why this is the case. Considering the obvious advantage of Diagnose I/O, it is not very interesting to fix the channel programs used by the DASD driver when running on VM.

8.10.2 DASD MDC measurement

The following ESADSD2 real time screen shot shows a sample measurement over time to help you to understand the effects of track minidisk cache against block minidisk cache. A block-level copy was done on CMS from the LNX013 volume, one at 11:00 with track cache in use, and one at 11:07 with record cache. Using track cache, about 1500 I/Os were issued; using record cache, 18,000 I/Os were issued. Intuitively, this is reasonable with there being 15 blocks per track; using track cache for sequential I/O should greatly reduce the number of physical I/O.

The following shows the activity to the device over time. When evaluating DASD response time, the response time value is usually the most significant; it shows how much time an average I/O takes to the device. When this value is large, then the components of response are evaluated. The components of DASD are evaluated as follows:

Pend time	This is the time for the I/O to be started on the channel; normally less than 1mS.
Disc time	This is the time for the control unit to access the data. This includes rotational delays, seek delays, and processing time inside the control unit. Disc (for disconnect) time is normally less than 2 to 3 mS on cache controllers.
Connect time	This is the time to transfer the data on the channel, normally less than 2 mS for a 4K block of data.
Service time	This is normally the sum of pend time plus disconnect time plus connect time. In some environments, installations may choose to use just disconnect plus connect, but this is not typical.
Queue time	This is the result of many users accessing the same device. If the device is already servicing another user when an I/O is started, the I/O sits in queue. The length of time in queue is queue time. This is the component of response time that, under load, is the most variable and the most serious.

The sample shows that connect time is high when using track cache, and low when using record cache. Record cache moves one 4 K block of data each I/O, and track cache will move up to 15 blocks of data. This example shows the best case for track cache, being a copy of a large file. More typical is random 4 K I/O, in which case reading in a track of cache wastes transfer time and cache space.

When evaluating performance, data has different requirements. If moving data sequentially, then using track cache can be measured.

```
Screen: ESADSD2  ITS0                    ESAMON V3.1  08/07 10:58-11:16
1 of 3  DASD Performance Analysis - Part 1  DEVICE 3ba1          2064 40ECB
```

Dev	Device	%Dev	<SSCH/sec->	<-----Response times (ms)----->								
Time	No.	Serial	Type	Busy	avg	peak	Resp	Serv	Pend	Disc	Conn	
10:59:00	3BA1	LNx013	3390-9	0.0	0.5	0.5	0.8	0.8	0.5	0.0	0.3	
11:00:00	3BA1	LNx013	3390-9	8.6	8.5	8.5	10.2	10.2	1.3	0.0	8.9	<=Track Cache
11:01:00	3BA1	LNx013	3390-9	15.9	17.2	17.2	9.2	9.2	0.2	0.0	9.0	
11:07:00	3BA1	LNx013	3390-9	24.5	197.3	197.3	1.2	1.2	0.2	0.0	1.0	<=RecordCache
11:08:00	3BA1	LNx013	3390-9	12.4	102.7	102.7	1.2	1.2	0.2	0.0	1.0	
11:15:00	3BA1	LNx013	3390-9	0.1	0.6	0.6	1.0	1.0	0.2	0.0	0.8	
11:16:00	3BA1	LNx013	3390-9	0.1	0.2	0.2	3.6	3.6	1.9	0.0	1.7	

8.10.3 High connect time analysis

DASD performance when running Linux guests is very different. After reviewing the following analysis, an I/O trace was performed. However, as we will see, sometimes “an I/O is not an I/O”: Linux using the Start Subchannel I/O driver will chain over 100 CCWs together - with write operations, up to 130 I/O chained together and perceived as one I/O. At 130 times 4 K blocks, that’s over 500 K transmitted per I/O! Sometimes, an I/O is not just an I/O.

The graph in Figure 8-2 on page 168 shows the distribution of read-CCWs over the channel programs². Some 30% of the channel programs have just a single read-CCW and another 30% have 32 reads in them! To phrase it differently: more than 70% of the reads come from a channel program that was reading 128 KB at once.

² The reads are simply “command-chained” in the channel programs, not using suspend and resume operations like CP does for paging I/O.

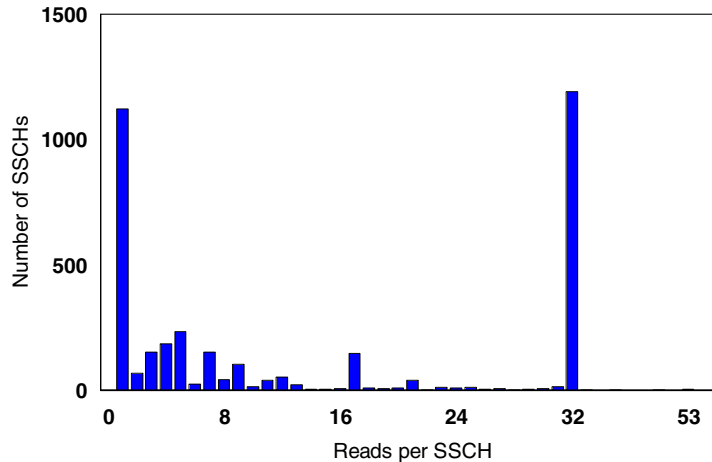


Figure 8-2 Number of read-CCWs in a channel program

The measurements for Figure 8-2 come from a SuSE “default system” install where the contents of the three CDs was copied to a single large minidisk with an ext2 file system. When the CDs were unpacked to the clean disk, very likely files ended up mostly in consecutive blocks in the file system. Because the install process mainly reads large rpm packages from the disk, this makes it easy for Linux to build long channel programs.

While this may not be the typical Linux application we need to run, it is not an artificial situation either because Linux tries very hard to combine multiple disk I/Os in a single operation.

Note: From looking at the read-CCWs combined in a channel program, we get the impression that “track” in the Linux device driver does not match the track on the real device. We did not test this because it would be easier to verify by reading the source code. For efficiency of the control unit cache and MDC, it could be attractive if Linux were more aware of the geometry of the real DASD devices.

Example 8-4 on page 169 looks at a control unit during a test run. The control unit is an RVA. In this case, the operations were 100% read I/O. (High connect times limit the ability of a control unit to service other users, so consideration of the type of work and the appropriate hardware to support the work will be needed.)

In this example, the control unit 3B03, with a range of 256 addresses, is performing a peak of 58.3 I/Os per second, each with an average connect time of 13 milliseconds. Using the 17:21 interval, multiplication of 14 mS times 54 I/O per second shows that a channel is about 75% busy. As channel utilization increases, delays waiting for the channel will occur. You should always ensure that sufficient channel resources are available to meet your workload requirements.

Example 8-4 Control unit during test run

```
Screen: ESADSD2  ITS0                      ESAMON V3.1  08/07 17:09-17:33
1 of 3  DASD Performance Analysis - Part 1  CU 3b03                      2064 40ECB
```

Time	Dev No.	Serial	Device Type	%Dev Busy	<SSCH/sec>		<-----Response times (ms)---->				
					avg	peak	Resp	Serv	Pend	Disc	Conn
17:17:00	3B03	.	3990	0.0	8.2	8.2	0.5	0.5	0.1	0.0	0.3
17:18:00	3B03	.	3990	0.0	10.7	10.7	7.9	7.9	0.2	0.5	7.2
17:19:00	3B03	.	3990	0.3	49.7	49.7	15.7	15.7	0.9	1.0	13.8
17:20:00	3B03	.	3990	0.4	56.8	56.8	16.6	16.6	0.7	1.0	14.9
17:21:00	3B03	.	3990	0.4	54.2	54.2	16.2	16.2	0.6	1.4	14.1
17:22:00	3B03	.	3990	0.4	58.3	58.3	15.7	15.7	0.8	1.9	13.0
17:23:00	3B03	.	3990	0.2	29.9	29.9	19.2	19.2	0.5	1.0	17.7

After looking at the performance analysis of this control unit, the next step is to understand how many paths to the device there are and how they are impacted. Example 8-5 shows there are 4 paths to the devices on this control unit: 41, 4C, 36, and 56.

Example 8-5 DASD configuration display

```
Screen: ESADSD1  ITS0                      ESAMON V3.1  08/07 19:37-19:38
1 of 3  DASD Configuration                  LIMIT 500 DEVICE 3b0 2064 40ECB
```

Dev No.	SysID	Serial	Device Type	Shr	<----Online CHPIDs---->								Ctl Model	Unit (if ded)	UserID	MDisks Linked
					01	02	03	04	05	06	07	08				
3BA0	OCFD	LNx012	3390-9	NO	41	4C	36	56	3990-3E	.	.	0
3BA1	OCFE	LNx013	3390-9	NO	41	4C	36	56	3990-3E	.	.	19
3BA2	OCFF	VMLPG2	3390-9	NO	41	4C	36	56	3990-3E	.	.	0
3BA3	ODO0	LNx014	3390-9	NO	41	4C	36	56	3990-3E	.	.	5
3BA4	ODO1	LNx015	3390-9	NO	41	4C	36	56	3990-3E	.	.	9

Time	Dev No.	Serial	Device Type	%Dev Busy	<SSCH/sec> avg peak		<-----Response times (ms)---->				
							Resp	Serv	Pend	Disc	Conn
20:01:00	3BA3	LNX014	3390-9	0.3	0.1	0.1	24.4	24.4	0.3	22.6	1.4
20:08:00	3BA3	LNX014	3390-9	1.0	0.5	0.5	20.1	20.1	0.2	16.1	3.8
20:09:00	3BA3	LNX014	3390-9	0.0	0.2	0.2	0.4	0.4	0.2	0.0	0.2
20:12:00	3BA3	LNX014	3390-9	6.8	1.5	1.5	46.0	46.0	0.5	3.1	42.3
20:13:00	3BA3	LNX014	3390-9	54.4	18.7	18.7	29.0	29.0	1.4	2.1	25.5
20:14:00	3BA3	LNX014	3390-9	84.2	23.7	23.7	35.6	35.6	1.0	2.7	31.9
20:15:00	3BA3	LNX014	3390-9	63.9	18.0	18.0	35.5	35.5	1.3	1.5	32.7
20:16:00	3BA3	LNX014	3390-9	65.5	18.3	18.3	36.8	35.8	1.2	1.5	33.1
20:17:00	3BA3	LNX014	3390-9	32.7	6.8	6.8	47.8	47.8	1.6	0.9	45.3
20:20:00	3BA3	LNX014	3390-9	2.1	1.3	1.3	15.9	15.9	0.5	6.2	9.1
20:21:00	3BA3	LNX014	3390-9	0.2	0.2	0.2	11.6	11.6	0.1	5.1	6.3

8.10.5 DASD/cache

I/O response time is made up of several components that include disk rotation time, seek time, data transfer times, control unit overheads and queue time. The technologies to deal with these can be faster disks and different forms of cache (processor-based cache or storage controller-based cache). Example 8-8 further analyzes the data from the previous example. Note that cache is active 100% of the time (Pct. Actv Samp), and three of the four samples were in the 10 to 12% read. This validates the statement that this measurement was of write I/O.

The I/O for write hits on the RVA was almost 100% hit, using DASD fast write. DASD fast write is a function provided by the control unit that accepts the data, and terminates the I/O operation from the host perspective. Then the control unit moves the data to disk. This optimization allows more I/O to be started to the device without waiting for data to actually be written to the relatively slow disks. The small number of read I/O were almost always a "hit", meaning satisfied by data in the cache.

Example 8-8 Cache analysis

Screen: ESADSD5 ITS0 ESAMON V3.1 08/07 20:11-20:15
 1 of 3 3990-3 Cache Analysis DEVICE 3ba3 2064 40ECB

Time	Dev No.	Serial	Pct. Actv Samp	<-----per second----->									
				<-----Total----->			<----Read----->						
				I/O	Hits	Hit%	Read%	I/O	Hits	Hit%			
20:12:00	3BA3	LNX014	100	1.4	1.3	90.8	24.1	0.3	0.3	90.5			
20:13:00	3BA3	LNX014	100	18.2	17.6	96.9	12.2	2.2	2.2	100			

```
20:14:00 3BA3 LNX014 100 23.2 22.3 96.1 10.4 2.4 2.4 97.9
20:15:00 3BA3 LNX014 100 17.0 16.5 97.3 12.3 2.1 2.0 97.7
```

8.11 Network performance

Network performance analysis is part of ESALPS. This allows you to determine what nodes are active, and how much network activity is being generated by each one. The screen in Example 8-9 shows the active nodes. The ones that are recognized as running Linux are noted. The TCPIP and HUB6 are VM TCP/IP stacks.

From this screen, moving the cursor to a node and pressing PF2 will show the configuration of that node. Note that in the Name column, a convention was used to include the virtual machine name in the configuration file when setting up the SNMP daemon on Linux. This allows you to look at this configuration data and recognize which virtual machine is running the server.

Example 8-9 Active nodes sample

```
Screen: ESATCPD  ITS0                      ESAMON V3.1  08/09 13:07-13:08
1 of 1  TCP/IP Node list                    NODE *      2064 40ECB
```

Node	IP Address	Name
TCPIP	.	.
HUB6	.	.
IV0123	9.12.0.123	nf3000-1
ITS0237	9.185.246.237	linux7 (Linux)
ITS0232	9.185.246.232	linux2 (Linux)

```
PF1=Help    PF2=ESATCPC  PF3=Quit  PF4=ESATCPT  PF5=ESAHST4  PA1=CP
PF7=Backward PF8=Forward                                PF12=Exit
PA2=Copy
====>
```

Performance data comes in different flavors. The ESATCP2 screen shows the IP layer of data. The four screens showing data from the TCP/IP stacks are ESATCP1, ESATCP2, ESATCP3, and ESATCP4.

Looking at a stack as TCP/UDP (transport layer) on top, that is ESATCP1. The next layer of the stack is the IP layer, shown in ESATCP2. ICMP is shown in ESATCP3, and the hardware/Interface layer is shown in ESATCP4.

Example 8-10 shows the IP layer from the test system. In this instance, there was not a lot of activity. What is shown is the number of datagrams forward and delivered. Note that the “HUB” stack is a VM TCP/IP stack in a virtual machine called HUB6, which is acting as a virtual hub between the Linux servers. It forwards all datagrams, rather than delivering them to the local transport layer and applications.

Many errors in TCP/IP are found in the “Discarded” category. There are many reasons to discard datagrams, such as when they are wrongly addressed, or use an invalid port. We suggest you track errors such as these to detect hackers, and applications that have coding errors.

Example 8-10 IP layer from test system

```

Screen: ESATCP2  ITS0                               ESAMON V3.1  08/09 14:14-14:16
1 of 2  TCPIP Internetwork Layer Data                NODE * LIMIT 500      2064 40ECB

```

Time	Node	<Internet Protocol Datagrams per Second >						<Datagram output>			
		Total	Fwrdd	Dlvrd	Hdr	Addr	Port	Other	Reqst	NoRte	Other
14:16:00	TUX8MSTR	1.55	0.00	0.97	0.00	0.00	0.00	0.00	1.32	0.00	0.00
	ITS0232	0.98	0.00	0.98	0.00	0.00	0.00	0.00	0.98	0.00	0.00
	ITS0237	1.18	0.00	1.18	0.00	0.00	0.00	0.00	1.08	0.00	0.00
	HUB6	6.53	6.53	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TCPIP	3.02	0.00	3.02	0.00	0.00	0.00	0.00	3.18	0.00	0.00
14:15:00	TUX8MSTR	2.60	0.00	1.05	0.00	0.00	0.00	0.00	6.83	0.00	0.00
	ITS0232	0.25	0.00	0.25	0.00	0.00	0.00	0.00	0.22	0.00	0.00
	ITS0237	0.82	0.00	0.82	0.00	0.00	0.00	0.00	0.88	0.00	0.00
	HUB6	3.48	3.48	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	TCPIP	3.65	0.00	3.65	0.00	0.00	0.00	0.00	3.90	0.00	0.00

8.11.1 Network errors

ESALPS also utilizes the TUNETCP macro, which provides network error information. This macro checks for up to 50 different errors on each node being measured. When network slowdown is perceived, executing this macro from either a CMS ID or from a Web browser will show all errors detected using the SNMP data source.

8.12 Server resources

Each server has resource requirements that are needed for both understanding server performance, and for projecting the capacity requirements and growth of the server. The performance characteristics of each server that you will want to monitor are: storage, processor, DASD I/O, network traffic, swap, and probably a few more.

The next step beyond measuring server requirements is to monitor individual applications. When the resource requirements of each application are known, then the growth of each application allows for more accurate capacity planning.

Capacity planning allows you to plan price performance, and to provide Service Level Agreements. In the following sections, we provide suggestions on what application and server data you should monitor and why, in order to ensure optimal performance.

8.12.1 Resources by application

Building a profile of an application allows for accurate capacity planning. Knowing the characteristics of an application also allows you to know when current operational characteristics are out of the normal range, suggesting a problem. Storage and processor requirements by application should be well known.

8.12.2 Resources by server

Each server's resource (Storage, Processor and I/O) requirements should be measured. Detecting variations in server requirements can be done with very little overhead. Determining that a server is outside the normal range of operation early means that problem resolution will take less time.

Example 8-11 shows the processor time and storage profile of the top few users on this test/development system. By reviewing this data from your production workload, you'll have an idea of what servers will top the list (using more CPU than other servers), which servers typically use a lot of storage, and how much. There are additional displays showing I/O data by user ID, as well.

Example 8-11 Processor time and storage profile - top users

```
Screen: ESAUSR2  ITS0                               ESAMON V3.1  08/08 17:29-17:30
1 of 3  User Resource Utilization                   USER *      2064 40ECB
```

Time	UserID	<-----CPU time----->			<----Main Storage (pages)----->					
/Class		Total	Virt	Rat	Total	Actv	Lock	WSSize	Actv	Avg
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

17:30:00 System:	36.088	35.354	1.0	371K	371K	465	412K	412K	11K
TUX60000	31.641	31.528	1.0	23302	23K	0	22303	22K	22K
TUX60001	2.069	2.021	1.0	22159	22K	0	24576	24K	24K
VMLINUX7	0.283	0.255	1.1	31360	31K	30	32768	32K	32K
VMLINUX1	0.223	0.190	1.2	19198	19K	0	24110	24K	24K
ESAWRITE	0.195	0.193	1.0	1357	1357	1	1356	1356	1356
TUX60002	0.190	0.150	1.3	7483	7483	0	7218	7218	7218
VMLINUX4	0.186	0.156	1.2	23275	23K	31	23244	23K	23K
VMLINUX9	0.171	0.141	1.2	6195	6195	0	6076	6076	6076

8.12.3 Resources by accounting

Service Level Agreements include caps on resources used by the customer. If a customer has multiple servers, you will want an easy way to monitor those resources by customer. The monitor can be used as the capture ratio (the amount of resource accounted for divided by the amount of resource used) is normally above 99% when using ESAMON to capture the data. The monitor data contains accounting codes from the CP directory. All of the performance data can then be reported by accounting code.

8.13 Alerts

Automating detection of problems is important when you have hundreds or thousands of servers. Your system will run much better if problems are detected early. ESAMON provides a large set of alerts defined in a file called EXCPN ALERTDEF. The alert function can be started by any user with access to ESAMON with the command **ESAMON ALERT**. Examples of alerts included by default are:

- ▶ Large virtual machine storage sizes and working sets.
- ▶ High virtual machine processor utilization, spool consumption, page rate, and I/O rate.
- ▶ Looping user detection.
- ▶ Idle user detection.
- ▶ Missing user detection to ensure all required users are online. The required users are defined in the file named MISSING USER.
- ▶ Missing DASD detection to ensure all required DASD are online. The file MISSING DASD provides the list of required volume serials.
- ▶ High system page rates.
- ▶ Storage offline.
- ▶ IDASD utilization and rates.
- ▶ Processor utilization.

Alerts can be defined by the installation. Following are some that you may want to add:

- ▶ Excessive resources by accounting number
- ▶ Eligible lists
- ▶ IP Traffic above a specific limit
- ▶ Linux swap rates and virtual disk pages resident
- ▶ Buffer cache excessive – this leads to high storage requirements
- ▶ MDC “not allowed” – fair share for MDC exceeded

8.13.1 Defining and modifying alerts

Each ESAMON alert is defined in a file called EXCPN ALERTDEF, which contains multiple alerts such as the following defined. Example 8-12 defines an alert with code VMCP, which will show users above 5% in blue, users above 10% in blue reverse video, and users above 15% in yellow (the latter also has a warning message sent to the operator).

The text of the alert message is defined with the text operand. The EXCPN ALERTDEF file can (and should) be tailored to meet installation needs. It was designed for a CMS interactive environment supporting thousands of users.

Example 8-12 An alert definition

```
ALERT CPUUTIL VMCP
LEVEL1 5 BLUE
LEVEL2 10 BLUE REV
LEVEL3 15 YELLOW REV ACTION CP MSG OP &USERID RUNNING CRAZY
text User &userid at &cpuutil% of processor
```

Figure 8-3 on page 177 shows a user using excessive CPU; 5 users with excessive working sets; a user that was idle for an extended period; users that should be logged, but are missing; spool utilization greater than 20%; and more. Thresholds were set low for this example; in your case, you'll want to edit the exception definition file to meet your installation's requirements. Each type of alert can be a different color, reverse video, and/or blinking, as a way to emphasize specific problems.

```

Screen: ALERT                                ITS0                                9 Aug 2001 09:29:00
----- Exception Analysis -----
Type Description
VMCP User TUX8MSTR at 98.45% of processor
VMWS Working set for VMLINUX8 98304 pages
VMWS Working set for VMLINUX5 32768 pages
VMWS Working set for VMLINUX7 32768 pages
VMWS Working set for VMLINUX2 32768 pages
VMWS Working set for VMLINUX3 32768 pages
VMVS User VMLINUX8 virtual storage size: 393216K
XMVM User BARTON not logged onto system
XMVM User OPERATOR not logged onto system
XACP Processor utilization at 102.16%
NUSR Logged on users: 34.00
INQU 16.0 users in queue
SPDL Spool space is 26.51% used
DSRV Device 2051.00(VMLU1R) service time: 3.90
DSRV Device 2050.00(VMLPST) service time: 2.40

01          02          03 Quit          04          05          06
07 Backward 08 Forward 09          10          11          12
MÁ b                                     01/001

```

Figure 8-3 ESAMON alert

8.14 Service Level Agreements

You will need to include documentation on service measurement facilities if you provide Service Level Agreements to users or customers. The requirements should include what to report, and how to report it. With today's technology, many customers will want to use a Web-based application to view their data. You should evaluate your performance reporting needs and be prepared for service level reporting.

From a measurement perspective, you will want to monitor resources by accounting number, assuming the Service Level Agreement can be matched to one. Availability of each server should also be monitored.

ESATCP provides alerts to a designated user, which could be to the operator or to a special service machine that has been set up with a PROP function. Each Linux node that will be monitored can have a designated virtual machine user ID that will be alerted for error messages and for a lack of responsiveness.

The terms to include in your Service Level Agreements should define the service being provided, the costs of those service, and escalation procedures, as follows:

- ▶ Resource consumption by consumer, minimum guarantee, and cap
- ▶ Resource reporting mechanism, report details, reporting granularity
- ▶ Alert definition, who is to be alerted, and responses to be taken
- ▶ Availability guarantee, reporting mechanism, and reporting granularity

8.14.1 Availability alerts

There are multiple methods you can use to measure availability. Unfortunately, the most common is to measure the server as being “available” unless a user has called in to complain! When running a service environment, knowing when servers are down is more important. The technologies used can be something like ping, which checks a server for response on a regular basis.

ESATCP provides an alert function using SNMP; each node that you have defined to ESATCP can be monitored for availability at a granularity of your choice. Setting the AVAILTIME parameter to as low as 5 seconds will cause a message to be sent every 5 seconds. When no response for 5 seconds is perceived, an alert will be sent to the designated user. (Note that this measures SNMP responsiveness, and not the applications.)

This is suitable for high level availability where potential for losing connectivity and/or the server itself exists. One application may still have failed without impacting other applications or SNMP. As you develop more requirements for availability, you should have tools that test each application. For an example, refer to the discussion on NetSaint in 13.7.1, “NetSaint” on page 323.

8.14.2 Cost of measuring availability

Whichever method of measuring availability you choose, ensure that the cost is minimal. Stories of disabling monitors and having network traffic drop by a significant percent are not uncommon. The cost of using SNMP for testing availability to a server at each designated interval is two UDP packets on the network, each less than 100 bytes.

8.14.3 Availability reporting

The PDB provided by ESALPS records the amount of time a server is up during each interval. The default interval is 60 seconds, with hourly summaries. The following PDB extract provides an hourly summary of availability.

```
EXTRACT:  
RECTYPE='SU'  
X = 'STOPTIME'  
Y = 'HSTSYS.UPTIME'
```

8.14.4 Measuring service

Each server or customer will have agreed-upon resource access—meaning that the server is guaranteed some amount of processing power and some number of I/O each interval. The interval could be per hour, or even per minute.

The service consumed is provided in the ESALPS Performance Data Base. In addition, the wait states of these servers are provided as well. The monitor samples each server to determine what the server is waiting for (this could be page wait, CPU wait, idle or several other states).

For this kind of analysis, the monitor should use a sampling rate of .1, or 10 times per second. The default setting used by ESALPS is normally a rate of 1 per second. This should be changed during the ESALPS installation.

8.15 Measurement function installation

For measuring Linux using NETSNMP, you will need to install NETSNMP on each server you wish to measure. ESALPS is installed on z/VM.

8.15.1 ESALPS installation

ESALPS is made up of ESATCP, ESAMAP, ESAMON, and ESAWEB. They include the support for NETSNMP data, as well as standard MIB-II data. These products are installed per directions that are provided with the products. Personnel are available for on-site installation assistance as well.

Each node (Linux or otherwise) that you wish to monitor will need to be defined to ESATCP as a node file. This file includes the IP address and the community name.

8.15.2 NETSNMP installation

Installing NETSNMP is documented in 13.6.2, “SNMP installation” on page 306. Configure ESATCP with the password (community name) you have coded, and then restart ESATCP. If SNMP is installed with the virtual machine name in the SYSTEM description, you’ll be able to easily match IP Node with the virtual machine when the virtual machine is operating under VM.

8.16 Measurement methodology

With measurement data provided from any Linux platform, and in fact any UNIX that runs NETSNMP, you can compare application requirements from one platform to another. This will assist you in choosing which applications to run on zSeries and S/390.

8.16.1 Measuring Linux applications

Each application running on a server (virtual or dedicated) will have different resource requirements. Using the host software resource report (ESAHST1) provided by ESALPS, you can determine the resource requirements of each application.

In Example 8-13, sampling everything from SNMP over a period of 3 minutes, the SNMP Daemon used 0.93% of this Linux (on S/390) server. The HTTP servers are using 8.8 MB each, and SNMPPD is using 2468 K.

Example 8-13 LINUX HOST Software Analysis Report

```
Screen: ESAHST1  ITS0                               ESAMON V3.1  08/07 16:46-16:47
1 of 1  LINUX HOST Software Analysis Report        NODE * LIMIT 500      2064 40ECB
```

Time	Node	<--Software Program----->				<CPU Seconds>		CPU	Storage(K)
		Name	ID	Type	Status	Total	Intrval	Pct	Current
16:47:00	ITS0237	httpd	0	0	0	4	0.00	0.00	8836
		snmpd	0	0	0	229	1.67	0.93	2468
		sulogin	0	0	0	0	0.00	0.00	448
		httpd	0	0	0	4	0.00	0.00	8836
		httpd	0	0	0	5	0.00	0.00	8692
		inetd	0	0	0	0	0.00	0.00	576

8.16.2 Measuring Linux server requirements

Each server may run many applications. The sum of the resource requirements of these servers impact the total storage requirements, the amount of swap space required, and the processing power requirements. The ESAHST1 report provides, by application, the processor and storage requirements of each application. These values should be followed over peak periods to show how they would impact other workloads if moved to a zSeries or S/390. Using the abbreviated sample from Example 8-13, this server uses about 30 MB for the identified applications.

In Example 8-14 and Example 8-15, an ESAUCD2 real-time example, you can then measure the total amount of storage that Linux has allocated. The first screen shows the amount of real storage and swap storage available and in use by Linux. When using Linux under z/VM, you'll want to minimize storage requirements.

Alerts can and should be set to show when Linux guests use swap, and when the buffer exceeds some threshold. Using swap indicates the need for more memory, while a large buffer indicates too much storage.

Of the 512 MB defined on this system (516328 K), about 480 MB is accounted for between the shared storage, the buffer storage, the cache storage and the "used" storage. Reducing the size of the ITS0232 machine by 400 K would have no impact on the applications currently in use.

Example 8-14 LINUX UCD Memory Analysis Report (screen 1 of 2)

```
Screen: ESAUCD2  ITS0                      ESAMON V3.1  08/08 14:35-14:55
1 of 2  LINUX UCD Memory Analysis Report    NODE * LIMIT 500    2064 40ECB
```

Time	Node	<--Real Storage-->			<-----SWAP Storage----->			Total	
		Total	Avail	Used	Total	Avail	Used	MIN	Avail
14:55:00	ITS0232	516328	457K	58960	143K	142K	1360	16000	58960
	ITS0237	257000	176K	80524	143K	143K	0	16000	320K
14:54:00	ITS0232	516328	457K	58960	143K	142K	1360	16000	58960
	ITS0237	257000	176K	80524	143K	143K	0	16000	320K
14:53:00	ITS0232	516328	457K	58968	143K	142K	1360	16000	58968
	ITS0237	257000	176K	80524	143K	143K	0	16000	320K
14:52:00	ITS0232	516328	457K	59100	143K	142K	1360	16000	59100
	ITS0237	257000	176K	80524	143K	143K	0	16000	320K

Example 8-15 LINUX UCD Memory Analysis Report (screen 2 of 2)

```
Screen: ESAUCD2  ITS0
2 of 2  LINUX UCD Memory Analysis Report
```

Time	Node	<--Storage in Use-->			Error Message
		Shared	Buffer	Cache	
14:52:00	ITS0232	37788	378112	9432	
	ITS0237	28412	28952	10216	
14:51:00	ITS0232	38380	378112	9432	
	ITS0237	28412	28952	10216	
14:50:00	ITS0232	38380	378112	9432	
	ITS0237	28412	28952	10216	

8.16.3 Measuring VM Virtual Machine

On VM, the resource reports (ESAUSR2, ESAUSR3, ESAUSR4) show the resources of the virtual machine. When analyzed, the data from the ESAHST1 display closely matched the processing requirements reported against the virtual machine by VM on the ESAUSR2 display.

8.17 Tuning guidelines

In the following sections, we provide miscellaneous configuration guidelines that will help you avoid problems and bypass errors that may be not be obvious to installations installing VM for the first time.

8.17.1 Paging and spooling (one extent per Real Device Block)

There should be only *one* page or spool extent per volume. Having multiple extents adds to overhead and may degrade performance. Both spool and page I/O have been optimized with a “never-ending channel program” that allows I/O to bypass the overhead of starting I/O. By having volumes with different types of data, there is an added overhead for each I/O of stopping one I/O and then starting another.

8.17.2 Enterprise Storage Server (ESS)

For VM, you’ll want to define as many Real Device Blocks as possible; only use 3390-9 emulation when absolutely necessary.

The issue with current DASD caching technology is based on a large percent of the I/O being handled by the cache. Under z/VM (at least through V4.1), there is only one real device block per logical disk—and there is a restriction that only one I/O can be started at a time to each logical disk.

Thus, if you have a very large amount of data on a single volume, then when one I/O must retrieve data from the disk, no other I/O can be started even for data that is currently residing in the cache. OS/390 supports large volumes using Parallel Access Volumes (PAV). This allows OS/390 to have multiple I/O to a single logical device by defining multiple paths to a device.

Thus, with PAV, a logical device can be busy, and the data that’s being cached for that device by the ESS is available on other paths. Z/VM does not support this. Without PAV support, you will get optimum performance with smaller and more device addresses.

However, there are two considerations:

1. Large files that are accessed by a single task are not impacted by not having duplicate paths to data. The task must wait for an I/O to complete before starting another one.
2. Large numbers of small servers with random I/O should have the ability to have concurrent I/O. When configuring your storage controller, maximize the number of concurrent I/O by maximizing the number of logical devices.

8.17.3 Virtual machine sizes

Reduce virtual machine sizes as much as possible. Tailoring a server to a specific application and minimizing its storage requirements will mean more storage is available for other servers.

8.17.4 DASD format

The DASD formats and I/O drivers are documented in 8.10, “DASD subsystem” on page 161. You should measure I/O response times, the impact of using MDC, and DASD cache to determine if you are getting the most out of your DASD subsystem.

With `dasdfmt`, MDC only works with track cache, and you must use the ECKD driver that does start subchannels.

The CMS Format Reserved has several advantages, both operationally and from a performance perspective. Either the DIAG driver (Diagnose250) or the ECKD driver may be utilized. When using the DIAG driver, MDC can utilize record level caching. For applications with 4 K blocks read in a random fashion, record level caching will read in just 4 K records instead of full tracks of data. This can reduce the I/O time, channel delays, and storage requirements. Operationally, the files may be read using CMS utilities such as backup.

8.17.5 MDC: fair share considerations (NOMDCFS)

Servers that provide data to other servers should have OPTION NOMDCFS in their directory. MDC is managed with a fair share algorithm that will disallow data to be inserted in to MDC by users that have exceeded their share. For some servers, however, using fair share is inappropriate—so for these servers, put OPTION NOMDCFS in their CP directory entry.

8.17.6 Swap: RAMdisk vs. virtual disk

To reduce Linux storage (by reducing the amount of storage Linux will use for caching data, yet still have enough storage to meet operational requirements), two methods are available:

1. You can define a RAMdisk as part of Linux virtual storage and use this as swap. Linux will then only use this storage for a swap disk.
2. You can use the VM virtual disk facility. By defining a virtual disk and then telling Linux to use it as a swap device, you have a swap device in storage.

When using virtual disks for swap, the important measurement is the amount of storage being used by the swap disk, from the VM perspective. When the amount of storage used for swap becomes large as compared to the virtual machine size, you are likely incurring overhead of moving pages from swap to Linux main storage (which is a cost in processing time).

While no rules of thumb have been developed for this yet, you could assume that controlling the rate of Linux swap activity should be the secondary objective, with the primary objective being to reduce the total storage (virtual machine plus virtual disk) requirements. The ESAVDSK real time display provided by ESAMON shows exactly how much storage is in use for the virtual disk. The ESAVDSK report produced by ESAMAP shows the same information, but normally over a longer period of time.

The following ESAVDSK is an example of a virtual disk used for swap for the duration of a single task. When there was a requirement for more storage, the virtual disk was used. After it was no longer needed, the virtual disk was paged out.

```
Screen: ESAVDSK Velocity Software, Inc.          ESAMON V3.1
<--pages-->  DASD    X-
Resi- Lock-  Page Store
dent   ed   Slots Blks
-----
12:15:01 LINUX001 VDISK$LINUX001$0202$0009    36    0    50    0
12:16:01 LINUX001 VDISK$LINUX001$0202$0009    36    0    50    0
12:17:01 LINUX001 VDISK$LINUX001$0202$0009   173    0    50    0
12:18:01 LINUX001 VDISK$LINUX001$0202$0009   293    0    35    0
12:19:01 LINUX001 VDISK$LINUX001$0202$0009   293    0    35    0
12:39:01 LINUX001 VDISK$LINUX001$0202$0009   259    0    35    0
12:40:01 LINUX001 VDISK$LINUX001$0202$0009   259    0    35    0
12:41:01 LINUX001 VDISK$LINUX001$0202$0009   207    0    86    0
12:42:01 LINUX001 VDISK$LINUX001$0202$0009   207    0    86    0
12:43:01 LINUX001 VDISK$LINUX001$0202$0009    13    0   280    0
12:44:01 LINUX001 VDISK$LINUX001$0202$0009    13    0   280    0
12:45:01 LINUX001 VDISK$LINUX001$0202$0009    13    0   280    0
```

8.17.7 Timer tick kernel changes

The currently available Linux for zSeries and S/390 distributions implement the timer functions in the kernel using a 10 mS interrupt that increments the “jiffies” global variable. Work is being done on an implementation that is more suitable to Linux running as guests under VM.

When planning on operating many Linux servers under z/VM, you should plan on implementing this “no more jiffies” patch as soon as it is available. This reduces the processor requirements of supporting many idle guests. Without this patch, you can expect 0.2 to 0.3% of a processor (G5) to be used by each idle server.

8.17.8 Kernel storage sharing

Reducing storage requirements by sharing storage is used by CMS for the CMS operating system, by programs, and for data. This technology is slowly being developed for Linux. We suggest you watch for developments and implement them when possible.



Part 2

Practical considerations

In this part of the book we provide explicit examples of the work we did during this residency. For a theoretical discussion of the concepts behind installing and managing z/VM and Linux for zSeries and S/390 systems, see Part 1, “Theoretical considerations” on page 1.



VM configuration

Running a large number of Linux virtual machines on a VM system is a serious challenge, with many configuration and tuning issues to deal with in order to make the system work properly. Some of these issues are related to VM, and some are Linux issues. In this chapter, we focus on the VM aspects of the configuration.

9.1 General VM configuration issues

Using an average VM system straight out of the box, you will probably not be able to run a large number of Linux images efficiently, so you should be aware of the following general guidelines.

9.1.1 Allocate sufficient paging space

Virtual machines running Linux tend to be rather large (compared to average CMS users), so be prepared to have sufficient paging DASD set up to accommodate all the virtual storage that you give out to your users. Unlike with CMS users, over time virtual machines running Linux will use all the storage you give them.

The recommendation currently is to have twice as much paging space on DASD than the sum of your total virtual storage, in order to let CP do block paging. This means that to run 10 Linux virtual machines of 512 MB each in an LPAR with 2 GB of main storage, you need to have 14 GB worth of paging DASD. Failure to do so may cause your VM system to take a PGT004 abend before you notice paging space filling up.

With this amount of space you will not be tempted to mix it with other data, but it should be clear you do not mix paging space with other data on the same volumes. If you plan to use saved systems to IPL Linux images as outlined in this chapter, you also need to seriously evaluate spooling capacity, because NSS files reside on the spool volumes. See 8.17.1, “Paging and spooling (one extent per Real Device Block)” on page 182 to learn why this is important.

9.2 Things to do for new Linux images

When creating new Linux images, the following tasks need to be performed.

9.2.1 Create a central registry

In order to manage a large number of Linux images on a VM system, you need to maintain a central registry of the Linux images and use standard processes to create the user IDs. Several of the utility services on VM would need to do the correct things to these images, based on the registration.

For the purpose of this discussion, it does not really matter whether this “central registry” is your white board in the office or a few files on a shared disk somewhere in the system. (However, ease of access and the options for automation make it attractive to use online files to hold the configuration data.)

9.2.2 Create the user ID in the CP directory

The CP directory entry only defines the virtual machine, but many of the parameters (such as storage) that define the virtual machine can be specified or overruled at startup time. Also, many aspects of the virtual machine, such as performance settings, cannot yet be specified in the CP directory and therefore need to be handled by automation software. Other aspects, such as IUCV authorization, do need to be defined in the CP directory. The point is that if, in your installation, many things need to be arranged outside the CP directory, it is questionable whether Linux images need to be bound to specific VM user IDs.

Compare this to an installation with discrete servers where these servers do not have a hard disk and boot from the LAN. The tables in bootp will define what image needs to run on what hardware. This makes it easy to deal with hardware failures and so on.

You could adopt a similar strategy in VM and use the parameters that *must* be in the CP directory to help you determine what user ID to use.

9.2.3 Allocate the minidisks

When you create a Linux image, you need to allocate and initialize the disks for the Linux image. You need to plan this carefully so that you can place the minidisks on the proper volumes.

9.2.4 Define the IP configuration

Each Linux image will need TCP/IP connectivity. Apart from getting an IP address for the system and having it registered in the Domain Name System (DNS), the Linux image will also need a network connection. Depending on the network architecture used, this means you must define the IP address in the OSA configuration or prepare the point-to-point connection in the VM TCP/IP or Linux hub.

9.2.5 Install and configure the Linux system

When the user ID is defined in VM and TCP/IP characteristics are known, the Linux system can be installed and configured. Note that, with a number of similar Linux images running on the same VM system, there are more efficient ways to get another working Linux image than just run the entire installation process; these issues are addressed in Chapter 10, “Cloning Linux images” on page 209.

9.2.6 Register the user ID with automation processes

This registration will ensure that the user ID is automatically started, stopped, monitored, and so on.

9.2.7 Register the user ID so backups can be made

This registration should also ensure recovery will be done when something “breaks”.

9.3 Using VM TCP/IP as the virtual router

The VM TCP/IP stack can be used as a virtual router. With the current restrictions of the CTC and IUCV drivers in Linux (see 17.1, “Ability to reconfigure CTC and IUCV” on page 408), there are advantages in using the VM TCP/IP stack.

For example, new interfaces and point-to-point links can be defined and activated on the VM TCP/IP stack without stopping and starting it. However, the process to do this—as well as the syntax of the configuration files—can be slightly intimidating. The following section demonstrates how to use the OBEYFILE command to dynamically create the connections. The topology of the network is shown in Figure 9-1 as a guide to the IP addresses given in the examples.

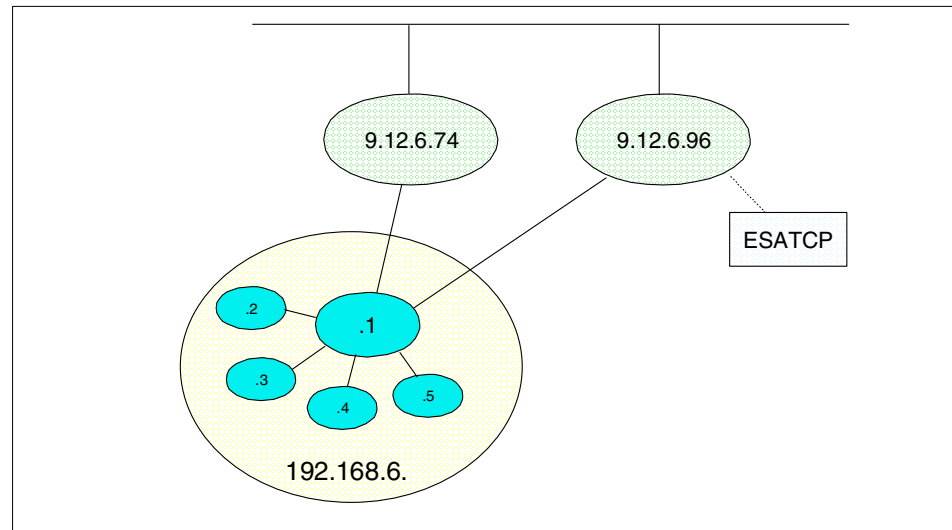


Figure 9-1 Topology of the network

The scenario described here is for IUCV connections. The same process can be done for CTC connections, with a few minor differences. The first difference is in the DEVICE and LINK statements (see *TCP/IP Planning and Customization*, SC24-5981, for the details). The other difference is that virtual CTC devices need to be defined in the VM TCP/IP stack virtual machine and COUPLED to the corresponding other virtual CTC device.

9.3.1 Dynamic definitions and the PROFILE TCPIP file

When the VM TCP/IP stack is started, it reads the configuration from its profile. When you change the configuration of a running VM TCP/IP stack with the OBEYFILE command, you must also update the profile to make sure these changes will be picked up when the VM TCP/IP stack is restarted.

The OBEYFILE command is unlike other VM commands. When a user issues the OBEYFILE command, that causes the VM TCP/IP stack link to the minidisk of that user. It then reads the file specified on the OBEYFILE command from that disk to pick up the configuration statements. This means the VM TCP/IP stack must be authorized to link to the user's disk (either by the ESM or through a valid read-password).

Note: If no ESM is used on VM, the read-password must be supplied as an option for the OBEYFILE command. It is annoying that the OBEYFILE command parses the parameters and options different from normal CMS commands. The read-password (specified as an option after the "(" character) is only recognized when filetype and filemode of the file are specified.

A "naive" implementation of OBEYFILE can cause problems with ad hoc changes to the TCP/IP configuration. If the program is invoked by an automated process, most of these problems can be avoided.

Note: Now that the QUERY MDISK command with the USER option is available even for general users, we believe it should be considered a bug that the VM TCP/IP stack does not use this for the OBEYFILE command. With the current implementation, it is quite possible for a VM TCP/IP stack to link an incorrect disk and activate the wrong configuration statements.

Because you also need to update the TCP/IP profile, you want that disk to be linked R/O by the VM TCP/IP stack. This way the user that issues the OBEYFILE commands can also update the profile.

9.3.2 Creating the device and link

The point-to-point connection in VM TCP/IP requires both a device and a link to be defined in the PROFILE TCPIP file:

```
▶▶—DEVICE—device_name—IUCV—0 0—other_virtual_machine—priority—▶▶
```

Figure 9-2 The DEVICE statement for an IUCV connection

```
▶▶—LINK—link_name—IUCV—link_number—device_name—▶▶
```

Figure 9-3 The LINK statement for an IUCV connection

While having device names and link names promotes flexibility, for point-to-point connections it can become quite cumbersome.

Fortunately TCP/IP doesn't care if we use the same identifier both for the device name and for the link name¹. And since the VM TCP/IP stack has just a single point-to-point connection to each Linux image, we might as well use the user ID of the Linux virtual machine as device name and link name. Since the parsing rules for the TCP/IP profile do not require the DEVICE and LINK statement to be on different lines, we can put both on the same line. The syntax may look a bit redundant, but this makes it much easier to automate things.

The definition in the TCP/IP profile for a point-to-point link to our point-to-point connections can now be defined as shown in Example 9-1.

Example 9-1 The DEVICE and LINK statements for our virtual router

```
device tcpip iucv 0 0 tcpip a link tcpip iucv 0 tcpip  
device vmlinux6 iucv 0 0 vmlinux6 a link vmlinux6 iucv 0 vmlinux6  
device tux80000 iucv 0 0 tux80000 a link tux80000 iucv 0 tux80000  
device tux80001 iucv 0 0 tux80001 a link tux80001 iucv 0 tux80001  
device tux80002 iucv 0 0 tux80002 a link tux80002 iucv 0 tux80002  
device tux80003 iucv 0 0 tux80003 a link tux80003 iucv 0 tux80003
```

¹ There is no concern that future versions of VM TCP/IP will be more strict in this aspect.

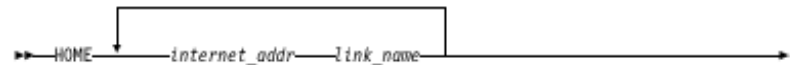
The first connection is the “uplink” to the VM TCP/IP stack which we used to give ESATCP (part of the Linux Performance Suite) access to the Linux images. The second one provides a direct connection to another Linux image on the same VM system to allow clients in that Linux image (e.g. telnet) to access the Linux images. This connection would not be possible via the real network. See 4.4.3, “Using the OSA with Linux” on page 95 for more details.

If you compare this with the syntax definition for the DEVICE and LINK statement, you can deduce which occurrence of the user ID is what. (This may seem confusing, but will be worth the effort because you won’t need to remember whether the START statement requires the link name or the device name.)

9.3.3 Defining the home address for the interface

The IP address of the VM TCP/IP stack side of the point-to-point connection must be defined in the HOME statement.

Figure 9-4 The syntax of the HOME statement



For point-to-point connections, the same IP address can be specified for each link. Since the IP address does not have to be in the same subnet as the other side of the connection (we specify a subnet mask of 255.255.255.255), we can even use the uplink IP address for it.

The virtual router in our example does not have its own real network interface either, but uses an IUCV connection to the main VM TCP/IP stack. The first part of the HOME statement in our profile is shown in Example 9-2.

Example 9-2 The HOME statement in the TCP/IP profile

```
home
  192.168.6.1  tcpip
  192.168.6.1  vmlinux6
  192.168.6.1  tux80000
  192.168.6.1  tux80001
  192.168.6.1  tux80002
  192.168.6.1  tux80003
```

Unfortunately, VM TCP/IP requires all interfaces to be listed in a single OBEYFILE operation when a new interface is added.

9.3.4 Defining the routing information

The GATEWAY statement is used to specify the IP address of the stack at the other side of the point-to-point connection.

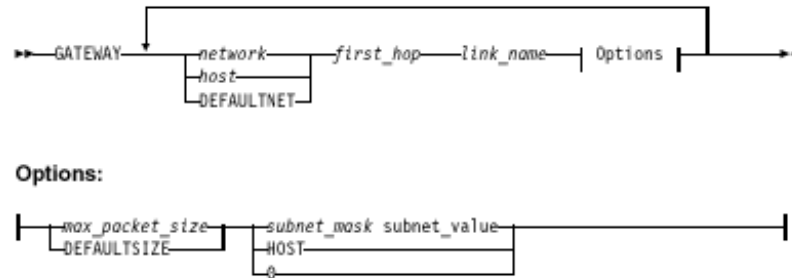


Figure 9-5 The syntax of the GATEWAY statement

As shown in Figure 9-5, each of the connections must be listed in the GATEWAY statement. Example 9-3 shows the routing statements for our virtual router. The tcpip link is the uplink connection to the VM TCP/IP stack.

Example 9-3 The GATEWAY statement in the profile

GATEWAY						
; (IP) Network	First	Link	Max. Packet	Subnet	Subnet	
; Address	Hop	Name	Size (MTU)	Mask	Value	
; -----	-----	-----	-----	-----	-----	-----
9.12.6.96	=	tcpip	8188	host		
9.12.6.74	=	vmlinux6	8188	host		
defaultnet	9.12.6.74	vmlinux6	8188	0		
192.168.6.2	=	tux80000	8188	host		
192.168.6.3	=	tux80001	8188	host		
192.168.6.4	=	tux80002	8188	host		
192.168.6.5	=	tux80003	8188	host		

The defaultnet route is to the Linux machine vmlinux6 (instead of to the VM TCP/IP stack, as you might expect); this is because the 192.168 addresses only exist on this VM system. Trying to let one of these images connect to another host in the 9. network wouldn't work because there is no route back into this VM system.

Just as with the HOME statement, VM TCP/IP requires the entire GATEWAY statement to be supplied in the OBEYFILE command when something must be changed or added to it.

9.3.5 Starting the connection

To start the point-to-point connection, the START command is given for the device.



```
▶—START—device_name—▶
```

Figure 9-6 The syntax of the START command

A single START command is used to start the connection for the device. The TCP/IP profile must have START commands for all devices that you want to start. If you are defining the devices and links up front, it may be an advantage to postpone the START until the Linux guest is ready to connect. Otherwise you would waste time (and console log lines) with the VM TCP/IP stack retrying that connection.

Retry and reconnect

The connection between two TCP/IP stacks via IUCV consists of two IUCV paths. Each side will “connect” its sending connection to the other party. For a working IP connection, both paths need to be up. When the sending IUCV path is “severed” by the other end, the stack will respond by “severing” the other path as well.

There is difference between the way the VM TCP/IP stack and the Linux netiucv driver handle the connection setup. When an existing connection is stopped from the VM side, Linux will notice that and bring the link down as well (and show that in the system log).

However, when the link is started again from the VM side, Linux will not pick up the connection request. Even an **ifconfig iucv0 up** command will not do the trick because the network layer assumes the connection is still up. To make Linux take action, you need to execute **ifconfig iucv0 down** followed by an **ifconfig iucv0 up** command. Unfortunately, that removes the default route you may have set up using that connection. We believe the Linux netiucv driver should be changed to listen for a connection attempt again after the path was severed.

If the VM TCP/IP stack is retrying the link, it will attempt to connect to the other side every 30 seconds. Between those attempts, the VM TCP/IP stack is “listening” all the time and will respond immediately when Linux tries to establish a connection.

For a CTC connection, there also is a retry every 30 seconds when the connection is down. The process is slightly different in that the VM TCP/IP stack does not have a read outstanding all the time when the connection is waiting to be retried. This means that when a new Linux image is brought up, it may need to wait up to 30 seconds before the VM TCP/IP stack is able to take notice of the attempt. This waiting period may seem trivial, but it really is not when you talk about bringing up a new image in 90 seconds.

Attention: We believe there are problems with the way the Linux CTC driver handles the reconnect. More than once we found Linux in a tight loop, trying to restore a broken CTC connection. We did not have the time to dig into these problems. Casual debugging of this with the CP TRACE command is difficult with the 10 mS timer tick going on.

The 2.4.5 version of the kernel appears to be changed in using a shorter period to wait for a response from the other side. That period could be too short, since we were frequently unable to restart a failing connection.

9.3.6 Putting all the pieces together

A simple program was written to add the connection for a Linux image to the VM TCP/IP stack. It updates the PROFILE TCPIP and issues the OBEYFILE command to activate the new connection on the running VM TCP/IP stack. To make the file easier to parse, we added a few special comments in the file to mark the place where items should be inserted. Our TCPIP PROFILE is shown in Example 9-4.

A generic parsing routing for the configuration file is complicated, but we do not need this flexibility when the new entries are added through an automated process anyway.

Example 9-4 The PROFILE TCPIP for our virtual router

```
tinydatabufferpoolsize      20
monitorrecords
timestamp prefix

device tcpip      iucv 0 0 tcpip      a link tcpip      iucv 0 tcpip
device vmlinux6  iucv 0 0 vmlinux6 a link vmlinux6  iucv 0 vmlinux6
device tux8mstr  iucv 0 0 tux8mstr a link tux8mstr  iucv 0 tux8mstr
device tux80000 iucv 0 0 tux80000 a link tux80000 iucv 0 tux80000
device tux80001 iucv 0 0 tux80001 a link tux80001 iucv 0 tux80001
; =device

home
  192.168.6.1 tcpip
```

```

192.168.6.1 vmlinux6
192.168.6.1 tux8mstr
192.168.6.1 tux80000
192.168.6.1 tux80001
; =home

GATEWAY
; (IP) Network First      Link      Max. Packet Subnet      Subnet
; Address      Hop        Name      Size (MTU) Mask        Value
; -----
9.12.6.96     =          tcpip     8188        host
9.12.6.74     =          vmlinux6 8188        host
defaultnet   9.12.6.74 vmlinux6 8188        0
192.168.6.254 =          tux8mstr 8188        host

192.168.6.3  =          tux80000 8188        host
192.168.6.4  =          tux80001 8188        host
; =gateway

start tcpip
start vmlinux6
start tux8mstr

start tux80000
start tux80001
; =start

```

With the restrictions we've put on the layout of the configuration file, the program to do the updates can be really simple, as shown in Example 9-5. The program adds the proper entries to a copy of the configuration file on the A-disk. It then builds a temporary file with the DEVICE statement, the START statement and the HOME and GATEWAY sections, and issues an OBEYFILE command against that file. When OBEYFILE gives a return code 0, the configuration file is replaced by the new one, and the temporary files are erased.

Example 9-5 Simple program to add a connection to the configuration

```

/* ADDTCPIP EXEC      Add a Linux guest to TCP/IP      */

parse arg userid nr .

'PIPE state PROFILE TCPIP * | spec w1.3 1 | var config'
workfile = 'TEMP TCPIP A'

'PIPE <' config '|locate /'userid/' | count lines | var cnt'
if cnt > 0 then
do

```

```

        say 'Link for' userid 'probably already present'
        return 1
    end

    devs = 'device' userid 'iucv 0 0' userid 'a link' userid 'iucv 0' userid
    home = ' 192.168.6.1 ' userid
    gate = ' 192.168.6.'left(nr,3) '=          ' userid '8188          host'
    strt = 'start' userid

'PIPE (end \)',
  '\ <' config,
  '| x1: strtolabel /; =device/',
  '| i: fanin',
  '| >' workfile,
  '\ var devs | i:',
  '\ x1:',
  '| x2: strtolabel /; =home/      | i:',
  '\ var home | i:',
  '\ x2:',
  '| x3: strtolabel /; =gateway/ | i:',
  '\ var gate | i:',
  '\ x3:',
  '| x4: strtolabel /; =start/    | i:',
  '\ var strt | i:',
  '\ x4: | i:'

'PIPE (end \)',
  '\ <' workfile,
  '| strfrlabel /; =device/',          /* Take HOME and GATEWAY sect */
  '| strtolabel /; =gateway/',
  '| preface var strt',              /* and the START */
  '| preface var devs',              /* Add the DEVICE statement */
  '| >' userid 'TCPIP A'

'OBEYFILE' userid 'TCPIP A (READ'
if rc = 0 then
  do
    'COPYFILE' workfile config '(OLDD REPL'
    'ERASE' workfile
    'ERASE' userid 'TCPIP A'
  end
return rc

```

Deleting a link is also possible. Because of the simple layout of the configuration file, we can do it with a program as shown in Example 9-6 on page 201.

Unfortunately, we cannot delete the device statement once it is defined to the VM TCP/IP stack, but we can at least stop it.

Example 9-6 Deleting a connection from the configuration

```
/* DELTCPIP EXEC      Delete a Linux image from TCP/IP configuration */

parse arg userid .
'PIPE var userid | xlate lower | var userid'

'PIPE state PROFILE TCPIP * | spec w1.3 1 | var config'
workfile = 'TEMP TCPIP A'

'PIPE <' config '|' nlocate /'userid'/ | >' workfile

'PIPE (end \)',
  '\ <' workfile,
  '| strfrlabel /; =device/',          /* Take HOME and GATEWAY sect */
  '| strtolabel /; =gateway/',
  '| literal STOP' userid,            /* and the START */
  '| >' userid 'TCPIP A'

'OBEYFILE' userid 'TCPIP A (READ'
if rc = 0 then
  do
    'COPYFILE' workfile config '(OLDD REPL'
    'ERASE' workfile
    'ERASE' userid 'TCPIP A'
  end
return rc
```

9.3.7 Define and couple the CTC devices

In addition to what is shown in previous sections for point-to-point connections over IUCV, a virtual CTC needs to be defined and coupled. Both DEFINE and COUPLE are CP commands that can be issued through the NETSTAT CP interface. The COUPLE command can be issued from either side of the connection. The `hcp` command (from the `cpint` package) can be used to issue the COUPLE commands from the Linux side.

To have the virtual CTCs defined at startup of the VM TCP/IP stack, you can define them in the user directory or include them in the SYSTEM DTCPARMS file with the `:vctc` tag. When the CTC is defined through the DTCPARMS file, the COUPLE command is also issued (provided the Linux machine is already started up). The PROFILE EXEC of your Linux guest should also be prepared to define the virtual CTC (if not done through the user directory) and try to couple to the VM TCP/IP stack, in case Linux is started after the VM TCP/IP stack.

If you run a large number of Linux images this way, you probably should have a control file read by the PROFILE EXEC of your Linux guest to define the proper virtual CTCs. One option would be to read and parse the DTCPARMS control file of TCP/IP so that you have a single point to register the CTCs.

9.4 Using DirMaint to create Linux virtual machines

On a VM system with more than a few user IDs, it is impractical to maintain the user directory manually. Editing the directory by hand is cumbersome and error-prone. Security is another consideration, because if you do not run an External Security Manager for VM, logon passwords and minidisk passwords are also maintained in the user directory—and they are visible in clear text to the person maintaining the directory (and anyone who looks over his or her shoulder).

DirMaint is the IBM program product to manage your VM user directory. Its complete name is “5748-XE4 Directory Maintenance VM/ESA”.

If your installation is using DirMaint, you must use DirMaint to maintain the user directory; you do not have the option in that case of managing your Linux user IDs by hand-editing the USER DIRECT (and trying to do so could cause serious problems). The same holds true when the VM installation is using a directory management product from a solution developer like VM:Secure. You cannot realistically run different directory management products on the same VM system.

Using DirMaint is not the only way to manage your user directory; it can also be managed by a CMS application as a simple flat file in CMS and brought online with the DIRECTXA command. If you only have very typical standard Linux images, then maintaining this flat file could be automated fairly easily. However, when you run z/VM to host a large number of Linux images, you are likely to end up with a lot of user IDs that are non-standard or different.

9.4.1 Why to avoid GET and REPLACE

Even when DirMaint is in control of your user directory, you can still mostly maintain it yourself if you want to use GET and REPLACE commands to update user entries in DirMaint. However, we recommend that you learn the DirMaint commands to do incremental directory updates, because this is less error-prone than editing the directory entries by hand. Understanding how to use these DirMaint commands will also give you an idea of how the process can be automated.

An additional bonus for avoiding the use of GET and REPLACE is that the DirMaint console log will give you a full report of each change that was made to the directory.

9.4.2 Keeping the user directory manageable

Both directory profiles and prototype files can be used to simplify management of the user directory.

Directory profiles

The CP user directory supports profiles to be used in the definition for a user in the directory. The INCLUDE directory statement in the user entry identifies the profile to be used for that user. Obviously DirMaint also supports the use of these directory profiles. The profile itself is managed by DirMaint similar to user IDs, so you can use normal DirMaint commands to add statements to profiles.

The profile contains the directory statements that are identical for the user IDs (e.g. link to common disks, IUCV statements). You can create different profiles for the different groups of users that you maintain.

Example 9-7 Sample directory profile

```
PROFILE TUX6PROF
ACCOUNT TUX6
IPL 1B0
IUCV TUX6MSTR
MACHINE XA
CONSOLE 0009 3215 T
SPOOL 000C 2540 READER *
SPOOL 000D 2540 PUNCH A
SPOOL 000E 1403 A
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR
LINK TUX6MSTR 0191 0191 RR
LINK TUX6MSTR 01A1 01A1 RR
LINK TUX6MSTR 01A0 02A0 RR
LINK VMLINUX6 01B0 01B0 RR
```

The include profile shown in Example 9-7 is used for all Linux images in a penguin colony to allow each of the user IDs to set up an IUCV connection to the “leader of the colony” (which is TUX6MSTR, in this case) and have links to some of the disks of the leader.

Prototype files

A *prototype* is like a skeleton for a user entry in the directory. The prototype is only used by the DIRM ADD command with the LIKE option to create a new user entry according to the prototype.

Example 9-8 Using DIRM ADD with LIKE to add a user ID

```
dirm add tux80004 like tux8 pw icefloes
DVHXT1191I Your ADD request has been sent for processing.
Ready; T=0.04/0.04 14:58:40
DVHREQ2288I Your ADD request for TUX80004 at * has been accepted.
DVHBIU3425I The source for directory entry TUX80004 has been updated.
DVHBIU3425I The next ONLINE will take place as scheduled.
DVHSCU3541I Work unit 08145842 has been built and queued for processing.
DVHSHN3541I Processing work unit 08145842 as RVDHEIJ from VMLINUX,
DVHSHN3541I notifying RVDHEIJ at VMLINUX, request 75.1 for TUX80004
DVHSHN3541I sysaffin *; to: AMDISK 01A0 3390 AUTOG 100 LNX MR BLKSIZE
DVHSHN3541I 4096
DVHRLA3891I Your DMVCTL request has been relayed for processing.
DVHDRC3428I Changes made to directory entry DATAMOVE have just been
DVHDRC3428I placed online.
DVHDRC3428I Changes made to directory entry TUX80004 have just been
DVHDRC3428I placed online.
DVHRLA3891I Your DMVCTL request has been relayed for processing.
DVHREQ2289I Your ADD request for TUX80004 at * has completed; with RC
DVHREQ2289I = 0.

DVHSHN3541I Processing work unit 08145842 as RVDHEIJ from VMLINUX,
DVHSHN3541I notifying RVDHEIJ at VMLINUX, request 75.1 for TUX80004
DVHSHN3541I sysaffin *; to: AMDISK 01A0 3390 AUTOG 100 LNX MR BLKSIZE
DVHSHN3541I 4096
DVHDRC3428I Changes made to directory entry DATAMOVE have just been
DVHDRC3428I placed online.
DVHDRC3428I Changes made to directory entry TUX80004 have just been
DVHDRC3428I placed online.
DVHSHN3430I AMDISK operation for TUX80004 address 01A0 has finished
DVHSHN3430I (WUCF 08145842).
```

The DIRM ADD command in Example 9-8 shows how user ID TUX80004 was added using the prototype TUX8. The USER statement in the prototype file would cause each user ID created from the prototype to have the same password.

To avoid that, DirMaint requires the password for the user ID to be specified on the ADD command. The asynchronous messages from this command also refer to the DATAMOVE user ID. Because the AMDISK statement in the prototype specifies that disk should be formatted, it is handed to the DATAMOVE user ID first to have it formatted. A so-called “workunit” is created to format the disk. The second part of the output shows the messages related to the workunit being created and completed.

Changes that you apply to a prototype file afterward will not affect the user entries created using that prototype. The prototype is used for directory statements that cannot be placed in the profile (like USER and INCLUDE) and MDISK statements that are similar but not identical (although the number of minidisks and their sizes will be the same, the starting cylinder will be different for each Linux image). Automatic allocation (using the AUTOV or AUTOG option) can be used to define the minidisks.

Example 9-9 Prototype file

```

USER TUX8 ICEFLOE 48M 256M G
    INCLUDE TUX8PROF
    MDISK 01A0 3390 AUTOG 100 LNX MR

```

When a new user ID is added using this prototype file, DirMaint will automatically allocate 100 cylinders in group LNX for the minidisk.

9.4.3 Rotating allocation

Automatic allocation in DirMaint is not only very useful when combined with skeletons, but also for ad hoc adding of minidisks to existing user IDs. When multiple volumes are grouped together, DirMaint can do rotating allocation on these volumes and distribute the minidisks over the volumes in that group.

Example 9-10 Fragment of EXTENT CONTROL for rotating allocation

```

:REGIONS.
  *RegionId  VolSer  RegStart  RegEnd  Type
  VMLU1R    VMLU1R    001       3338 3390-03  3390 1  USER
  LNX013    LNX013     1         10016 3390-09  3390 1  USER
  LNX014    LNX014     1         10016 3390-09  3390 1  USER
  LNX015    LNX015     1         10016 3390-09  3390 1  USER
:END.
:GROUPS.
  *GroupName RegionList
  ANY  VMLU1R
  LNX (ALLOCATE ROTATING)
  LNX  LNX013 LNX014 LNX015
:END.

```

The (ALLOCATE ROTATING) entry in the LNX group in the example causes DirMaint to distribute the minidisks over these three volumes. When you use automatic allocation in skeleton files, the disks for the new user would otherwise probably be allocated on the same volume. This could be bad for I/O performance if you have a high degree of multiprogramming in the Linux image, especially when you use large disks like (emulated) 3390-9.

9.4.4 Implement exit for minidisk copy

The DVHDXP exit in DirMaint would need to be implemented to allow DirMaint to copy disks that contain a Linux file system. While increasing or decreasing the size of the minidisk with an ext2 file system is fairly hard to do, copying an ext2 file system from one minidisk to the other is not difficult (especially when CMS RESERVED mini disks are being used, the DFSMS COPY command can do this).

DirMaint is prepared to use DFSMS when installed. This should allow DATAMOVE at least to copy a CMS RESERVED minidisk to another extent of the same size and device type. (Unfortunately, we were unable to verify this because DFSMS was not installed on the VM system we used.) We believe DFSMS COPY does not currently copy the IPL records of the disk, so if your Linux images need to IPL from disk (rather than NSS), that would be something to watch out for.

Implementing a routine for DVHDXP using the program shown in Example 10-4 on page 214 is fairly straightforward using the DirMaint documentation. DirMaint development is aware of these restrictions, so it is possible they will be removed in some future version.

9.5 Using an alternate boot volume

Linux for S/390 does not use 1110 as the Intel implementation does². Being unable to get back to your previous kernel can make testing a new kernel somewhat risky. However, you can mount a small minidisk over /boot and use that as the IPL device; Example 9-11 on page 206 shows how to prepare one of the two IPL minidisks.

Example 9-11 Preparing a separate boot disk

```
# cat /proc/dasd/devices
0205(ECKD) at (94:0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0204(ECKD) at (94:4) is dasdb:active at blocksize: 4096, 36000 blocks, 140 MB
0201(ECKD) at (94:8) is dasdc:active at blocksize: 4096, 468000 blocks, 1828 MB
```

² The patches for the 2.4.5 kernel from June 2001 change various things in this area. This may even include an option to select from different kernels.

```

0202(ECKD) at (94:12) is  dasdd:active  at blocksize: 4096, 468000 blocks, 1828 MB
206A(ECKD) at (94:16) is  dasde:active  at blocksize: 4096, 18000 blocks, 70 MB
206B(ECKD) at (94:20) is  dasdf:active  at blocksize: 4096, 18000 blocks, 70 MB
# mke2fs /dev/dasdf1 -b 4096
mke2fs 1.19, 13-Jul-2000 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
18016 inodes, 17997 blocks
899 blocks (5.00%) reserved for the super user
First data block=0
1 block group
32768 blocks per group, 32768 fragments per group
18016 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
# mount /dev/dasdf1 /mnt
# cd /boot
# tar cf - . | tar xf - -C /mnt
# cd /
# umount /mnt
# mount /dev/dasdf1 /boot
# df

```

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
/dev/dasdb1	139452	83080	49176	63%	/
/dev/dasdc1	1842540	752128	996816	43%	/usr
/dev/dasdf1	69720	1812	64312	3%	/boot

As you can see, the IPL minidisk could have been much smaller than the 100 cylinders we happened to have. This new boot disk can now be updated with the new kernel, and **sil**o makes the disk bootable.

Example 9-12 Making the alternate disk bootable

```

# cd /boot
# ls
. System.map-2.2.16 image image.config ipldump.boot
iplfba.boot parmfile parmfile.orig
.. boot.map image.autoconf.h image.version.h ipleckd.boot
lost+found parmfile.map
# cp /usr/src/linux/arch/s390/boot/image .
# cp /usr/src/linux/System.map System.map-2.2.16
# cat parmfile
dasd=0205,0204,0201,202,206a,206b root=/dev/dasdb1 noinitrd
# df .

```

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
/dev/dasdf1	69720	1820	64304	3%	/boot

```

# silo -d /dev/dasdf
o->ipldevice set to /dev/dasdf

```

```
Testlevel is set to 0
IPL device is: '/dev/dasdf'
bootsector is: '/boot/ipleckd.boot'...ok...
bootmap is set to: '/boot/boot.map'...ok...
Kernel image is: '/boot/image'...ok...
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
ix 0: offset: 00026a count: 0c address: 0x00000000
ix 1: offset: 000277 count: 80 address: 0x0000c000
ix 2: offset: 0002f7 count: 80 address: 0x0008c000
ix 3: offset: 000377 count: 7c address: 0x0010c000
ix 4: offset: 0003ff count: 02 address: 0x00188000
ix 5: offset: 000401 count: 01 address: 0x00008000
Bootmap is in block no: 0x00000402
#
```

To complete the process, you should add the new /boot to /etc/fstab so that the correct System.map will be picked up by **klogd**, but you may want to skip that step the first time you try your new kernel. You can create several alternate boot disks this way, if necessary.

Note: Another approach is described in the IBM Redbook *Linux for S/390 and zSeries: Distributions*, SG24-6264. It renames the /boot directory first, and then creates a new mount point /boot and mounts the new boot disk on that.

Although that approach works as well, we believe the method we detailed in this book is somewhat more elegant.



Cloning Linux images

In order to run a large number of Linux images on VM, you need to have those images in the first place. There are different ways to create these images, as discussed in this chapter.

10.1 Overview

One way of creating Linux images is to install each image from scratch, as if you were going with the set of CDs from one server to the next. If you run a significant number of Linux images, and the images are very similar, you may want options that will help you avoid repeating the entire install process for each Linux image. Procedures for this type of repeated installation for discrete servers have been developed for internal use in many organizations.

Often these procedures are tailored to meet the specific requirements of the organization. An initiative for a generic approach is the IBM Open Source project “Linux Utility for cluster Installation” (LUI). The source code for this project is available at the developerWorks Web site. Unfortunately, during this residency we did not have time to see if LUI is applicable to Linux for S/390 as well. Some Linux distributions offer their own approach for repeated installs (such as the YaST shortcuts in SuSE and the `mkkickstart` utility in Red Hat).

When you run Linux images on VM, you have additional options. With VM, you can easily access the disks of the Linux images even when the Linux image is not running. You can use VM utilities to copy the disks from one image over to another image, and then apply the modifications that are needed to use this copy in the next Linux image. This process of making the copy and applying the changes is often referred to as “cloning” Linux images.

10.2 Installing Linux images the easy way

The most straightforward way to install Linux images on a VM system is to create the VM user ID with sufficient minidisks, and install the Linux from some distribution on these minidisks. If you have only a few Linux images on the same VM system, you could simply repeat the process for each of them. This process is well documented in the redbook *IBM @server Linux for zSeries and S/390: Distributions*, SG24-6264.

However, if you have more than just a few Linux images, manually doing the entire installation over and over again is tedious and inefficient, so you may want a few shortcuts. Because the SuSE distribution uses a full-screen menu-based installer, it does not lend itself very well to automated repeated installs. The Turbolinux and Red Hat distribution is easier to automate if you are planning to do a fresh install for each Linux image.

10.2.1 Providing fast access to the install medium

If you have only a small number of Linux images, you may want to do a fresh install for each of them. You may also want to do this for educational or recreational purposes. Rather than getting all the rpm packages via TCP/IP for each install, you can put a copy of the ISO images on a large minidisk and have the Linux image use a R/O link to that disk.

The commands in Example 10-1 show how to mount the ISO images of the installation CDs into your file system before you start the installation program. With the ISO images mounted like this, you can point YaST to the `/install/cd1` directory to find the installation material.

Example 10-1 Mount the ISO images via the loop device

```
# mkdir /install
# cd /install
# mkdir cd1 cd2 cd3
# mount -o loop,ro suse-us-s390-cd1.iso cd1
# mount -o loop,ro suse-us-s390-cd2.iso cd2
# mount -o loop,ro suse-us-s390-cd3.iso cd3
# cd /
# yast
...

```

You can simplify this process even further by unpacking the ISO images to a single large ext2 file system (thereby avoiding the use of the loop device during the install). This way you can also add your own packages to the installation medium, or upgrade some of the packages in it.

Example 10-2 Copy the contents of the ISO images to a file system

```
# mount /dev/dasde1 /mnt
# cd /install
# cd cd1 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cd cd2 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cd cd3 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cp /boot/ipleckd.boot .
# cd /
# umount /install
# mount /dev/dasde1 /boot
# ln suse/images/tapeipl.kr image
# ln suse/images/parmfile parmfile
# ln suse/images/initrd initrd
# silo -d /dev/dasde -r initrd

```

The last few commands in Example 10-2 create links in the root directory of the disk to each of the starter files (kernel, parameter file and initrd image) and a copy of `ipleckd.boot`. The `silo` command makes the disk bootable for installation.

To install Linux in a new user ID, you can link to this big disk that contains the unpacked three ISO images and IPL it. Installation from such a disk is very fast.

Note: There is a bug in the SuSE install program in that it misses an FBA device (for example, a VDISK) when building the parameter file after installing the packages. After you exit YaST, you must mount the new root device again and edit the parameter file to include your swap device at the beginning of the parameters for the DASD driver.

In order to be able to install further packages after the reboot, you'll want to include the disk with the installation files as well—do that at the end of the parameters for the DASD driver. Finally, run `silo` again.

10.3 Building a quick start disk

By using the “hidden” option of the `silo` command, you can make a disk with a kernel, parameter file, and initrd to be used as the startup system or rescue system (to avoid punching kernel and RAMdisk images). As with the recipe in 9.5, “Using an alternate boot volume” on page 206, you need to mount the disk to be prepared on `/boot` in order for `silo` to work. The difference is in the `-r` option of `silo`, which allows you to specify the initrd image.

Example 10-3 Using silo to make a quick start disk

```
vmlinux6:/boot # silo -d /dev/dasdh -r initrd.gz
o->ipldevice set to /dev/dasdh
o->ramdisk set to initrd.gz
Testlevel is set to 0
IPL device is: '/dev/dasdh'
bootsector is: '/boot/ipleckd.boot'...ok...
bootmap is set to: '/boot/boot.map'...ok...
Kernel image is: '/boot/image'...ok...
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
initialramdisk is: 'initrd.gz'...ok...
ix 0: offset: 00016d count: 0c address: 0x00000000
ix 1: offset: 00017a count: 80 address: 0x0000c000
ix 2: offset: 0001fa count: 80 address: 0x0008c000
ix 3: offset: 00027a count: 6d address: 0x0010c000
ix 4: offset: 0002e7 count: 01 address: 0x00008000
ix 5: offset: 001d10 count: 0c address: 0x00800000
ix 6: offset: 001d1d count: 80 address: 0x0080c000
```



```
..[snip].  
ix 22: offset: 002520 count: 80 address: 0x0100c000  
ix 23: offset: 0025a0 count: 32 address: 0x0108c000  
Bootmap is in block no: 0x000002e8
```

You will also notice that the list of blocks in this case is much longer than when no RAMdisk image is specified. If you now **umount** the disk and issue a **sync** command, the disk can be linked from other user IDs to quickly boot a system with RAMdisk.

10.4 Copying disks instead of doing a full install

With virtual machines on VM, you have options other than simply repeating the entire installation process for each image. For example, you can use the DASD Dump and Restore (DDR) utility after a full installation of Linux to copy the contents of the minidisks of one virtual machine to another set of minidisks. This is very practical if you want to keep a copy of your entire Linux system when you are going to do significant changes to your Linux system. That other copy can be started by linking the minidisk at the correct address (this is necessary because the kernel parameters refer to the minidisk address).

Important: Make sure you *shut down* your Linux image before you copy the contents of the minidisks. This is obvious when you run the DDR utility in the same virtual machine that was running Linux. However, when you run the DDR utility in another user ID with a R/O link to your Linux disks, you need to keep this in mind.

This is because when the disks of a *running* Linux image are copied, the resulting file system will at best be “dirty” (not cleanly unmounted, so an **fsck** is required at boot time). Since Linux buffers data in memory before writing it out to disk, your copy could be incomplete and inconsistent if the system is actively doing work. The same applies to normal backups of these disks with your VM backup product.

Instead of switching between IPLs of one of these copies in the same virtual machine, you can also take the copy and IPL it in another virtual machine. However, there are a few complications when you do this. For example, the two copies of Linux you get this way are completely identical, including any configuration changes done by the installation program (e.g. IP address, device addresses, etc).

Therefore, in order to use this approach for generating new Linux images, you need to find the correct point in the installation process to copy the disk. You also need to understand what configuration changes must be repeated for the new image, and how to do these changes.

10.4.1 Copying disks for cloning images

Many will tell you that using DDR is the best way to copy minidisks. While it is probably the cheapest, Table 10-1 shows that it is not always the fastest way to do the job.

Table 10-1 Copying a 250-cylinder minidisk with ext2 file system

	Elapsed time (s)	I/Os	MDC hits	CPU time (s)
DDR copy (first)	30	3014	0	0.148
DDR copy (second)	29	3014	0	0.125
CMS copy (first)	34	13759	41	0.880
CMS copy (second)	19	13759	1166	0.800

The alternative copy on CMS was done with a little pipeline that is shown in Example 10-4 (which we could do because the disk was prepared with RESERVE before we ran `mke2fs` against it). Because this ext2 file system happened to be filled only for 70% in this case, there are still a lot of blocks filled with binary zero. Those blocks do not need to be copied to the target disk. The second run is even faster because MDC is offering a lot of help. DDR does not appear to benefit from MDC.

Example 10-4 Using CMS Pipelines to copy the payload of a reserved disk

```

PIPE
  < tux01a0 tux6mstr j                /* Read from the input disk */
  | spec number 1.10 ri 1-* n          /* Insert block number in record */
  | not verify 11-* x00                /* Drop when just '00'X chars */
  | fileupdate tux01a0 tux60000 k      /* to write them to disk */

```

```

PIPE
  mdiskblk number j 1.2                /* Read boot record from disk */
  | mdiskblk write k 1.2                /* and write to target disk */

```

The question remains as to how much you could benefit from MDC in a real-life situation when cloning Linux images, and whether you can spare the CPU cycles. When the file system is more scattered over the disk, or when the file system contains more data than the arbitrary value of 70% used in our test, then the benefit of the CMS-based copy will soon disappear.

10.4.2 Determine the point to copy the disks

All current distributions use a process where you boot Linux with a RAMdisk to populate the disks, and then boot from disk. After booting from disk, SuSE for example runs **SuSEconfig** again and rewrites all kind of files (including the keys for SSH, if you installed that). Therefore, the most practical point at which to copy the disks during the install process is just before this reboot from disk.

Note: This means you must keep this original install system, without booting it, if you want to clone more images from it. If necessary, you can get back into YaST again and add more packages if you have to, and subsequent copies will then also have those packages installed.

10.4.3 Locating the files to be customized for each cloned image

For each cloned Linux image, you now have to customize the files that are prepared during the first phase of the install (with IP address, host name, root password, etc).

There is no easy recipe for this step; determining which files need to be changed will depend on the distribution and the packages installed. This step will have to be done for each new release of the distribution, although the differences between two releases of the same distribution will probably be very small.

Various tools in Linux are available to assist in determining which files to change. (For example, we ran a **find -mtime 0 -type f** command, which showed that only 98 out of 82711 files in our SuSE install were changed). If you look at the list of files, you see there are also a few with “old” and “bak” in their names that you can ignore for this process.

For each file found by this **find** command, you can use **grep** to search for files that contain the IP address or host name, as shown:

```
# grep 192.168.0.2 `find -mtime 0 -type f`
```

We expected the changes for a SuSE distribution to be rather trivial¹ because **SuSEconfig** takes many of the configuration parameters from `/etc/rc.config`. After the reboot, **SuSEconfig** rewrites most of the files that were identified as changed during the installation.

¹ This turned out to be more complicated than expected because many things happen outside SuSEconfig (for example, `/etc/route.conf`). The process outlined in the following sections can handle all these changes anyway, so there is less reason to depend on SuSEconfig for some of the work.

When the cloned image uses the same type of network interface and the same DNS and gateway, you do not have to change these when cloning an image. And when you select a unique-enough hostname and IP address for the first install, you may be able to just use a few **sed** commands, as shown in Example 10-5.

Example 10-5 Update the rc.config with IP address and hostname

```
cat /mnt/etc/rc.config \  
    | sed s/192.168.0.2/192.168.0.4/ \  
    | sed s/tux6mstr/tux60002/ > /root/rc.config  
cp /root/rc.config /mnt/etc/rc.config
```

To change the initial root password, you can **chroot** to the file system to be prepared and invoke the **passwd** command. (For SuSE, however, this is somewhat useless since it will prompt for a new password anyway at the first reboot.)

Using diff to find the changes

You can also take a more “brute force” approach to identifying the differences between cloned images by using the **diff** command. The **diff** command compares two files and lists the differences between the files.

For this approach, you perform an identical install according to the book on two different Linux images. Keep these installations the same for the configuration items that will be the same for all your cloned images (e.g. the DNS or domain name), and then deliberately introduce a difference for the items that must be customized for each cloned image (e.g. IP address and hostname).

After completing the installation up to the point where the first boot from disk is required, link both file systems in a single Linux image (the examples here show this running with a RAMdisk system because that’s an easier way to configure the DASD driver).

Example 10-6 Mounting the two file systems to be compared

```
# insmod dasd dasd=200,1a0,1a1,1b0,1b1,0cd  
Using /lib/modules/2.2.16/block/dasd.o  
# cd /mnt  
# mkdir a b  
# mount /dev/dasdb1 a -r  
# mount /dev/dasdc1 a/usr -r  
# mount /dev/dasdd1 b -r  
# mount /dev/dasde1 b/usr -r  
# df
```

Filesystem	1k-blocks	Used	Available	Use%	Mounted on
/dev/ram2	25901	22377	3524	86%	/
/dev/dasdb1	174116	44968	120160	27%	/mnt/a
/dev/dasdc1	1415848	692296	651632	52%	/mnt/a/usr

/dev/dasdd1	69628	44968	21068	68%	/mnt/b
/dev/dasde1	1415848	692296	651632	52%	/mnt/b/usr

The fragment in Example 10-6 shows how to load the DASD driver and mount the file systems under the root file system. Because both installs were done on a root file system with two disks, all these disks are needed here.

Unfortunately, these file systems contain absolute path names in the symbolic links, so you can not simply run `diff` against both file trees. Instead, change the absolute path names in the symbolic links to relative ones (there are 69 of those in a “default” install of SuSE). You can identify the problems with the following commands:

```
# find -type l|xargs ls -l $1|cut -b 56-|grep "[^ ]* -> /"
```

Alternatively, if you don’t want to change the symbolic links in the file system, you can first run `md5sum` against *each* file in *both* file systems (to compute a checksum for each file), and then use `diff` on the files that have different checksums. Obviously, computing the checksum for each file with `md5sum` is not a cheap process, but on the configuration we used, it completed in two minutes for a “default install” of SuSE. You need to do this once during the preparation.

The scenario in Example 10-7 shows we use the `chroot` command before running `diff`. With `chroot`, you run another command (the shell, in this case) with the specified directory as the root of the file system. The absolute symlinks in the file system now match the file system. The `exit` command terminates the shell and returns back to the configuration before the `chroot` command.

Example 10-7 Computing the checksum for each file in the file system

```
# df
Filesystem          1k-blocks    Used Available Use% Mounted on
/dev/ram2            25901        22381    3520  86% /
/dev/dasdb1         174116        48884   116244  30% /mnt/a
/dev/dasdc1         1415848      692296   651632  52% /mnt/a/usr
/dev/dasdd1          69628        44968    21068  68% /mnt/b
/dev/dasde1         1415848      692296   651632  52% /mnt/b/usr
# mount b -o remount,rw
# chroot b
# find / -type f | xargs md5sum $1 > /sums
# sort -k 2 /sums > sumsort
# exit
```

Because **find** did not always return the files in the same order, it was necessary to sort the list on file name. We could have done the sort by piping the output of **xargs** into it, but the temporary files created by **sort** caused a lot of noise in the output.

Using the files with checksums created in the previous steps, we can now identify the files that are different by using **diff** to compare these two files, as follows.

```
diff -u a/sumsort b/sumssort |grep ^\|- |cut -b 36- \  
| a/usr/bin/gawk '{print "diff -u a"$0" b"$0}' ' \  
| grep / |/bin/sh
```

Note: We “cheated” a bit with the **gawk** command. Because the RAMdisk system did not have **gawk** installed, we used the **gawk** executable from the mounted file system that we were comparing. However, a better solution would be to install **gawk** in the RAMdisk system—or to run with a full Linux system.

A portion of the output of running **diff** against these pairs of files is shown in Example 10-8. The **patch** command can take a file like this as its input, to apply the changes to the files in the file system of a cloned Linux image.

Example 10-8 Fragment of the output of diff comparing the files

```
--- a/etc/rc.config      Wed Jul 25 08:42:09 2001  
+++ b/etc/rc.config      Wed Jul 25 08:48:21 2001  
@@ -132,7 +132,7 @@  
#  
# IP Adresses  
#  
-IPADDR_0="192.168.0.2"  
+IPADDR_0="192.168.0.3"  
  IPADDR_1=""  
  IPADDR_2=""  
  IPADDR_3=""
```

To use a process like this for customizing cloned images, you have to generate the proper modified version of this output and feed it to the **patch** command. After cleaning up the output of **diff** by hand and removing the patches to files that will be replaced by **SuSEconfig** anyway, a diff file of only 83 lines remained, updating 8 different files in the system.

In the diff file, we replaced things like the host name and IP address by strings that are easy to identify (like `:hostname:`), and a simple script with a few **sed** commands can now produce the patch file.

Example 10-9 Script to create the specific patch for a cloned image

```
#!/bin/sh
if [ -z $5 ]; then
    echo "Need hostname IP-address gateway-userid gateway-ip gateway-mtu"
    exit
fi
cat generic.diff \
| sed "s/:hostname:/$1/ " \
| sed "s/:cloneip:/$2/ " \
| sed "s/:gatewayid:/$3/ " \
| sed "s/:gatewayip:/$4/ " \
| sed "s/:gatewaymtu:/$5/" \
```

Figure 10-1 illustrates how the first and second install are compared with **diff**. The output is then transformed into a generic patch. The shell script in Example 10-1 creates a specific patch out of the generic patch. This specific patch is used as the input for the patch program.

Note: If you generate a patch as outlined, you'll also touch files that are marked as "do not modify" because they are maintained by **SuSEconfig**.

In theory, you could simply modify `rc.config` and then run **SuSEconfig** to generate the other files. However, in our case, we decided to patch the output files anyway because that avoids running **SuSEconfig** and rebooting the system.

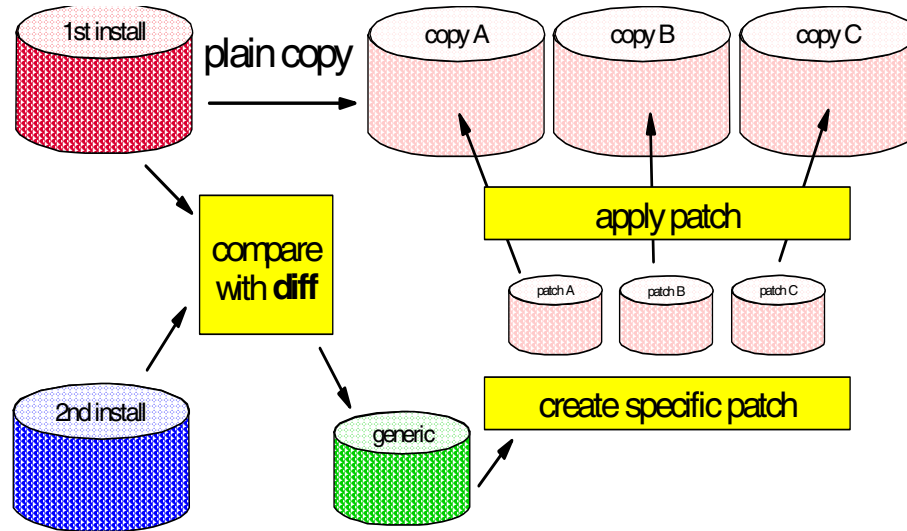


Figure 10-1 Using diff and patch to configure the cloned images

10.4.4 Reducing the number of changes required

Carefully designing your virtual server architecture can significantly reduce the number of changes to be made to each cloned image. If you have a number of different classes of Linux images, then it may be easier to have one separate master image for each category. This will result in fewer changes to be made to a cloned image—which is not only easier to manage, but also *essential* if you want to share disk space among Linux images.

Many useful results can be realized through proper design of VM user IDs for the Linux images. If you keep the differences in the CP directory and the PROFILE EXEC, you can avoid changes to the cloned images. This means, for example, that you keep the same device address for your virtual CTCs on each Linux image, even though they connect to a different address on your virtual hub or VM TCP/IP stack. This way the entry in `/etc/modules.conf` can be the same for each image.

A significant simplification would be if the cloned images could make use of Dynamic Host Configuration Protocol (DHCP) instead of having hardcoded IP addresses in the configuration files. However, DHCP requires an Ethernet or Token Ring interface, so it doesn't work with point-to-point connections like IUCV and CTC interfaces.

Note that running DHCP does not have to mean you hand out different IP addresses for your systems (which is what DHCP normally is used for). You can manually bind IP addresses to the physical interfaces and keep the same IP address each time the server starts. DHCP would only be the easy way to configure the IP stack of each system.

You could even do dynamic IP addresses for your servers. This would be an option when combined with Dynamic Domain Name Service (DDNS). In that case, each system would register through DDNS, for example, as the next system offering a particular service. Round-robin DNS would then cause the traffic to be spread over multiple systems.

10.4.5 When to apply the configuration changes

There are at least two different ways to apply the changes to the copied disks for the new Linux image.

- ▶ Store a small script in the boot process to apply these changes the first time the cloned image is executed. Instead of using a custom script with hardcoded values, you can write a generic script that obtains the IP address and hostname through some other mechanism. The `hcp` command from the `cpint` package can be used to obtain the user ID of the virtual machine running the image.
- ▶ Mount the file system into an existing running Linux image and run a script to apply the changes. This is a more flexible choice, because you can do different repair actions that may be necessary (like installing additional packages). Unfortunately, it is currently difficult to access the file system of a Linux image that is not running (also known as a “dead penguin”), or one that does not have a network connection.

Which method will work best in your situation depends on various aspects of your installation (such as network topology, naming convention, etc).

10.5 Sharing code among images

The process outlined in 10.4, “Copying disks instead of doing a full install” on page 213 can simplify the install process, but it does not yet exploit z/VM facilities for sharing resources. For those who are used to working with VM, it appears obvious that you want to share the common code between your Linux images. For a “default install” of SuSE, about 85% of the disk space is used for the `/usr` directory of the file system. Most packages have their binaries installed in `/usr`, so it seems obvious that this also should be shared.

The Linux Standard Base (LSB) defines the file system hierarchy such that /usr can be shared among images (outside the S/390 world, over NFS). Unfortunately, it also defines that application code should be installed in /opt, which makes sharing /usr less attractive.

The file system can be split over two separate minidisks after installation, but it's much easier to let YaST do that job; during installation, you define one disk to hold the /usr directory and YaST will copy all those files to that minidisk. This process is apparently so obvious that many people have built Linux images that had a R/O link to a common minidisk holding the /usr subtree of the file system. This results in a significant reduction in disk space requirements for running a large number of Linux images.

However, as with many cheap solutions, this one also comes with a few drawbacks.

- ▶ Once a Linux image is using the shared /usr disk, it is no longer possible to make changes to the contents of the disk.

In CMS, to share R/O minidisks, people have developed tricks to deal with the “one writer - many readers” situation, but for Linux this does not work because every Linux image will buffer parts of that disk in its buffer cache.
- ▶ Portions of applications live outside the /usr directory, for example in /bin and /sbin. When those other portions are part of the private disk space of the Linux image, it will be difficult to maintain consistency when upgrading packages. This means there is no easy way to upgrade the systems afterward.
- ▶ Many applications and users need to write into portions of /usr for their function. An example of this is the package manager rpm that keeps its database (the software inventory) in /usr, as well.
- ▶ Many applications do not separate code and data (like WebSphere, writing the log file in the bin directory by default), which makes it very difficult to share code.
- ▶ The recent standards define /opt to hold packages that are not part of the code system. This would make sharing /usr less effective.

We believe a realistic answer to this could be a new device driver as described in “Shadowed disk support” on page 413. This code is not available yet.

Despite these drawbacks, this simple way of sharing the disk may be attractive for special situations (for example, “disposable penguins” only needed for a limited period). If an application has a specific requirement to write in an otherwise R/O directory, a private writable directory can be provided through a symlink to another part of the file system, or by “overmounting” with another block device (via the loop driver, for example).

10.6 Breeding a colony of penguins

For a demonstration of the cloning process, we had to use a configuration with /usr on a separate R/O linked minidisk, despite the drawbacks illustrated in 10.5, “Sharing code among images” on page 221. Trying the cloning process with private disk space for each cloned penguin would have been too expensive, both in disk space and in time to create the images.

Unfortunately, YaST runs SuSEconfig after the first reboot, which turns out to write into the /usr directory. This means we need to finish the install process completely before we have a file system that can be copied and used by others. Since other installations may have similar restrictions, we decided to take a more generic approach and complete the install before copying the disks.

The cloning process is demonstrated with each of the steps invoked by hand. This does not mean the process could not be automated, but it’s probably easier to follow this way than with a single program that does all.

10.6.1 Images used in the cloning process

The images used in the cloning process will have a few minidisks:

01A0	A 100-cylinder private disk to hold the root file system
02A0	R/O link to the original root file system for copying
01A1	A 2000-cylinder disk linked R/O by all images
00CD	R/O link to a starter disk with initrd

Each of the cloned images will have an IUCV connection to a single common VM TCP/IP stack.

10.6.2 Create a patch file for cloning

We create a patch for the cloning process as described in “Using diff to find the changes” on page 216. Doing this after finishing the install is not much different from doing it earlier in the process.

Obviously, **diff** will also find a lot of differences in log files and other things that you do not want to end up in the patch, so you can get those out of the process as soon as possible with, for example, a few **grep** commands before sorting the list of filenames.

To create the two different disks to compare, we copied the 01A0 disk of the install system to a new minidisk and then ran YaST in the installation system to change the IP address and hostname.

Although the chosen network setup resulted in identical gateway addresses and default routes for each of the cloned images, we decided to put that in the patch anyway to have a more generic patch.

Example 10-10 Patch generated for the demo setup

```

--- a/etc/HOSTNAME      Wed Aug  8 02:54:04 2001
+++ b/etc/HOSTNAME      Wed Aug  8 03:04:17 2001
@@ -1,1 @@
-tux8mstr
+:hostname:
--- a/etc/hosts Wed Aug  8 02:54:04 2001
+++ b/etc/hosts Wed Aug  8 03:04:17 2001
@@ -21,4 +21,4 @@
  ff02::2          ipv6-allrouters
  ff02::3          ipv6-allhosts

-192.168.6.254      tux8mstr.hub6.itso.ibm.com      tux8mstr
+:myip:            :hostname:.hub6.itso.ibm.com :hostname:
--- a/etc/rc.config  Wed Aug  8 02:54:02 2001
+++ b/etc/rc.config  Wed Aug  8 03:04:16 2001
@@ -132,7 +132,7 @@
#
# IP Adresses
#
-IPADDR_0="192.168.6.254"
+IPADDR_0=":myip:"
  IPADDR_1=""
  IPADDR_2=""
  IPADDR_3=""
@@ -151,7 +151,7 @@
# sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
-IFCONFIG_0="192.168.6.254 pointopoint 192.168.6.1 mtu 8188 up"
+IFCONFIG_0=":myip: pointopoint :gwip: mtu :gwmtu: up"
  IFCONFIG_1=""
  IFCONFIG_2=""
  IFCONFIG_3=""
@@ -207,7 +207,7 @@
# (e.g. "riemann.suse.de" or "hugo.linux.de")
# don't forget to also edit /etc/hosts for your system
#
-FQHOSTNAME="tux8mstr.hub6.itso.ibm.com"
+FQHOSTNAME=":hostname:.hub6.itso.ibm.com"

#
# Shall SuSEconfig maintain /etc/resolv.conf (needed for DNS) ?
--- a/etc/route.conf  Thu Aug  9 16:21:37 2001
+++ b/etc/route.conf  Thu Aug  9 16:19:37 2001

```

```

@@ -34,4 +34,4 @@
# 192.168.0.1          0.0.0.0          255.255.255.255      ipp0
# default            192.168.0.1
192.168.6.1          0.0.0.0          255.255.255.255      iucv0
-default            192.168.6.1
+default            :gwip:

```

The resulting patch is shown in Example 10-10. It turns out to be very small. It touches only four files in the system (those marked with +++ characters). This patch was stored in the root directory of the installation system so that it would be available in each cloned image as well.

Copy the root file system and IPL starter system

The root file system is copied using the COPYDISK program shown in Example 10-11. Exploiting the fact that this root file system is filled for 70%, this turned out to be the fastest solution.

Example 10-11 The COPYDISK program

```

/* Copy a reserved disk to a formatted disk */
signal on error
arg cuu1 cuu2 .

'ACCESS' cuu1 'J'
'PIPE COMMAND LISTFILE * * J | var infile'
parse var infile fn .
'ACCESS' cuu2 'K'
queue '1'; 'RESERVE' fn userid() 'K'
'PIPE <' infile,
    '| spec number 1.10 ri 1-* n',
    '| not verify 11-* x00',
    '| fileupdate' fn userid() 'K'
'PIPE mdiskblk number J 1.2 | mdiskblk write K'
return rc

```

After the root file system is copied, the RAMdisk system is started.

Example 10-12 Copying the root file system and IPL with RAMdisk

copydisk 2a0 1a0

```

DMSACC724I 2A0 replaces J (2A0)
DMSACP723I J (2A0) R/O
DMSACC724I 1A0 replaces K (1A0)
DMSRSV603R RESERVE will erase all files on disk K(1A0). Do you wish to
continue? Enter 1 (YES) or 0 (NO).
1
DMSRSV733I Reserving disk K

```

Ready; T=0.54/0.80 20:43:15

ipl 0cd clear

Linux version 2.2.16 (root@ikr_tape.suse.de) (gcc version 2.95.2 19991024
(release)) #1 SMP Tue May 1 11:47:13 GMT 2001
Command line is: ro ramdisk_size=32768 root=/dev/ram0 ro

Load the DASD driver and mount the disks

Now the DASD driver is loaded and the new disks are mounted.

Example 10-13 Accessing the disks from the RAMdisk system

```
# insmod dasd dasd=1a0,1a1
Using /lib/modules/2.2.16/block/dasd.o
dasd:initializing...
dasd:Registered successfully to major no 94
dasd(eckd):ECKD discipline initializing
dasd:Registered ECKD discipline successfully
dasd(fba):FBA discipline initializing
dasd:Registered FBA discipline successfully
dasd(eckd):01A0 on sch 0: 3390/0C(CU:3990/04) Cyl:100 Head:15 Sec:224
dasd(eckd):01A0 on sch 0: 3390/0C (CU: 3990/04): Configuration data read
dasd: devno 0x01A0 on subchannel 0 (ECKD) is /dev/dasda (94:0)
dasd(eckd):01A1 on sch 9: 3390/0C(CU:3990/04) Cyl:1500 Head:15 Sec:224
dasd(eckd):01A1 on sch 9: 3390/0C (CU: 3990/04): Configuration data read
dasd: devno 0x01A1 on subchannel 9 (ECKD) is /dev/dasdb (94:4)
dasd:waiting for responses...
dasd(eckd):/dev/dasda (01A0): capacity (4kB blks): 72000kB at 48kB/trk
dasda:(CMS1)/TUX1A0:(MDSK) dasda dasda1
dasd(eckd):/dev/dasdb (01A1): capacity (4kB blks): 1080000kB at 48kB/trk
dasdb:(CMS1)/TUX1A1:(MDSK) dasdb dasdb1
dasd:initialization finished
# mount /dev/dasda1 /mnt
# mount /dev/dasdb1 /mnt/usr -r
```

Apply the patch to the copied file system

The patch is applied to the copied file system by running the updclone.sh program. By running the program with **chroot**, it sees the file system mounted at /mnt as its root file system for the duration of the program.

```
# chroot /mnt /updclone.sh tux80000 192.168.6.2 192.168.6.1 8188
patching file etc/HOSTNAME
patching file etc/hosts
patching file etc/rc.config
patching file etc/route.conf
```

The **updclone.sh** script reads the generic patch and transforms that into a patch specific for this image using the host name and IP address specified. For a less error-prone implementation, you should consider storing the host names and IP addresses in a table on that disk. That way, a **grep** could be used to get the arguments for the **updclone.sh** script.

Example 10-14 The updclone.sh program to apply the patch

```
#!/bin/sh

if [ -z $4 ]; then
    echo "Need hostname IP-address gateway-ip gateway-mtu"
    exit
fi
cat generic.diff | sed "s:/hostname:/$1/ " \
| sed "s:/myip:/$2/ " \
| sed "s:/gwip:/$3/ " \
| sed "s:/gwmtu:/$4/ " \
| patch -p1 $5
```

If you register the Linux images in DNS, you could even consider getting the arguments for **updclone.sh** from that. The **hcp** command could be used to get the user ID of the virtual machine. Given a practical naming convention, **nslookup** could get you the IP address of the Linux image and the gateway.

Shut down the system

The system shutdown should not only unmount the file systems cleanly, but also leave network connections in a better state to be restarted when the image is rebooted. Unfortunately, the starter system does not load a disabled-wait PSW, but loops after a shutdown. You get out of this using the **#CP** command to do the IPL.

Example 10-15 Shutting down the starter system

```
# shutdown -h now
Syncing all buffers...
Sending KILL signal to linuxrc for shutdown...
Sending all processes the TERM signal...
Aug  9 22:05:27 suse exiting on signal 15
Sending all processes the KILL signal...
Syncing all buffers...
Turning off swap...
Unmounting file systems...
/dev/dasdb1 umounted
/dev/dasda1 umounted
/dev/ram2 umounted
```

IPL from the patched root file system

With the patches applied, the Linux image can now be booted from the new disk.

Example 10-16 IPL from the root file system after the patch was applied

IPL 1A0 CLEAR

```
Linux version 2.2.16 (root@Tape.suse.de) (gcc version 2.95.2 19991024
(release))
#1 SMP Sun May 6 06:15:49 GMT 2001
Command line is: ro dasd=0200,01A0,01A1,0cd,0100,0101 root=/dev/dasdbl noinitrd
iucv=$TCPIP
```

```
We are running under VM
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 01A0 on subchannel 0000 - PIM = F0, PAM = F0, POM = FF
Detected device 0009 on subchannel 0001 - PIM = 80, PAM = 80, POM = FF
Detected device 000C on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
```

10.7 Linux IPL from NSS

A Named Saved System (NSS) is like a snapshot copy of part of the memory of a virtual machine. In addition to simulating the normal S/390 IPL of a device, VM can also IPL a virtual machine from a NSS. This is especially efficient for CMS, because the NSS for CMS is defined such that large portions of it can be shared between virtual machines. This reduces the storage requirements for running a large number of CMS virtual machines.

An NSS can have shared and non-shared pages. The S/390 architecture requires all pages in a single 1 MB segment to be either shared or non-shared. The shared pages will be shared among all virtual machines that IPL the NSS. For the non-shared pages, each virtual machine will get its own copy, initialized from the NSS.

10.7.1 Using an NSS with just the kernel

Unfortunately, the Linux for S/390 kernel is not designed to be shared among images. The writable portions of the kernel are mixed with the read-only portions. While the non-shared pages do not reduce overall memory requirements, having memory initialized at startup should at least speed up the boot process.

One of the complications with running the kernel from an NSS is that the kernel parameters (e.g. disk addresses for the DASD driver) need to be the same for each Linux image using this NSS. Fortunately, VM allows us to tailor the virtual machine to fit on the addresses defined for the kernel that was saved in the NSS.

While you can create the NSS by IPL from the virtual reader, it is easier to do when booting from disk because `silob` gives you the information you need to define your NSS.

Example 10-17 Output of silo to compute NSS addresses

```
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
ix 0: offset: 002592 count: 0c address: 0x00000000
ix 1: offset: 00259f count: 80 address: 0x0000c000
ix 2: offset: 00261f count: 80 address: 0x0008c000
ix 3: offset: 00269f count: 76 address: 0x0010c000
ix 4: offset: 001609 count: 01 address: 0x00008000
Bootmap is in block no: 0x0000160a
```

When you run `silob` to make a disk bootable, it displays the memory address where the kernel is going to be loaded. The addresses in Example 10-17 show that different portions of the kernel use the memory from 0 to 0x00182000. This means that the NSS should at least contain the pages 0-181² in exclusive write (EW) mode (and there is no reason to do more than that).

To freeze the Linux image at the correct point during the IPL process, you can use the CP TRACE command. The current Linux for S/390 kernel starts execution at address 0x010000, so the following TRACE command will cause the Linux image to stop at that point.

When execution is stopped at that point, the TRACE command causes the segment to be saved and also ends the trace.

Example 10-18 Defining the NSS and saving it

```
DEFSYS SUSE 0-181 EW MINSIZE=40M
HCPNSD440I The Named Saved System (NSS) SUSE was successfully defined in fileid
0089.
TRACE INST RANGE 10000.2 CMD SAVESYS SUSE "#TRACE END ALL
IPL 1B0 CLEAR
Tracing active at IPL
-> 00010000 BASR ODDO          CC 2
HCPNSS440I Named Saved System (NSS) SUSE was successfully saved in fileid 0089.
Trace ended
```

² You can use the “Scientific” mode of the calculator in your MS Windows accessories to do the computations.

After the NSS has been saved as shown in Example 10-18, the Linux images can boot this kernel with a simple IPL SUSE command.

CP IPL SUSE

```
Linux version 2.2.18 (root@vmlinux6) (gcc version 2.95.2 19991024
SMP Thu Jul 19 10:07:30 EST 2001
```

Because the NSS is defined as exclusive write (EW), the kernel pages are not shared by the Linux images and using the NSS does not reduce overall storage requirements as it does with CMS. Work is in progress to change the layout of the kernel such that significant portions of the code can be shared.

10.7.2 Using an NSS as a starter system

To simplify the boot process, we packaged the kernel with the RAMdisk image in a single NSS that was completely defined as EW. We compared an NSS with a compressed RAMdisk image to one with an uncompressed RAMdisk image, expecting an impressive performance boost when skipping the RAMdisk uncompress at each IPL. However, Table 10-2 shows that the opposite was true in our case.

Table 10-2 Elapsed time to boot from NSS

Compressed RAMdisk image	12 s
Uncompressed RAMdisk image	19 s

A more detailed comparison of the two scenarios showed us that the CPU usage for an IPL from the compressed RAMdisk is indeed significantly higher (as expected), but the 10 seconds of elapsed time missing in the case of the uncompressed RAMdisk turn out to be spooling I/O for CP loading the pages of the NSS in (one at a time).

However, when a Linux image was already running from its non-shared copy of the NSS, the next image could IPL from NSS in 3 seconds—this suggests that CP incorrectly considers all EW pages from a NSS as if they were shared pages (where it is safe to say EW is the most obvious indication of not sharing the page).

For an NSS like CMS, this is not a significant issue since there are just a few EW pages in the segment. However, we can assume that excessive use of an NSS with a lot of EW pages should be avoided.

10.7.3 Picking up IPL parameters

One of the problems with an IPL from NSS is that the kernel parameters are defined in the NSS and will be the same for each image that IPLs the NSS.

The CP IPL command in z/VM has a PARM option that allows the user to pass parameters to the system that is being IPLed. Traditionally, this is used by CMS to control some options in the IPL process (like bypass execution of the system profile). A logical extension of this is to use that option to tailor the command line parameters for Linux as well (for example, to specify what disks to use).

In order to experiment with this, a “quick and dirty” patch was written against the Linux kernel to pick up the argument from the IPL command.

Example 10-19 Patch to pass IPL parameters to the kernel

```

--- boelinux-2.2.16/arch/s390/boot/ipleckd.S  Mon Apr 30 17:22:41 2001
+++ linux/arch/s390/boot/ipleckd.S      Wed Aug  8 18:20:12 2001
@@ -41,6 +41,7 @@

        .org 0xf0                                # Lets start now...
_start: .globl _start
+       stm    %r0,%r15,0x0fc0                  # save the registers for later
        l      %r1,__LC_SUBCHANNEL_ID          # get IPL-subchannel from lowcore
        st     %r1,__LC_IPLDEV                 # keep it for reipl
        stsch  .Lrdcdata
@@ -112,7 +113,23 @@
        mvc   0x500(256,%r3),0x80(%r4)
        mvc   0x600(256,%r3),0x180(%r4)
        mvc   0x700(256,%r3),0x280(%r4)

-.Lrunkern:
+.Lrunkern:                                # We align here to 0x0200
+       j      .stopnss                        # because that is easy to
+       .org   0x0200                          # remember for the trace
+.stopnss:
+       lm     %r0,%r15,0x0fc0                 # last instr when not NSS
+       stm    %r0,%r15,0x0fc0                 # first instr from NSS
+       tr     0x0fc0(64,0),.ebcasc            # translate saved registers
+       lm     %r3,%r4,.Lstart
+.find00:
+       cli   0x480(%r3),0x00                  # end of string?
+       la    %r3,1(%r3)
+       jnz   .find00
+       mvi   0x47f(%r3),0x020                 # put a blank instead of 0x00
+       mvc   0x480(64,%r3),0x0fc0
+       mvi   0x4c0(%r3),0x00                  # and a 0x00 in case all 64 used
+       lm     %r3,%r4,.Lstart
+.notnss:
+       #     lhi    %r2,17
+       #     sll   %r2,12
+       #     st    %r1,0xc6c(%r2)             # store iplsubchannel to lowcore
@@ -296,7 +313,43 @@
        .long 0x47400010,0x00000000+.Llodata
.Lrdccw:

```

```

        .long 0x86400000,0x00000000
-       .org 0x800
+
+
+       .org 0xe00      # EBCDIC to lowercase ASCII table
+.ebcasc:
+       .byte 0x00,0x01,0x02,0x03,0x07,0x09,0x07,0x7F
+       .byte 0x07,0x07,0x07,0x0B,0x0C,0x0D,0x0E,0x0F
+       .byte 0x10,0x11,0x12,0x13,0x07,0x0A,0x08,0x07
+       .byte 0x18,0x19,0x07,0x07,0x07,0x07,0x07,0x07
+       .byte 0x07,0x07,0x1C,0x07,0x07,0x0A,0x17,0x1B
+       .byte 0x07,0x07,0x07,0x07,0x07,0x05,0x06,0x07
+       .byte 0x07,0x07,0x16,0x07,0x07,0x07,0x07,0x04
+       .byte 0x07,0x07,0x07,0x07,0x14,0x15,0x07,0x1A
+       .byte 0x20,0xFF,0x83,0x84,0x85,0xA0,0x07,0x86
+       .byte 0x87,0xA4,0x9B,0x2E,0x3C,0x28,0x2B,0x7C
+       .byte 0x26,0x82,0x88,0x89,0x8A,0xA1,0x8C,0x07
+       .byte 0x8D,0xE1,0x21,0x24,0x2A,0x29,0x3B,0xAA
+       .byte 0x2D,0x2F,0x07,0x8E,0x07,0x07,0x07,0x8F
+       .byte 0x80,0xA5,0x07,0x2C,0x25,0x5F,0x3E,0x3F
+       .byte 0x07,0x90,0x07,0x07,0x07,0x07,0x07,0x07
+       .byte 0x70,0x60,0x3A,0x23,0x40,0x27,0x3D,0x22
+       .byte 0x07,0x61,0x62,0x63,0x64,0x65,0x66,0x67
+       .byte 0x68,0x69,0xAE,0xAF,0x07,0x07,0x07,0xF1
+       .byte 0xF8,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70
+       .byte 0x71,0x72,0xA6,0xA7,0x91,0x07,0x92,0x07
+       .byte 0xE6,0x7E,0x73,0x74,0x75,0x76,0x77,0x78
+       .byte 0x79,0x7A,0xAD,0xAB,0x07,0x07,0x07,0x07
+       .byte 0x5E,0x9C,0x9D,0xFA,0x07,0x07,0x07,0xAC
+       .byte 0xAB,0x07,0x5B,0x5D,0x07,0x07,0x07,0x07
+       .byte 0x7B,0x61,0x62,0x63,0x64,0x65,0x66,0x67
+       .byte 0x68,0x69,0x07,0x93,0x94,0x95,0xA2,0x07
+       .byte 0x7D,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70
+       .byte 0x71,0x72,0x07,0x96,0x81,0x97,0xA3,0x98
+       .byte 0x5C,0xF6,0x73,0x74,0x75,0x76,0x77,0x78
+       .byte 0x79,0x7A,0xFD,0x07,0x99,0x07,0x07,0x07
+       .byte 0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37
+       .byte 0x38,0x39,0x07,0x07,0x9A,0x07,0x07,0x07
+
+       # end of pre initialized data is here CCWarea follows
+       # from here we load 1k blacklist
+       # end of function

```

Note that the patch shown in Example 10-19 is not a production-strength solution, and it has not yet been submitted to the IBM team in Boeblingen who maintain the S/390-specific portions of the kernel source.

Note: The IPL parameters are not restricted to the IPL from NSS. You can also use them when you IPL from disk. With a little more code in the kernel, it's possible to make the code pick up a SAVESYS option to save the NSS, as is done for CMS.

The length of the parameters passed on the IPL command is restricted to 64 characters. This may not be sufficient in many cases, so we wrote the code such that the parameters are appended to whatever is already in the kernel (i.e. the parameters in the parameter file as it was used in silo).

Tip: Additional parameters specified in the kernel command line (not used by the kernel or device drivers) end up as environment variables to the boot scripts. This can be used to pass options to the boot scripts (e.g. IP address).

To exploit this patch, a trace should be set up to stop execution at 0x0200, at which point the SAVESYS command can be issued, as shown in Example 10-20.

Example 10-20 Defining and saving the NSS

```
CP DEFSYS SUSE216A 0-181 EW MINSIZE=40M PARMREGS=0-15
The Named Saved System (NSS) SUSE216A was successfully defined in fileid 0110.
TRACE I R 200.2
IPL 1A0 CLEAR
Tracing active at IPL
-> 00000200 LM 980F0FC0 00000FC0 CC 2
SAVESYS SUSE216A
Named Saved System (NSS) SUSE216A was successfully saved in fileid 0110.
TRACE END ALL
Trace ended
```

This is very similar to what is shown in 10.7.1, “Using an NSS with just the kernel” on page 228, except for the different address to freeze the IPL.

Example 10-21 Demonstrate the modified kernel command line

```
IPL SUSE216A PARM TEST=EXAMPLE
Linux version 2.2.16 (root@tux60000) (gcc version 2.95.2 19991024 (release)) #1
SMP Wed Aug 1 18:21:27 EST 2001
Command line is: ro dasd=0200,01A0,01A1,0cd root=/dev/dasdb1 noinitrd
test=example
We are running under VM
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 01A0 on subchannel 0000 - PIM = F0, PAM = F0, POM = FF
Detected device 01A1 on subchannel 0001 - PIM = F0, PAM = F0, POM = FF
Detected device 0009 on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
```

The IPL in Example 10-21 shows the **test=example** option added to the IPL command. This shows up again at the end of the command line echoed by the kernel.

Note: Because CP will uppercase the command when typed in on the console, the patch was made to translate it to lowercase. This way we can specify the options for Linux that need to be lowercase. This obviously causes problems when you want to do things in uppercase. Doing it right will take more thinking and more coding.

The current implementation of the DASD driver requires all disks to be specified in a single `dasd=` parameter. If we want to tailor the list of disks at each IPL, we need to specify them all on the IPL command. When the `dasd=` parameter is specified more than once, the last one is used. This is useful because the patch allows us to override the list of disks defined in the parameter file.

You can do more with this than override the `dasd=` parameter. Parameters that are not processed by the built-in driver during the boot process will be made available as environment variables to the init process. This means you can pick up values in the boot scripts (e.g. to configure your IP address). If you need the values after the init process has completed, you can take the contents of the `/proc/cmdline` pseudo file to retrieve the parameters.



Network infrastructure design

This chapter uses the networking theory introduced in Chapter 4, “Networking a penguin colony” on page 73 to help you design your virtual networking environment.

11.1 Virtual IP addressing

In this section we describe how to set up our virtual IP address solution using the Linux dummy interface. In 4.2.3, “Virtual IP addressing” on page 79, we introduce the concept of virtual IP addressing using dummy, and this method becomes part of the solution presented in the remainder of this chapter.

11.1.1 Sample configuration

In our example scenario, we have a single Linux instance which uses private IP addressing on the CTC device to the router, but requires a public IP address for network connectivity.

11.1.2 Compiling dummy.o

If dummy interface support is not present in your kernel, you’ll have to build it. Although this is not a major task, instructions on how to build a complete new kernel are beyond the scope of this book. Instead, you can refer to the Linux HOWTOs to find one on kernel compilation that describes the basics.

In your kernel configuration file, dummy support is controlled by the `CONFIG_DUMMY` variable. Make this value `Y` or `M` to include dummy support. Figure 11-1 on page 237 shows a `make menuconfig` session at the S/390 Network device support panel. Here, we have selected to add the dummy support as a module. We recommend that the dummy support be built as a module, as this provides the least intrusive way of adding the support to a running system.

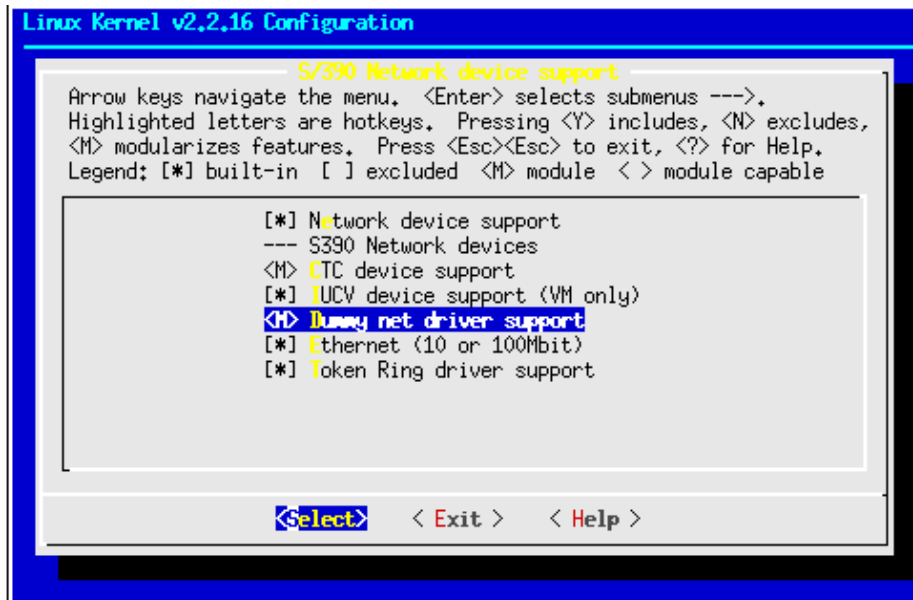


Figure 11-1 Compiling dummy net driver support into kernel

Once you have updated the kernel configuration, you can make just the network module directory with the following command (issued from the root of the Linux source tree):

```
make modules SUBDIRS=drivers/net
```

In our case, we received the following output from the compilation:

```
# make modules SUBDIRS=drivers/net
make -C drivers/net CFLAGS="-Wall -Wstrict-prototypes -O2
-fomit-frame-pointer -fno-strict-aliasing -D__SMP__ -pipe
-fno-strength-reduce -DMODULE" MAKING_MODULES=1 modules
make[1]: Entering directory `/usr/src/linux-2.2.16.SuSE/drivers/net'
gcc -D__KERNEL__ -I/usr/src/linux/include -Wall -Wstrict-prototypes -O2
-fomit-frame-pointer -fno-strict-aliasing -D__SMP__ -pipe
-fno-strength-reduce -DMODULE -c -o dummy.o dummy.c
rm -f $TOPDIR/modules/NET_MODULES
echo dummy.o >> $TOPDIR/modules/NET_MODULES
echo drivers/net/
drivers/net/
cd $TOPDIR/modules; for i in dummy.o; do \
  ln -sf ../drivers/net//$i $i; done
make[1]: Leaving directory `/usr/src/linux-2.2.16.SuSE/drivers/net'
```

Once the module has been built, it must be copied to the `/lib/modules` tree so that the kernel utilities can find the module when required. The easiest way is to simply copy the module using the following command:

```
# cp drivers/net/dummy.o /lib/modules/2.2.16/net/
```

This is obviously for a 2.2.16 kernel; you will need to confirm the correct directory under `/lib/modules/` for your system.

Attention: Usually, the command `make modules_install` is used to copy newly-compiled modules into the right `/lib/modules` directory. However, if you used the `make` command shown above and have not previously built a kernel and full modules in this source tree, do not use the `make modules_install` command to install the new module. Doing so would delete all of your existing modules.

After copying the module, run the command `depmod -a` to recreate the module dependency file. Once this is complete, you can issue `modprobe` or `insmod` to install the dummy module:

```
# insmod dummy
Using /lib/modules/2.2.16/net/dummy.o
```

You are now ready to configure a dummy interface.

11.1.3 Configuring dummy0

In our example, our Linux instance has a CTC device configured with an IP address of 192.168.11.1. We want to add a dummy interface with the IP address 9.12.6.99, which is visible to outside our network. The following command will do this:

```
# ifconfig dummy0 9.12.6.99 broadcast 9.12.6.99 netmask 255.255.255.255 mtu 1500
```

Now, we can ping our interface to see that it is active:

```
# ping 9.12.6.99
PING 9.12.6.99 (9.12.6.99): 56 data bytes
64 bytes from 9.12.6.99: icmp_seq=0 ttl=255 time=10.469 ms
64 bytes from 9.12.6.99: icmp_seq=1 ttl=255 time=5.903 ms
--- 9.12.6.99 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 5.903/8.186/10.469 ms
```

Our dummy interface is ready to receive traffic. To complete the configuration, the rest of the network will have to be configured to direct traffic for the virtual IP address to the correct Linux instance. In our test case, we entered static routes in the intervening routers. This allowed us to connect to services from Windows machines on the network to our Linux instance using the virtual IP address; see Figure 11-2:

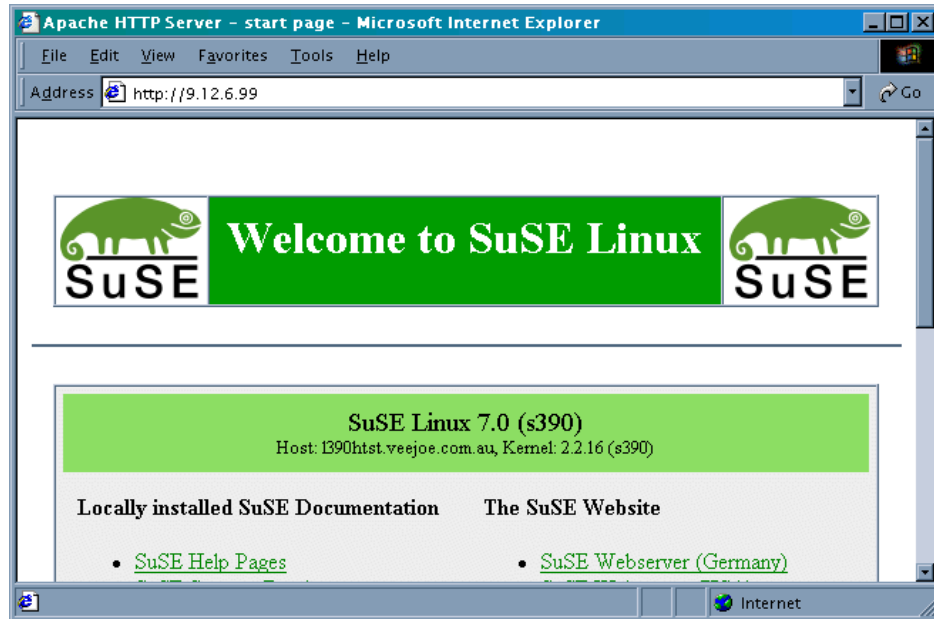


Figure 11-2 Accessing server using virtual IP address

The next step would be to add the virtual IP address to name resolution services such as DNS or NIS. When clients request the address for our server, these services should respond with the virtual IP address rather than an interface address. The interface addresses can remain in the name server, but with a different name that identifies which *interface* in the server the address belongs to (such as hub port or device name). This can aid problem resolution when tracing connectivity faults.

Important: This method of implementing a virtual IP address only works for connections inbound to the Linux instance. For outbound connections, Linux will use the address of the network interface. We can use NAT to make the traffic appear to originate at the dummy interface, and this is explained in 11.2.12, “Examples for using NAT in the enterprise and ISP/ASP” on page 265.

11.1.4 Using virtual IP addressing in penguin colonies

It may seem intensive to use host routes to access virtual IP addresses on numerous Linux instances. However, when combined with a hierarchical routing model, the routes required to reach the Linux servers can be aggregated into small network routes.

For example, if you have a number of second-level Linux routers with worker penguins behind them, you would allocate a small subnet of IP addresses to each Linux router. The addresses used by the workers would be allocated from this subnet. This enables you to have only a single subnet route, for the addresses behind each Linux router. This process could be continued back up the chain of Linux routers, “supernetting” the addresses from each lower level.

Attention: This process is known as *route summarization*, and is often performed in TCP/IP networks to control and minimize the amount of routing data transferred between routers.

Normally, route summarization is performed automatically by dynamic routing protocols in routers, but the method shown here reduces (or even eliminates) the need to run a dynamic routing protocol within the penguin colony.

In this example, worker penguins reside at the third level of a penguin colony design. Two levels of routing are being performed, with the first level done by a VM TCP/IP guest.

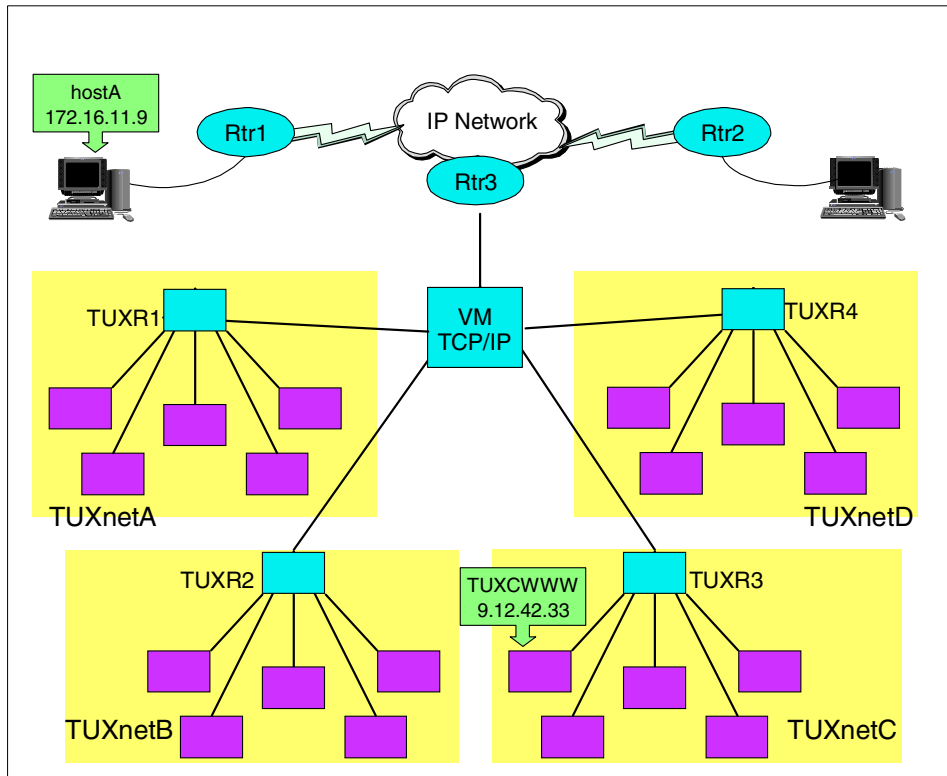


Figure 11-3 Example penguin colony network

We are using IUCV connections inside the penguin colony shown in Figure 11-3, so we work within the limitations of the IUCV driver in having a maximum of 10 IUCV links defined. This means that each router serves nine workers. We also must work within TCP/IP subnetting rules, which dictate that to provide 9 IP addresses in a subnet we must use a '/28' mask, which provides 14 addresses for use (this gives us a small amount of room for expansion).

Virtual Subnet	Router name	Virtual IP hosts	Route served
TUXnetA	TUXR1	9.12.42.1 - 9.12.42.14	9.12.42.0/28
TUXnetB	TUXR2	9.12.42.17 - 9.12.42.30	9.12.42.16/28
TUXnetC	TUXR3	9.12.42.33 - 9.12.42.46	9.12.42.32/28
TUXnetD	TUXR4	9.12.42.49 - 9.12.42.62	9.12.42.48/28

Each router penguin must have the host routes to all the worker penguins it manages (these would usually be added automatically when the interfaces are configured, but because we are using virtual addresses on dummy interfaces, they would have to be configured in advance). In the VM TCP/IP stack, however, only the four subnet routes shown in the previous table are needed. The relevant section of the GATEWAY statement from the TCP/IP PROFILE of the TCP/IP guest is shown here:

```
GATEWAY
; IP Network First      Link      Max. Packet  Subnet      Subnet
; Address   Hop          Name      Size (MTU)  Mask        Value
; -----
  9         192.168.1.1  TUXR1     1500        0.255.255.240  0.12.42.0
  9         192.168.1.2  TUXR2     1500        0.255.255.240  0.12.42.16
  9         192.168.1.3  TUXR3     1500        0.255.255.240  0.12.42.32
  9         192.168.1.4  TUXR4     1500        0.255.255.240  0.12.42.48
```

Routers in the enterprise network that need to direct traffic to VM TCP/IP for forwarding now only require a single route that covers all of our worker penguins. In this case, the total range of host addresses goes from 9.12.42.1 to 9.12.42.62. This is the 9.12.42.0 network with a 255.255.255.192 mask.

As an example, if the network uses Cisco routers, the IOS configuration command to add this network route is:

```
Rtr3(config)# ip route 9.12.42.0 255.255.255.192 192.168.0.1
```

where 192.168.0.1 is the address of the OSA adapter used by VM TCP/IP to connect to the IP network. This command would be entered at the router adjacent to the OSA.

Note: It should not be necessary to manually enter this route into all routers in the network. A dynamic routing protocol in the IP network would usually take care of propagating the route to the rest of the network. Also, by using a dynamic routing protocol in VM TCP/IP, the route could be advertised directly from VM.

Alternatively, to allow a Linux host on the same network as the OSA to connect to the penguin colony, the route command would look like this:

```
# route add -net 9.12.42.0 netmask 255.255.255.192 gw 192.168.0.1
```

To verify this, we can follow the path of a packet through the network. In our diagram, hostA starts a Web browser and requests a document from TUXCWWW. The IP address of TUXCWWW is 9.12.42.33, which is within the network 9.12.42.0/26.

The routers in the network have learned that the bottom router in the diagram is the path to the 9.12.42.0/26 network, so the packet is forwarded there, and that router forwards it to VM TCP/IP. When the VM TCP/IP stack receives the packet, it recognizes that 9.12.42.33 is part of the 9.12.42.32/28 network, and forwards the packet to TUXR3.

Finally the packet arrives at TUXR3, and it knows that 9.12.42.33 is a host route accessible via the IUCV connection to the TUXCWWW guest. The packet is forwarded to TUXCWWW, the destination.

The path for return traffic, while inside the penguin colony, would simply use default routes. The worker penguins nominate the router penguin as default, and the router penguins nominate VM TCP/IP as default. VM TCP/IP, in turn, would use the appropriate network router as default. Once into the IP network, normal IP routing would return the traffic to hostA.

If we need to add more routers for more penguins, changing the mask from 255.255.255.192 to 255.255.255.128 would change the definition to the 9.12.42.0/25 network, which covers host addresses from 9.12.42.1 to 9.12.42.126. This means that we can add another four routers to the configuration, carrying on the same pattern for subnet allocation as the original four routers.

Attention: Consult your networking team, or a reference on TCP/IP routing, before you decide on an addressing scheme. There are dependencies on subnetting and supernetting that can become complex and are not easily changed after the network is in production, so planning for expansion is very important.

Also, the details of ensuring that the routes to the worker penguins are properly distributed to the IP network must be analyzed. As host configurators, don't assume that just because the routing network uses a dynamic routing protocol, that the right thing will happen "automagically". Discuss your requirements with the network team, so that a solution which is mutually agreeable can be achieved.

11.2 Packet filtering and NAT with IPTables

In this section we present an example configuration using the designs discussed in 4.3, "Packet filtering and Network Address Translation" on page 81.

11.2.1 What you need to run packet filtering

To set up a packet filter server with IPTables, your Linux installation needs to meet the following requirements:

1. You need kernel Version 2.4.5 or higher with the latest S/390 patches from the Linux for S/390 developerWorks page at:

<http://www10.software.ibm.com/developerworks/opensource/linux390/index.shtml>

We recommend that you use the latest available stable version. The kernel has to be compiled with support for netfilters. This means that you have to select **CONFIG_NETFILTER** in the kernel configuration. Depending of your packet filtering configuration, you also need to select the proper configuration options under **IP: Netfilter Configuration**. We recommend that you compile all your networking options and available modules. If you want to use your Linux server as a router, choose **IP - advanced router** under **TCP/IP networking**. This will also increase the routing performance.

2. Loadable kernel modules version 2.4.6 or newer with the latest S/390 patches from the Linux for S/390 developerWorks page.
3. IPTables 1.2.2 or newer.

Important: IPTables is the follow-on to IPChains, which was used in the 2.2 kernel.

If your Linux installation does not include the required software levels, you should download all available patches and compile new versions of the software yourself. Usually the kernel that comes with a distribution is not enabled to support netfilters, so you need to recompile the kernel with the required support.

11.2.2 Network configuration for a packet filtering implementation

In this section we describe our lab network setup for implementing a packet filtering solution.

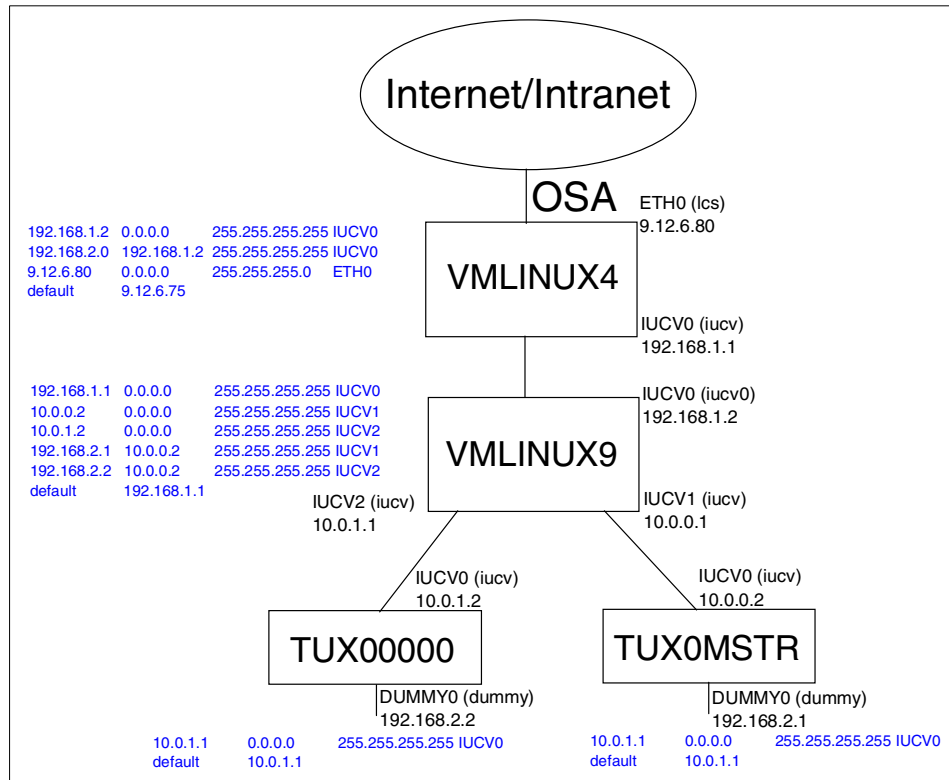


Figure 11-4 Lab network setup No.1

As you can see in Figure 11-4, we set up a two-layered router/firewall implementation. All our Linux images are running inside VM. In the example we have the following setup:

1. vmlinux4

vmlinux4 is acting as a main router for our penguin colony. It is connected to the outside world via an Ethernet connection using an lcs driver. The setup for the eth0 interface is as follows:

```
ETH0 9.12.6.80 255.255.255.0
```

vmlinux4 is then connected to the second-layer router/firewall VM Linux9 over the IUCV connection:

```
IUCV0 192.168.1.1 Pointopoint 192.168.1.2
```

In our example, we used SuSE 7.2 31-bit distribution; the entries in the `/etc/rc.config` file for this setup are shown in Example 11-1 on page 246.

Example 11-1 /etc/rc.config file entries for vmlinux4

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Addresses
#
IPADDR_0="9.12.6.80"
IPADDR_1="192.168.1.1"
IPADDR_2="192.168.2.1"
IPADDR_3="10.0.0.1"

#
# Network device names (e.g. "eth0")
#
NETDEV_0="eth0"
NETDEV_1="iucv0"
NETDEV_2="NONE"
NETDEV_3="NONE"

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpcclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="9.12.6.80 broadcast 9.12.6.255 netmask 255.255.255.0 mtu 1492 up"
IFCONFIG_1="192.168.1.1 pointopoint 192.168.1.2 mtu 1492 up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

We also have to set up the correct routing tables so that our second-level router/firewall and the servers connected to it will be accessible from the outside world. In our example, all the servers connected to the second-layer router/firewall have IP addresses in the 192.168.2.0 subnet.

In our example we have the following routes:

- Default route 9.12.6.75
- Route to the 192.168.2.0 subnet via the second-layer router/firewall

This means that we forward all the traffic to the subnet 192.168.2.0 to the second-level router/firewall.

Example 11-2 shows the `/etc/route.conf` file used in our installation:

Example 11-2 /etc/route.conf file for vmlinux4

9.12.6.0	0.0.0.0	255.255.255.0	eth0
192.168.1.2	0.0.0.0	255.255.255.255	iucv0
default	9.12.6.75	0.0.0.0	eth0
192.168.2.0	192.168.1.2	255.255.255.0	iucv0

2. vmlinux9

vmlinux9 is our second layer router/firewall. vmlinux9 is connected to the vmlinux4 over the IUCV connection IUCV0:

```
IUCV0 192.168.1.2 Pointopoint 192.168.1.1
```

Each server connected to the router/firewall on the second layer uses its own IUCV interface. In our example, we have two servers connected with IUCV1 and IUCV2 connections:

```
IUCV0 10.0.0.1 Pointopoint 10.0.0.2
IUCV0 10.0.1.1 Pointopoint 10.0.1.2
```

Example 11-3 shows the `/etc/rc.config` file entries for this setup:

Example 11-3 /etc/rc.config file entries for vmlinux9

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0_1_2_3" for four cards
#
NETCONFIG="_0_1_2"

#
# IP Adresses
#
IPADDR_0="192.168.1.2"
IPADDR_1="10.0.0.1"
IPADDR_2="10.0.1.1"
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
```

```

NETDEV_1="iucv1"
NETDEV_2="iucv2"
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpcclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="192.168.1.2 pointopoint 192.168.1.1 mtu 1492 up"
IFCONFIG_1="10.0.0.1 pointopoint 10.0.0.2 mtu 1492 up"
IFCONFIG_2="10.0.1.1 pointopoint 10.0.1.2 mtu 1492 up"
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"

```

We also have to set up the correct routing tables so that the servers connected to this router will be accessible from outside world. In our example, all servers have IP addresses in the 192.168.2.0 subnet and we have the following routes:

- Default route 192.168.1.1
- Route to the 192.168.2.1 address via the IUCV1 connection
- Route to the 192.168.2.2 address via the IUCV2 connection

The traffic for each individual server is forwarded to the unique connection used just for this server. In this case, only the packets with the address of the destination server will go to this server.

Example 11-4 shows the `/etc/route.conf` file used in our configuration:

Example 11-4 /etc/route.conf file for vmlinux9

192.168.1.1	0.0.0.0	255.255.255.255	iucv0
10.0.0.2	0.0.0.0	255.255.255.255	iucv1
10.0.1.2	0.0.0.0	255.255.255.255	iucv2
default	192.168.1.1	0.0.0.0	iucv0
192.168.2.1	10.0.0.2	255.255.255.255	iucv1
192.168.2.2	10.0.1.2	255.255.255.255	iucv2

3. tux0mstr

tux0mstr is a first server in our colony. It has an IUCV connection IUCV0 to the second-layer router/firewall vmlinux9:

```
IUCV0 10.0.0.2 Pointopoint 10.0.0.1
```

The real address of the server is implemented on the DUMMY0 interface. We used this approach because we wanted to have the real subnet in the connection to the router/firewall:

```
DUMMY0 192.168.2.1 255.255.255.255
```

Example 11-5 shows the corresponding entries in the `/etc/rc.config` file used in our configuration:

Example 11-5 /etc/rc.config file entries for tux0mstr

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Adresses
#
IPADDR_0="10.0.0.2"
IPADDR_1="192.168.2.1"
IPADDR_2=""
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
NETDEV_1="dummy0"
NETDEV_2=""
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpcclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="10.0.0.2 pointopoint 10.0.0.1 mtu 1492 up"
IFCONFIG_1="192.168.2.1 broadcast 192.168.2.1 netmask 255.255.255.255 mtu 1492
up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
```

```
IP_FORWARD="yes"
```

For the routing, we just need to set up the default route to the second-layer router/firewall. This means that all the packets are traveling back to the router.

Example 11-6 shows the `/etc/route.conf` file used in our configuration:

Example 11-6 tux0mstr /etc/route.conf file

10.0.0.1	0.0.0.0	255.255.255.255	iucv0
default	10.0.0.1	0.0.0.0	iucv0

4. TUX00000

TUX00000 is our second server in the colony and uses the same approach for the network setup as tux0mstr. Example 11-7 shows the `/etc/rc.config` file entries:

Example 11-7 /etc/rc.config file entries for TUX00000

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Adresses
#
IPADDR_0="10.0.1.2"
IPADDR_1="192.168.2.2"
IPADDR_2=""
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
NETDEV_1="dummy0"
NETDEV_2=""
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpcclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="10.0.1.2 pointopoint 10.0.1.1 mtu 1492 up"
```

```
IFCONFIG_1="192.168.2.2 broadcast 192.168.2.2 netmask 255.255.255.255 mtu 1492
up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

Example 11-8 shows the `/etc/route.conf` file used in our configuration:

Example 11-8 TUX00000 /etc/route.conf file

10.0.1.1	0.0.0.0	255.255.255.255	iucv0
default	10.0.1.1	0.0.0.0	iucv0

As you can see from the `/etc/rc.config` file entries, we enabled IP forwarding on both of our routers/firewalls and servers. With IP forwarding enabled, Linux can act as a router. In our example we did not use any router daemons for the routing purposes, but instead simply defined static routes.

11.2.3 How to permanently enable IP forwarding

By default, IP Forwarding is not enabled. In order to enable it, edit the `/etc/rc.config` file and make sure `IP_FORWARD` is set to `yes`. You can check the status of `IP_FORWARD` setting with the following command:

```
# grep IP_FORWARD /etc/rc.config
```

The output should be similar to:

```
IP_FORWARD="yes"
```

Run `SuSEconfig` to commit the changes, and restart the network by typing the following:

```
# rcnetwork restart
```

Alternatively, you can execute the following command:

```
# /etc/init.d/network restart
```

You can check if your IP forwarding is enabled by executing the following command:

```
# cat /proc/sys/net/ipv4/ip_forward
```

If IP forwarding is enabled, the output of this command will be `1`.

Now your server is ready to act as a router. You can verify this by pinging the eth0 interface with IP address 9.12.6.80 from the tux0mstr Linux on 192.168.2.1 IP address; you are pinging the external interface in our main router from the Linux server on the 10.0.0.2 IP.

As you'll notice, there will be no reply to this ping because the `ping` command will use the IP address of the IUCV0 connection as the source address—not the address of the DUMMY0 adapter that is used to assign the external address of the server. When the packet is received by the main router/firewall, this will reply to the address 10.0.0.2, because this address was specified as the source for the ping.

Since we have not have defined any special route to the 10.0.0.0 subnet, the packet will go the default gateway 9.12.6.75, and of course the gateway will drop the packet because it does not have the route definition for the 10.0.0.0 subnet. To resolve this issue, we have to add the following route to the `/etc/route.conf` file on the vmlinux4 Linux system:

- Route to the 10.0.0.0 subnet via the second layer router/firewall

Example 11-9 shows the modified `/etc/route.conf` file:

Example 11-9 Modified /etc/rc.config file

9.12.6.0	0.0.0.0	255.255.255.0	eth0
192.168.1.2	0.0.0.0	255.255.255.255	iucv0
default	9.12.6.75	0.0.0.0	eth0
192.168.2.0	192.168.1.2	255.255.255.0	iucv0
10.0.0.0	192.168.1.2	255.255.0.0	iucv0

After modifying this file, you should execute the following command:

```
# /etc/initd.d/route restart
```

Now you should try to execute the `ping` command on the tux0mstr Linux again:

```
# ping 9.12.6.80
```

If the ping is successful, both of your routers are working correctly. You will see output similar to Example 11-10:

Example 11-10 Pinging the main router external interface

```
tux0mstr:~ # ping 9.12.6.80
PING 9.12.6.80 (9.12.6.80): 56 data bytes
64 bytes from 9.12.6.80: icmp_seq=0 ttl=254 time=0.405 ms
64 bytes from 9.12.6.80: icmp_seq=1 ttl=254 time=0.425 ms
64 bytes from 9.12.6.80: icmp_seq=2 ttl=254 time=0.430 ms
64 bytes from 9.12.6.80: icmp_seq=3 ttl=254 time=0.427 ms
64 bytes from 9.12.6.80: icmp_seq=4 ttl=254 time=0.403 ms
```



```
--- 9.12.6.80 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.403/0.418/0.430 ms
```

To access the servers on 192.168.2.0 subnet in our colony from the outside world, the routers on the network have to be updated to reflect this configuration. This means that all the traffic for the 192.168.2.0 subnet has to go to the IP address 9.12.6.80 in our example.

In our test environment, we had the computers on the same subnet as our main router external interface. So we simply added the static route with the command from the Windows 2000 command prompt, and we were able to access the servers in our colony:

```
C:\> route add 192.168.2.0 mask 255.255.255.0 9.12.6.80
```

At this point, following our process, the routers for your Linux colony inside the zSeries should also be successfully set up.

11.2.4 The first IP Tables rules

Now we can deploy the routers. We want to limit the access to our servers in the colony, as shown in Figure 11-4 on page 245, so in our example, we'll allow only HTTP protocol to our servers. Before we implement our protection, however, we try to ping our server with the command:

```
# ping 192.168.2.1
```

Example 11-11 shows the result:

Example 11-11 Pinging the server

```
C:\>ping 192.168.2.1
```

Pinging 192.168.2.1 with 32 bytes of data:

```
Reply from 192.168.2.1: bytes=32 time=10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253
```

Ping statistics for 192.168.2.1:

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 10ms, Average = 2ms
```

The following steps will implement security rules for the servers.

Note: We discuss the meaning of the iptables commands in “Using IP Tables” on page 256.

1. We created a new chain for our rules with the command:

```
# iptables -N httpallow
```

2. We defined the rule, in our httpallow chain, which allows all connections from the computer we’re using for remote management of the router/firewall with the command:

```
# iptables -A httpallow -s 9.12.6.133 -j ACCEPT
```

3. We defined the rules, in our httpallow chain, which allows building a new connection only for the HTTP protocol with the command:

```
# iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT  
# iptables -A httpallow -m state --state NEW -p UDP--dport www -j ACCEPT
```

4. We defined the rule, in our httpallow chain, which keeps alive the established and related connections with the command:

```
# iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
```

5. With the following command we defined the rule, in our httpallow chain, which drops all other incoming packets:

```
# iptables -A httpallow -j DROP
```

6. Finally, with the following commands, we need to define that all the packets from the INPUT and FORWARD chain will jump into our httpallow chain:

```
# iptables -A INPUT -j httpallow  
# iptables -A FORWARD -j httpallow
```

Now we were ready to test our security implementation. We pinged the server with the command:

```
# ping 192.168.2.1
```

Example 11-12 shows the output:

Example 11-12 Pinging after applying security rules

```
C:\>ping 192.168.2.1
```

```
Pinging 192.168.2.1 with 32 bytes of data:
```

```
Reply from 9.32.44.3: Destination host unreachable.  
Reply from 9.32.44.3: Destination host unreachable.  
Reply from 9.32.44.3: Destination host unreachable.  
Request timed out.
```

Ping statistics for 192.168.8.1:
Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms

As you can see, **ping** is not working. This means our security was successful! Next we tried to connect to the server with a browser; Figure 11-5 shows the screen we received:



Figure 11-5 Accessing the Web server after applying security rules

11.2.5 Checking your filter

With the `/usr/sbin/iptables` command you can set up your rules for packet checking.

Note: By default, all checking policies are set to Accept. This means that all packets can come in, go through, or go out from your server without any restrictions.

You can examine the current checking policies by using the `-L` flag with the `iptables` command. You should see output similar to that shown in Example 11-13:

Example 11-13 Listing of the default IP Tables policies

```
# iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
```

11.2.6 Using IP Tables

With the `/usr/sbin/iptables` command, you can create, change or delete your own policies for checking packets or you can modify built-in policies. You cannot delete the built-in chains, but you can append your rules to the existing chains or even create your own chains.

To manage whole chains you can use the parameters described in Table 11-1.

Table 11-1 Parameters for managing the whole chains

Parameter	Description
-N	Create a new chain
-X	Delete an empty chain
-P	Change the policy for a built-in chain
-L	List the rules in a chain
-F	Flush the rules out of a chain
-Z	Zero the packets and the byte counters on all rules in a chain
-E	Rename the chain

For manipulating rules inside the chain, you can use the parameters explained in Table 11-2.

Table 11-2 Parameters for managing rules in the chain

Parameter	Description
-A	Append new rule to a chain
-I	Insert a new rule in a chain at some position
-R	Replace a rule at some position in a chain
-D	Delete a rule at some position in a chain
-D	Delete the first rule that matches in a chain

11.2.7 How to create a rule

Each rule specifies a set of conditions the packet must meet, and what to do if those conditions are met.

The most common syntax for creating a new rule is as follows:

```
# /usr/sbin/iptables -t table -A INPUT -s source -d destination \
# -p protocol -i input_interface -o output_interface -f -p extension \
# -m match_extension -j target
```

The parameters are described in the Table 11-3.

Table 11-3 IP Tables parameters

Parameter	Description
-t table	The table to manipulate. If you omit this parameter, the table “filter” will be used. The “filter” table holds the rules for packet filtering.
-A INPUT	Append a new rule to an INPUT chain.
-s source	IP address or host name of the source.
-d destination	IP address or host name of the destination.
-p protocol	Type of the protocol to which a rule is applied.
-i input_interface	The input network interface to match.
-o output_interface	The output network interface to match.
-f	Fragments flag - if this is used, the rule is only valid for second and further fragments through.

Parameter	Description
-p extension	<p>With this you invoke extension. The following “new match” extensions are available (you can also use a custom-supplied extension):</p> <ol style="list-style-type: none"> 1. TCP (-p tcp), the parameters are: <ul style="list-style-type: none"> --tcp-flags (ALL,SYN,ACK,FIN,RST,URG,PSH,NONE) --syn (short for --tcp-flags SYN,RST,ACK,SYN) --sport (source port) --dport (destination port) --tcp-option (examine TCP options) 2. UDP (-p udp), the parameters are: <ul style="list-style-type: none"> --sport (source port) --dport (destination port) 3. ICMP (-p icmp), the parameters are: <ul style="list-style-type: none"> --icmp-type (icmp type)
-m match_extension	<p>With this option you load “other match” extensions:</p> <ol style="list-style-type: none"> 1. mac (-m mac), the parameters are: <ul style="list-style-type: none"> --mac-source (source MAC address) 2. limit (-m limit), the parameters are: <ul style="list-style-type: none"> --limit (maximum average number of matches per second) --limit-burst (maximum burst, before --limit takes over) 3. owner (-m owner), the parameters are: <ul style="list-style-type: none"> --uid-owner (user ID) --gid-owner (group ID) --pid-owner (process ID) --sid-owner (session ID) 4. state (-m state), the parameters are: <ul style="list-style-type: none"> --state (NEW - new packets, ESTABLISHED - packets which belong to the established connection, RELATED - related packets, INVALID - unidentified packets)

Parameter	Description
-j target	<p>What we do with the packet that matches the rule. The built-in targets are:</p> <ol style="list-style-type: none"> 1. ACCEPT - packet will be accepted 2. DROP - packet will be dropped <p>For the target, you can also use a user-defined chain. By providing a kernel module or iptables extension, you can have additional targets. The default extension in the iptables distribution are:</p> <ol style="list-style-type: none"> 1. LOG - kernel logging of matching packets; the parameters are: <ul style="list-style-type: none"> --log-level (log level) --log-prefix (the string up to 29 characters, which will show at the beginning of the message) 2. REJECT - the same as DROP, but sender is sent the ICMP port unreachable error message. In some cases the message is not sent - RFC 1122, the parameters are: <ul style="list-style-type: none"> --reject-with (you can alter the replay packet used) <p>There are also two special built-in targets:</p> <ol style="list-style-type: none"> 1. RETURN - for a built-in chain, the policy of the chain is executed; for a user-defined chain, the traversal continues at the previous chain, just after the rule which jumped to this chain 2. QUEUE - the packet is queued to the userspace processing

For example, if you want to create a rule for denying the ICMP protocol packets, which are used when you execute the ping command, for a specific IP address you will do this by executing the command:

```
# /usr/sbin/iptables -A input -s IP_address -p icmp -j DROP
```

If you omit the protocol definition, *all* packets will be denied. So for example, to block access to your machine from network 172.168.1.0 with subnet mask 255.255.255.0, execute the following command:

```
# /usr/sbin/iptables -A input -s 172.168.1.0/255.255.255.0 -j DROP
```

Or you can use:

```
# /usr/sbin/iptables -A input -s 172.168.1.0/24 -j DROP
```

As you can see, the subnet mask can be specified with the number of used bits for that mask.

To disallow *any* traffic from your server to network 172.168.1.0 with subnet mask 255.255.255.0, use this command:

```
# /usr/sbin/iptables -A output -d 172.168.1.0/24 -j DROP
```

Here we used the `-d` parameter to specify the destination address.

11.2.8 Using the inversion `!` option

With some parameters, you can use the inversion option `!`, which means that the rule will be applied to everything *except* the parameters specified after `!`. For example, if you want to deny packets that come from all IP addresses except from network 192.168.1.0 with subnet mask 255.255.255.0, you can do this by executing the command:

```
# /usr/sbin/iptables -A input -s ! 192.168.1.0/24 -j DROP
```

Note: The rules you make are not permanent, so next time you restart the server they will be lost.

11.2.9 Making the rules permanent

To make rules permanent, you have to create the script with the IPTables commands and integrate this script into your boot process. Using the example from 11.2.4, “The first IP Tables rules” on page 253, we created a script similar to that shown in Example 11-14:

Example 11-14 Script for setting up rules

```
#!/bin/sh
/usr/sbin/iptables -N httpallow
/usr/sbin/iptables -A httpallow -s 9.12.6.133 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
/usr/sbin/iptables -A httpallow -j DROP
/usr/sbin/iptables -A INPUT -j httpallow
/usr/sbin/iptables -A FORWARD -j httpallow
```

In our example we named this script `/etc/init.d/filters`. Because the default run level for our system is 3, we positioned this script to execute at the end of the run level 3 with the following command:

```
# ln -s /etc/init.d/rc3.d/S99filters /etc/init.d/filters
```

After rebooting, we verified that the rules were loaded by using the following command:

```
# /usr/sbin/iptables -L
```


Example 11-15 shows our output:

Example 11-15 Current rules

```
# /usr/sbin/iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination
httpallow all  -- anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination
httpallow all  -- anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination

Chain httpallow (2 references)
target    prot opt source                destination
ACCEPT   all  -- tot67.itso.ibm.com  anywhere
ACCEPT   tcp  -- anywhere             anywhere             state NEW tcp
dpt:http
ACCEPT   all  -- anywhere             anywhere             state
RELATED,ESTABLISHED
DROP     all  -- anywhere             anywhere
```

11.2.10 Sample packet filtering configuration for ISP/ASP

In this section we show how to create packet filtering rules to protect the ISP/ASP environment. For this environment, we assume that the following services will be offered on the servers:

1. Web service - HTTP, HTTPS
2. FTP service - FTP
3. SMTP service - SMTP
4. POP3 service - POP3
5. Secure shell service - SSH

For the rules for each service, we follow the approach described in 11.2.4, “The first IP Tables rules” on page 253. For each type of service, we will allow NEW TCP and UDP packets, and packets from ESTABLISHED and RELATED connections.

In our example, we will allow all packets from the administration computer with IP address 9.12.6.133; all other packets will be dropped. Example 11-16 on page 262 shows the script we used to set these rules; we named this script `/etc/init.d/filters`.

Example 11-16 ISP/ASP packet filtering security

```
#!/bin/sh
/usr/sbin/iptables -N httpallow
/usr/sbin/iptables -A httpallow -s 9.12.6.133 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport https -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport https -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ftp-data -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport ftp-data -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ftp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport smtp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport smtp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport pop3 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport pop3 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ssh -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport ssh -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport domain -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport domain -j ACCEPT
/usr/sbin/iptables -A httpallow -j DROP
/usr/sbin/iptables -A INPUT -j httpallow
/usr/sbin/iptables -A FORWARD -j httpallow
```

Note: This script should be started each time the server is restarted, as described in 11.2.9, “Making the rules permanent” on page 260.

We also created a script for deleting all rules from the `/etc/init.d/filters` script. This script is called `/etc/init.d/filters-down` and is shown in Example 11-17.

Example 11-17 Script for deleting rules

```
#!/bin/sh
/usr/sbin/iptables -F httpallow
/usr/sbin/iptables -F INPUT
/usr/sbin/iptables -F FORWARD
/usr/sbin/iptables -X httpallow
```

To delete *all* rules, execute the following command:

```
# /etc/init.d/filters-down
```

To re-enable the rules, execute the following command:

```
# /etc/init.d/filters
```

Tip: If for some reason your network is not working after executing the script `/etc/init.d/filters-down`, you should execute the following commands:

```
# /etc/init.d/network restart
# /etc/init.d/route restart
```

11.2.11 Using IPTables for NAT

In 4.3.5, “Network Address Translation (NAT)” on page 88, we introduced the concept of NAT. In this section we discuss the types of NAT and how they are set up using IPtables.

To manipulate NAT chains we use IP Tables, the same tool used for packet filtering—but instead of using the default table “filter”, we use table “nat”.

Note: In the Linux 2.2 kernel, different tools were used to set up filtering and to set up NAT, so this is an improvement.

Figure 11-6 shows where NAT takes place in the packet’s journey through the router/firewall.

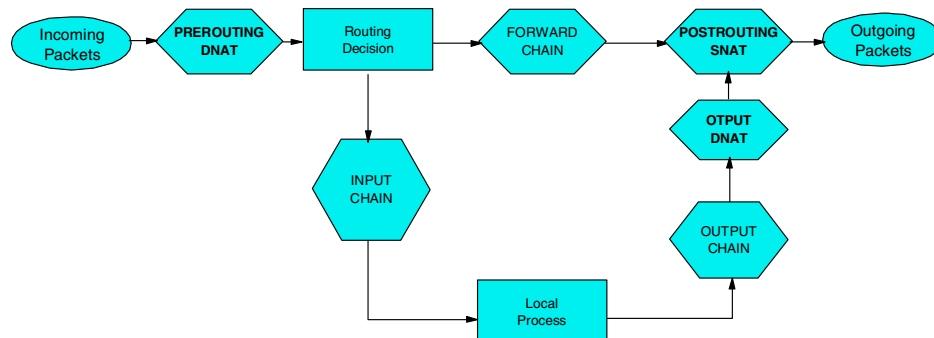


Figure 11-6 NAT role in IP packet travel

Basically, NAT is implemented using the same principle as packet filtering, using three built-in chains to control address translation. As you can see in Figure 11-6 those chains are:

1. PREROUTING - for DNAT, when packets first come in
2. POSTROUTING - for SNAT, when packets leave
3. OUTPUT - for DNAT of locally generated packets

Source NAT

Source NAT is specified with rule `-j SNAT` and the `--to-source` option, which specifies the IP address, a range of the IP addresses, and optional port of range of ports. You can also use the `-o` (outgoing interface) option to specify that the rule only applies to traffic on a particular interface.

For example, to change the source address of your packet to 172.168.2.1, execute the command:

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 172.168.2.1
```

To change the source address to 172.168.2.1, ports 1-1023, use this command:

```
# iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT \
--to-source 172.168.2.1:1-1023
```

Masquerading

See 4.3.5, “Network Address Translation (NAT)” on page 88 for a discussion of masquerading.

Masquerading is specified using rule `-j MASQUERADE`. For example, to masquerade everything leaving our router on `eth0`, we used:

```
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Destination NAT

Destination NAT is specified with the rule `-j DNAT` and the `--to-destination` option, which specifies the IP address, a range of the IP addresses, and optional port of range of ports. You can also use the `-i` (incoming interface) option to specify that the rule is to apply only to a particular interface.

To alter the destination of locally generated packets, use the `OUTPUT` chain.

For example, to change the destination address to 192.168.10.1, execute the following command:

```
# iptables -t nat -A PREROUTING -i eth0 -j DNAT \
--to-destination 192.168.10.1
```

To change the destination address of Web traffic to 192.168.10.1, port 8080, use this command:

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT \
--to-source 192.168.2.1:8080
```

Redirection

This is a specialized version of DNAT, a convenient equivalent of doing DNAT to the address of the incoming interface.

For example, to send the incoming port 80 Web traffic to our squid (transparent) proxy, we used the following command:

```
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 \  
-j REDIRECT --to-port 3128
```

Special protocols

Some protocols do not work well with NAT, so for each of these protocols we need two extensions—one for connection tracking of the protocol, and one for the actual NAT.

Inside the netfilter distribution, there are currently modules for FTP:

1. ip_conntrack_ftp.o
2. ip_nat_ftp.o

If you plan to use NAT for FTP transfers, insert the following modules as shown:

```
# insmod ip_conntrack_ftp  
# insmod ip_table_nat  
# insmod ip_nat_ftp
```

Attention: If you are doing source NAT, you must ensure that the routing is set up correctly. That means if you change the source address so it is different from your external interface (for example, if you use unused IP addresses in your subnet), you need to tell your router to respond to ARP requests for that address as well. This can be done by creating an IP alias:

```
# ip address add IP_address dev eth0
```

11.2.12 Examples for using NAT in the enterprise and ISP/ASP

In this section, we show you how to use NAT in the enterprise and in ISP/ASP environments.

Changing the source address

In 11.2.3, “How to permanently enable IP forwarding” on page 251, we showed how to add a special route to the 10.0.0.0/255.255.0.0 subnet, because some of the packets coming from our server had the source address from this subnet.

However, instead of providing this special route, we can use the SNAT translation on the servers:

```
# iptables -t nat -A POSTROUTING -o iucv0 -j SNAT --to-source dummy0_IPaddr
```

This command will always change the source IP address of the packet to the IP_address of the DUMMY0 interface, which in our example is used as the server's external interface. Because the routers/firewalls are already aware of the routes to this external address, we do not need to provide an additional route to the 10.0.0.0/255.255.0.0 subnet.

Providing access to the Internet from an intranet

In this example, we show how to provide transparent access for an internal computer on the private subnet via the VM Linux router connected to the Internet with the public IP address. The sample configuration is shown in Figure 11-7.

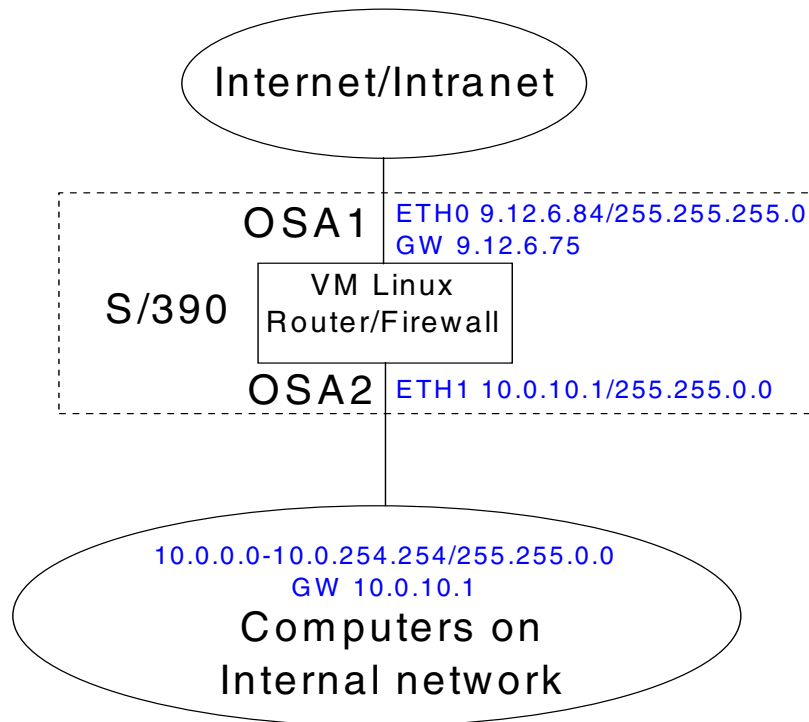


Figure 11-7 Internet access for local LAN

Without any settings, the packet traveling from the internal network via the VM Linux router/firewall will reach its Internet destination. But the source address of this package is from our *internal* subnet—and the server we are talking to does not know how to send the packet back.

To resolve this, we have to enable the source network address translation (SNAT) for each packet going out of the router. This can be done with the following command:

```
# iptables -t nat -A POSTROUTING -s 10.0.0.0/255.255.0.0 -o eth0 \  
-j SNAT --to-source 9.12.6.84
```

With this rule, we are configuring the kernel to change the source address of each packet coming from the 10.0.0.0/255.255.0.0 subnet to the IP address of our external interface 9.12.6.84.

When the packet from the internal subnet reaches the server on the Internet, this server will respond to the router; and when the packet comes back to the router, it will send that packet back to the computer on the internal subnet, with the destination address of this computer.

Tip: If you plan to also provide FTP access to the Internet, insert the following two modules into the kernel:

```
# insmod ip_conntrack_ftp  
# insmod ip_nat_ftp
```

Port forwarding

If you use the DeMilitarized Zone (DMZ) approach for your servers, this can be done using port forwarding. In the DMZ setup, you separate your Web/mail server from the server that is connected to the router/firewall over a private subnet.

This server's private subnet is separated from the local subnet used for computers accessing the Internet. The example of such a setup is shown in Figure 11-8 on page 268.

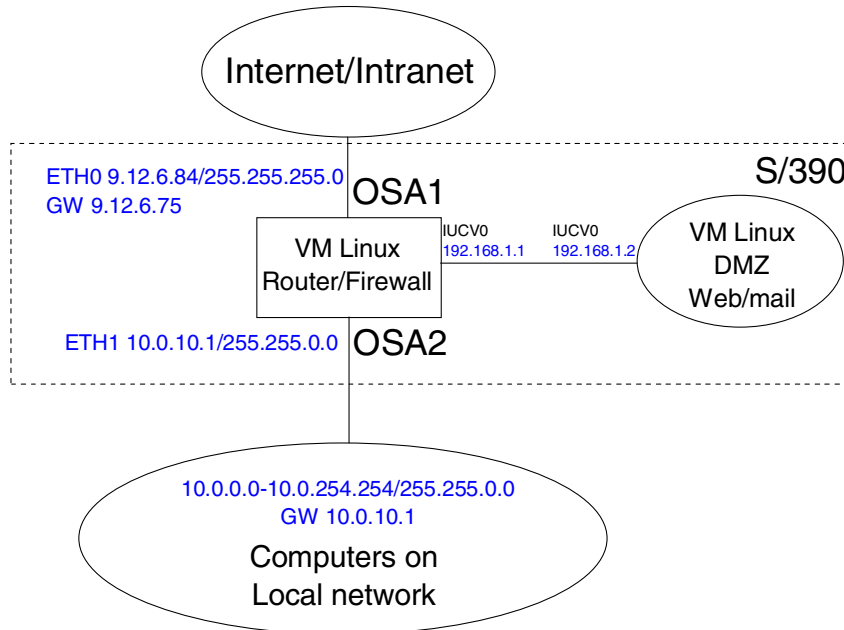


Figure 11-8 Port forwarding

In this scenario, the HTTP requests, for example, are coming to our router, which means that our www name is associated with the router external interface in the DNS. But because we do not have the Web server running on the router, we forward those packets to the Web server, which is in the DMZ. You can achieve this with the following command:

```
# iptables -t nat -A PREROUTING -p tcp -d 9.12.6.84 --dport 80 \
-j DNAT --to-destination 192.168.1.2:80
```

As you can see, we forward all TCP packets coming to address 9.12.6.84 port 80 to the address 192.168.1.2 port 80. You can also do port forwarding for other services.

11.2.13 Additional information

You can find more information on the official Linux IP Tables on the Linux Documentation Project home page:

<http://www.linuxdoc.org>



Backup using Amanda

In this chapter we discuss the Advanced Maryland Automatic Network Disk Archiver, known as Amanda.

Since Amanda has not previously been written about specifically for the Linux for zSeries and S/390 environment, we provide basic information about its operation and use in this environment. The Amanda package is included in the SuSE distribution.

For more information about general Amanda usage, refer to the Amanda Web page:

<http://www.amanda.org>

Along with useful information, this site contains page links to Amanda archives and a number of mailing lists on Amanda (you can also join the mailing lists from here).

12.1 About Amanda

Amanda is an open source backup scheduler, originally developed at the University of Maryland for scheduling the backup of computing facilities there.

Amanda uses a client-server arrangement to facilitate the backup of network-attached servers. Using Amanda, it is possible to have a single tape-equipped server backing up an entire network of servers and desktops.

Note: Like many other Open Source Software projects, Amanda comes with no warranty, has no formal support, and is developed in people's spare time.

This consideration should be kept in mind when choosing a backup strategy.

12.1.1 How Amanda works

Backups are scheduled on one or more servers equipped with offline storage devices such as tape drives. At the scheduled time, the Amanda server contacts the client machine to be backed up, retrieving data over the network and writing it to tape. The data from the client can be stored in a staging area on disk, which improves the performance of the tape writing process (or provides a fallback, in case of tape problems).

Amanda can perform compression of the data being backed up, using standard Linux compression utilities (**gzip**, **bzip**). If network utilization is high, the compression can be done on the client to reduce the network load and potentially reduce backup times. This also lightens the load on the backup server, which may be processing many simultaneous backups.

Figure 12-1 shows Amanda's client-server architecture.

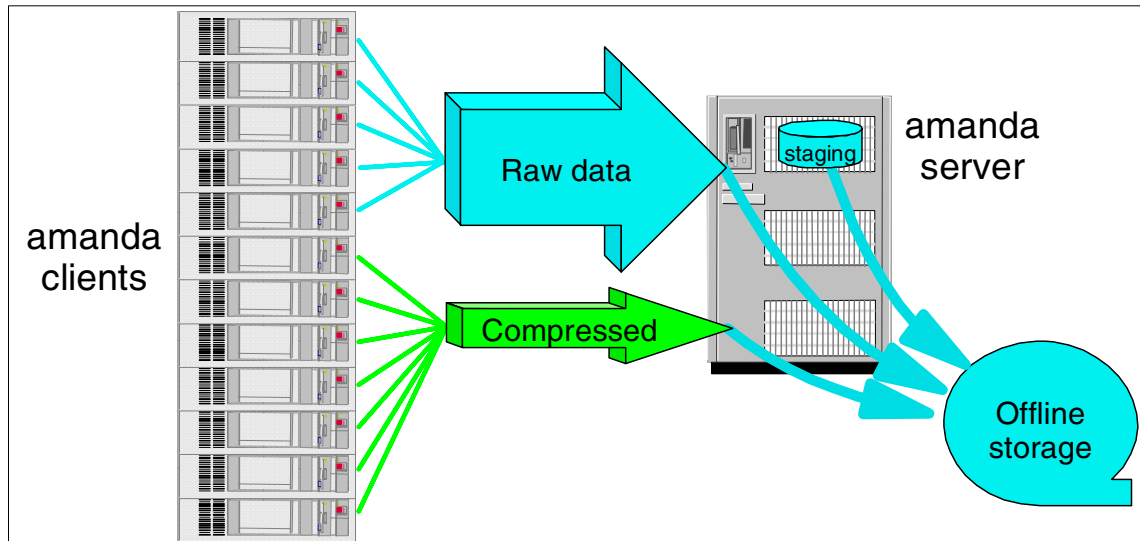


Figure 12-1 Amanda client-server architecture

Amanda uses its own authentication and access protocol, and it can also use Kerberos for authentication. It can be used to back up SMB/CIFS servers directly, which provides an alternative to running the **amandad** process on the clients¹.

Configuring an Amanda server involves five tasks:

1. Identifying the clients, and the devices on those clients, to be backed up
2. Establishing access controls to allow the server to read the clients' data
3. Configuring the disk staging areas on the server
4. Configuring the tapes and tape schedule
5. Setting up and activating the backup schedule

The backup schedule is usually invoked by entries in the `/etc/crontab` file to invoke the **amdump** program at the correct intervals. **amdump** initiates the connection to the client machine to read the data and output it to the staging disk. When this is done, **amdump** calls standard Linux tape management programs to write the data to tape. Amanda also provides support programs to recover from errors caused by tapes filling, incorrect tapes being loaded, and other possible failures.

Naturally, there is also an **amrestore** program to facilitate recovery of data from a backup.

¹ For Linux servers, this only makes sense if Samba was being set up on the server anyway, since it is more difficult to set up Samba than **amandad**.

12.2 Using Amanda in a penguin colony

Use of Amanda can be helpful in a penguin colony because of its low overhead and native Linux operation.

Note: Amanda is a backup scheduler, and does not actually perform the backups itself. The Amanda programs launch utilities like tar, dump and smbtar to perform the backup.

12.2.1 Planning for Amanda

The Amanda configuration process involves firstly determining the backups to be done, and creating configuration directories for these. The backup is then referred to by that directory name in all Amanda commands.

Important: Amanda is a file-level backup system. This means, for example, that it is not aware of the internal structure of databases (refer to “Complex application backup” on page 129 for more information on this).

At the time of writing, there is no plan to extend Amanda to provide awareness of file internals.

Amanda differs from other backup utilities in regard to the backup cycle. Other systems have a set process for the time that a full backup is done in relation to incremental backups (such as full backups on the weekend, and incremental backups overnight during the week).

Amanda does not work this way. It will switch between *level-0* (full) and *level-1* (incremental) backups during the cycle specified in the `amanda.conf` file, dependent upon the number of tapes available, the time since the last full backup, and so on. It will make sure that at least one level-0 backup is available at all times, and that a backup image is not overwritten if it would be required to form part of a backup.

Tip: The Amanda FAQ-O-Matic, available at the following site, contains information as to how a “traditional” backup cycle can be set up.

<http://www.amanda.org/cgi-bin/fom?>

12.2.2 Configuring Amanda

Amanda is configured using two major configuration files in the backup set configuration directory:

- ▶ amanda.conf
- ▶ disklist

amanda.conf

The file amanda.conf contains the following data for each backup:

- ▶ Specifications of tape drives to be used for this backup set
- ▶ Maximum network bandwidth to be used for this backup
- ▶ Definition of the backup cycle (number of tapes, duration, backup frequency)
- ▶ Other attributes of the backup set.

Our sample amanda.conf file is shown in Example 12-1.

Tip: The amanda.conf file provided with the Amanda product (in a backup set called “example”) contains a great deal of information about configuring Amanda backups.

Example 12-1 Example amanda.conf file

```
#
# amanda.conf - sample Amanda configuration file

org "ITSOArchive"      # your organization name for reports
mailto "amanda"        # space separated list of operators at your site
dumpuser "amanda"     # the user to run dumps under
inparallel 2           # maximum dumpers that will run in parallel (max 63)
netusage 800 Kbps      # maximum net bandwidth for Amanda, in KB per sec
dumpcycle 4 weeks      # the number of days in the normal dump cycle
runspcycle 20          # the number of amdump runs in dumpcycle days
tapecycle 25 tapes     # the number of tapes in rotation
bumpsize 20 Mb         # minimum savings (threshold) to bump level 1 -> 2
bumpdays 1           # minimum days at each level
bumpmult 4             # threshold = bumpsize * bumpmult^(level-1)
etimeout 300           # number of seconds per filesystem for estimates.
dtimeout 1800          # number of idle seconds before a dump is aborted.
ctimeout 30            # max. number of seconds amcheck waits for client host
tapebufs 20            # tells taper how many 32k buffers to allocate.
runtapes 1             # number of tapes to be used in a single run of amdump
tapedev "/dev/ntibm0" # the no-rewind tape device to be used
rawtapedev "/dev/ntibm0" # the raw device to be used (ftape only)
tapetype IBM-3480-B40 # what kind of tape it is (see tapetypes below)
labelstr "^ITS0[0-9][0-9]*$" # label constraint regex: all tapes must match

# Specify holding disks.
holdingdisk hd1 {
    comment "main holding disk"
    directory "/dumps/amanda"# where the holding disk is
```

```

    use 290 Mb          # how much space can we use on it
    chunksize 1Gb      # size of chunk if you want big dump to be
                      # dumped on multiple files on holding disks
}

infofile "/var/lib/amanda/ITS0Archive/curinfo" # database DIRECTORY
logdir   "/var/lib/amanda/ITS0Archive"        # log directory
indexdir "/var/lib/amanda/ITS0Archive/index"  # index directory
# tapelist is stored, by default, in the directory that contains amanda.conf

define tapetype IBM-3490E-B40 {
    comment "IBM 3490E-B40"
    length 913152 kbytes      # these numbers generated by the
    filemark 32 kbytes       # Amanda tapetype program
    speed 2439 kps           # (not supplied with SuSE)
}

# dumptypes
define dumptype global {
    comment "Global definitions"
    index yes
}
define dumptype always-full {
    global
    comment "Full dump of this filesystem always"
    compress none
    priority high
    dumpcycle 0
}
define dumptype root-tar {
    global
    program "GNUTAR"
    comment "root partitions dumped with tar"
    compress none
    index
    exclude list "/usr/local/lib/amanda/exclude.gtar"
    priority low
}
define dumptype user-tar {
    root-tar
    comment "user partitions dumped with tar"
    priority medium
}
define dumptype high-tar {
    root-tar
    comment "partitions dumped with tar"
    priority high
}
define dumptype comp-root-tar {

```

```

    root-tar
    comment "Root partitions with compression"
    compress client fast
}
define dumptype comp-user-tar {
    user-tar
    compress client fast
}
define dumptype holding-disk {
    global
    comment "The master-host holding disk itself"
    holdingdisk no # do not use the holding disk
    priority medium
}
define dumptype comp-user {
    global
    comment "Non-root partitions on reasonably fast machines"
    compress client fast
    priority medium
}
define dumptype nocomp-user {
    comp-user
    comment "Non-root partitions on slow machines"
    compress none
}
define dumptype comp-root {
    global
    comment "Root partitions with compression"
    compress client fast
    priority low
}
define dumptype nocomp-root {
    comp-root
    comment "Root partitions without compression"
    compress none
}
define dumptype comp-high {
    global
    comment "very important partitions on fast machines"
    compress client best
    priority high
}
define dumptype nocomp-high {
    comp-high
    comment "very important partitions on slow machines"
    compress none
}
define dumptype nocomp-test {
    global

```

```

    comment "test dump without compression, no /etc/dumpdates recording"
    compress none
    record no
    priority medium
}
define dumptype comp-test {
    nocomp-test
    comment "test dump with compression, no /etc/dumpdates recording"
    compress client fast
}

# network interfaces
define interface local {
    comment "a local disk"
    use 1000 kbps
}
define interface eth1 {
    comment "100 Mbps ethernet"
    use 800 kbps
}

```

Note: Amanda does not provide a tape definition for the IBM mainframe tape devices supported by the tape390 driver. Therefore, we had to follow instructions contained in the example amanda.conf file in order to create a tapetype entry. To do this, however, we had to obtain the source package for Amanda, because the tapetype program is not supplied with the SuSE binaries of Amanda.

Once the tapetype program was available, we ran it against our 3490E-B40 tape drive to produce the tapetype entry shown in Example 12-1 on page 273.

disklist

The other configuration file you have to create is named disklist, which tells the **amdump** program which disks (or directories) on which hosts to back up.

```

#
# File format is:
#
#      hostname diskdev dumptype [spindle [interface]]
#
# ITS0 machines.
#
vmlinux2      dasdb1      comp-root 1 local
vmlinux2      dasdc1      comp-user 2 local
vmlinux2      //tot12/vjc      smb-user 1 eth1
vmlinux7      dasda1      comp-root 1 eth1
vmlinux7      dasdb1      comp-user 2 eth1

```


As you can see, it is fairly easy to specify the servers and devices to be backed up. The `dump` type (third field) must be given for each entry. This value chooses the type of dump from the `amanda.conf` file.

The `spindle` attribute refers to disk configurations where different physical file systems may exist as partitions of a single physical disk (which is not usually an issue for zSeries). The attribute can be used to increase performance by ensuring that Amanda does not simultaneously back up different file systems that share the same physical disk.

Tip: Unless you specifically want to have backups operating sequentially, specify each file system on a particular host with a different `spindle` value to make sure that you get the maximum simultaneous operation.

Amanda can also back up using SMB, allowing Windows machines to be included in your Amanda backup sets. An example of how this is configured appears in the third line of our example `disklist`.

From the perspective of the Amanda server, the SMB share to be backed up is part of the file system of an Amanda client. On that Amanda client, however, the SMB code in Amanda uses `smbclient` to access the SMB share on the Windows host. Files are retrieved from the source using SMB, then sent using the normal Amanda protocols from the client to the server.

Restriction: Using Amanda to back up Windows shares does not retain the Access Control List (ACL) information from the Windows file system. If you have complex ACLs in your Windows servers, Amanda is not the best backup solution. It is more suitable for lightweight backups of data directories on desktop computers, for example.

Also, the binaries of the Amanda package as distributed with SuSE do not have SMB support enabled in the `amanda` client. To test this function, we had to rebuild `amanda` from source.

Other files are created in the configuration directory, but are maintained by Amanda. These include `tapelist`, which is a list of the tapes that belong to a particular backup set and is updated by the `amlabel` program.

Tape changer

Amanda can utilize a tape changer if one is installed. It does this using a shell script identified in the `tpchanger` entry in `amanda.conf`. As long as your tape changer provides a program-level interface, Amanda can make use of it. Sample scripts for popular tape changers are supplied with Amanda.

Without the `tpchanger` parameter set, Amanda automatically switches off any multitape capability. So in order to use the automatic tape loader (ATL) on our 3490E-B40 (which automatically loads the next tape in the rack when the current tape is ejected), we had to either find a suitable script, or write our own.

We used the `chg-multi` script provided with Amanda to drive our ATL. The script provides enough basic function to support our autoloader, but also can be used as a template for writing your own scripts. The `chg-multi` script is generic, which saved us from having to write specific commands in the script to drive our ATL.

Amanda drives the tape changer during a backup process. For example, since it knows all of the labelled tapes in a backup set and keeps track of which tape can be used next, it can skip through the tapes in the rack until the required tape is loaded. It can also load another tape if the backup requires it.

Activating Amanda

The following line needs to be added (or un-commented) to your `/etc/inetd.conf` configuration file to enable the Amanda client.

```
amanda dgram udp wait amanda /usr/lib/amanda/amandad amandad
```

In this example, `/usr/lib/amanda/amandad` is the path to the `amandad` executable. This line was already present on our SuSE installation (with the `amanda` package installed), and simply had to be un-commented.

Important: Any time the `inetd` configuration is changed, you must signal `inetd` to initialize. You can use the following command:

```
killall -HUP inetd
```

For the Amanda server, two more lines must be added to the `inetd` configuration to support the index service. Again, these lines were already in the `/etc/inetd.conf` file on our system and just had to be un-commented.

```
amandaidx stream tcp nowait root /usr/lib/amanda/amindexd amindexd  
amidxtape stream tcp nowait root /usr/lib/amanda/amidxtaped amidxtaped
```

12.2.3 Backing up with Amanda

Prior to making any backups, you must plan and set up your `amanda.conf` and `disklist` files. This is because all operations in Amanda are performed with respect to the backup set being used. All of the `am` commands require the backup set name (the configuration directory name) as a parameter.

Tip: When creating your configuration, it is a good idea to have your holding area on a separate file system from the data being backed up. Otherwise, staging files will become part of your backup, and your incremental backups for that file system will be huge.

You can also experiment with using the **exclude** parameter in `amanda.conf` to exclude the holding area from being backed up.

Once you've created your configuration, you can then label your tapes using the **amlabel** command. This command creates the label that Amanda uses to identify the tape. Various information is kept here, including the name of the backup set the tape belongs to and the date the tape was used. The **amlabel** command also adds the tape to the `tapelist` file.

Note: You'll need to label all tapes in your backup set prior to using them for backups.

When we ran `amlabel` on our first tape, we received this output.

```
# amlabel normal ITSODaily00
labeling tape in slot 0 (/dev/ntibm0):
rewinding, reading label, not an amanda tape
rewinding, writing label ITSODaily00, checking label, done.
```

Having labelled your tapes, you can now test your configuration using the `amcheck` program. This program will identify any problems with your configuration by doing the steps that Amanda would normally take in preparation for a backup.

Tip: Many Amanda users run `amcheck` *prior to* the backup run in their regular backup process. This is because if `amcheck` detects an error, it is easier to fix the problem and schedule the backup run later—rather than repair a backup that fails during execution.

A sample run of `amcheck` is shown here.

```
# amcheck normal
Amanda Tape Server Host Check
-----
WARNING: holding disk /dumps/amanda: only 294236 KB free
(296960 KB requested)
amcheck-server: slot 0: date X    label ITSODaily00 (first labelstr match)
NOTE: skipping tape-writable test
Tape ITSODaily00 label ok
NOTE: info dir /var/lib/amanda/ITSODaily/curinfo: does not exist
NOTE: it will be created on the next run
```

```
NOTE: index dir /var/lib/amanda/ITSODaily/index: does not exist
Server check took 2.702 seconds
```

```
Amanda Backup Client Hosts Check
```

```
-----
Client check: 2 hosts checked in 0.343 seconds, 0 problems found
```

```
(brought to you by Amanda 2.4.2)
```

In this example, `amcheck` is informing us that we are slightly short of holding disk space. It also did a tape check, and the results are shown ('date X' on an Amanda tape indicates a tape that has been labelled but not used).

The next two messages are indications that we have not done a backup before. Amanda can keep two sets of information about backups:

- ▶ `curinfo`
This is information about the current status of the backup set, including which disklist entries are backed up to what level, and so on.
- ▶ `index`
Optional (you must select it in your `dumptype`), the index keeps track of all files backed up, and is used by the `amrecover` program to ease the task of restoring data.

Amanda will create the relevant directories as required.

Finally, `amcheck` contacts the clients listed in the disklist to verify that they are contactable, and that authorization has been given to the backup server to obtain files from them.

The next step is to test a backup. Normally you would have the command issued from cron, but it is a good idea to run backups manually until you are comfortable with the process. The following command will start the `amdump` program, commencing a run of the "normal" backup set:

```
# amdump normal
```

Tip: The `amdump` program does not execute in the background, by default, so if you want to issue commands in your terminal window while `amdump` is running, you'll need to force `amdump` to the background. Invoke `amdump` as follows:

```
amdump normal &
```

While the backup is running, the `amstatus` command can give you information about the progress of the backup, as shown in Example 12-2:

Example 12-2 Output from amstatus

amstatus normal

Using /var/lib/amanda/ITSODaily/amdump from Mon Jul 30 16:58:38 EDT 2001

```
vmlinux2://tot12/vjc          0 82299k dumping   31872k ( 38.73%) (16:59:15)
vmlinux2:dasdb1              0 [dumps too big, but cannot incremental dump new disk]
vmlinux2:dasdc1              0 14944k finished (16:59:21)
vmlinux7:dasda1              0 181822k dumping  84064k ( 46.23%) (16:58:53)
vmlinux7:dasdb1              0 293784k wait for dumping
```

```
SUMMARY      part      real estimated
              size      size
partition    : 5
estimated    : 5          1976075k
failed       : 1          1397244k      ( 70.71%)
wait for dumping: 1          293784k      ( 14.87%)
dumping to tape : 0          0k           ( 0.00%)
dumping      : 2 115936k 264121k ( 43.90%) ( 5.87%)
dumped       : 1 14944k 20926k ( 71.41%) ( 0.76%)
wait for writing: 0          0k           ( 0.00%) ( 0.00%)
writing to tape : 0          0k           ( 0.00%) ( 0.00%)
failed to tape : 0          0k           ( 0.00%) ( 0.00%)
taped        : 1 14944k 20926k ( 71.41%) ( 0.76%)
all dumpers active
taper idle
network free kps: 2540
holding space : 29932k ( 10.17%)
dumper0 busy  : 0:00:28 (100.00%)
dumper1 busy  : 0:00:28 (100.00%)
taper busy    : 0:00:05 ( 19.82%)
0 dumpers busy : 0:00:00 ( 0.00%)
1 dumper busy  : 0:00:00 ( 0.00%)
2 dumpers busy : 0:00:28 (100.00%)
not-idle: 0:00:20 ( 73.56%)
no-dumpers: 0:00:07 ( 26.44%)
```

In this case, the backup of dasdb1 on vmlinux2 has failed because it is too large for the tape. However, the next time **amdump** was run, the dump of dasdb1 on vmlinux2 was added to the tape. So why did this occur?

Referring to FAQ lists for Amanda, when it has a large number of level 0 backups to do (as would happen for the first backup in a set), it is sometimes unable to plan a backup run that would pick them all up. The next time the backup is run, the file system is correctly backed up.

Restriction: Amanda cannot currently write an image to tape if it must span more than one tape. Each entry in the disklist file (i.e. each disk to be backed up) creates a single image file to be written to tape, and while images for separate disks can be written across tapes in a single run, a single image that is larger than a tape cannot be split across tapes.

If you have large partitions to be backed up, it will be necessary to configure them as separate entries in your disklist file until Amanda supports images spanning tapes.

Once the amdump program is complete, a mail message is sent to the operators given in the amanda.conf file; see Example 12-3:

Example 12-3 Backup completion report

Date: Mon, 30 Jul 2001 17:10:46 -0400
From: Amanda Admin <amanda@vmlinux2.itso.ibm.com>
To: amanda@vmlinux2.itso.ibm.com
Subject: ITSODaily AMANDA MAIL REPORT FOR July 30, 2001

These dumps were to tape ITSODaily00.
The next tape Amanda expects to use is: a new tape.

FAILURE AND STRANGE DUMP SUMMARY:

vmlinux2 dasdb1 lev 0 FAILED [dumps too big, but cannot incremental dump new disk]

STATISTICS:

	Total	Full	Daily
	-----	-----	-----
Estimate Time (hrs:min)	0:00		
Run Time (hrs:min)	0:12		
Dump Time (hrs:min)	0:12	0:12	0:00
Output Size (meg)	509.9	509.9	0.0
Original Size (meg)	1221.6	1221.6	0.0
Avg Compressed Size (%)	41.7	41.7	--
Filesystems Dumped	4	4	0
Avg Dump Rate (k/s)	713.4	713.4	--
Tape Time (hrs:min)	0:03	0:03	0:00
Tape Size (meg)	510.0	510.0	0.0
Tape Used (%)	57.2	57.2	0.0
Filesystems Taped	4	4	0
Avg Tp Write Rate (k/s)	2674.2	2674.2	--

?

NOTES:

planner: Adding new disk vmlinux2:dasdb1.

```

planner: Adding new disk vmlinux2:dasdc1.
planner: Adding new disk vmlinux2://tot12/vjc.
planner: Adding new disk vmlinux7:dasda1.
planner: Adding new disk vmlinux7:dasdb1.
driver: WARNING: /dumps/amanda: 296960 KB requested, but only 294188 KB available.
taper: tape ITSODaily00 kb 522240 fm 4 [OK]

```

?

DUMP SUMMARY:

HOSTNAME	DISK	L	DUMPER STATS				TAPER STATS			
			ORIG-KB	OUT-KB	COMP%	MMM:SS	KB/s	MMM:SS	KB/s	
vmlinux2	//tot12/vjc	0	164614	146048	88.7	4:49	505.8	0:53	2763.8	
vmlinux2	dasdb1	0	FAILED	-----						
vmlinux2	dasdc1	0	44931	14944	33.3	0:22	668.1	0:06	2685.1	
vmlinux7	dasda1	0	393946	144544	36.7	2:44	879.0	0:56	2602.8	
vmlinux7	dasdb1	0	647442	216576	33.5	4:16	844.9	1:21	2664.0	

(brought to you by Amanda version 2.4.2)

Driving the tape changer

The `amtape` program provides the interface to the changer script identified in the `amanda.conf` file. Using `amtape`, you can load the next tape in the rack, load a tape with a particular label, eject a tape, and other operations.

Tip: Refer to the `amtape` man page for more information, and keep in mind that if your ATL is gravity-fed, you'll only be able to move forward through the slots in the changer.

The following example shows some `amtape` commands in use.

```

# amtape normal show
amtape: scanning all 6 slots in tape-changer rack:
slot 0: date 20010730 label ITSODaily00
slot 1: date 20010731 label ITSODaily01
slot 2: date 20010731 label ITSODaily02
slot 3: date 20010801 label ITSODaily03
slot 4: date 20010801 label ITSODaily04
slot 5: date X          label ITSODaily05
# amtape normal reset
amtape: changer is reset, slot 0 is loaded.
# amtape normal current
amtape: scanning current slot in tape-changer rack:
slot 0: date 20010731 label ITSODaily02

```

In this example, an operator checks which tapes are currently loaded in the ATL. Amanda scans each tape and outputs the label information it finds. After this the slots are empty, so the operator reloads the tapes and resets Amanda's status of the changer. Then, the operator rechecks the tape in the current slot.

Important: With this type of tape changer, Amanda does not keep track of when tapes have been changed or moved. The **amtape reset** command is an administrative command that advises Amanda that the status of the ATL has changed and reset to start.

amadmin

The amadmin program provides commands that allow you to control the backup process. A number of options are available, including:

- ▶ Force a full backup of a disklist entry at the next run
- ▶ Mark a tape to be reusable or non-reusable
- ▶ Find which tapes the backups for a particular host or disk are on
- ▶ Display information about the backups for certain hosts or disks

Note: There are many commands available with amadmin. We suggest you refer to the amadmin man page to get more information.

In “Reporting with amadmin” on page 289, we discuss the use of some amadmin commands for use with reporting.

Scheduling your backup

Once your testing has gone smoothly, add an entry to `/etc/crontab` which will start the backup automatically at regular times. An example is shown here:

```
22 2 * * * amanda amdump normal
```

This will instruct **cron** to issue the command **amdump normal** under the user `amanda` every day at 2:22am.

Important: Remember to restart cron after making a change to crontab.

It is also possible to use the **amcheck** command in the schedule, to provide a pre-test for the backup run.

```
22 1 * * * amanda amcheck -m normal
22 2 * * * amanda amdump normal
```


These lines in the crontab will schedule an amcheck prior to the scheduled time of the backup (in this case, at 1:22am, with the backup scheduled for 2:22am). The `-m` switch on the `amcheck` command instructs it to run silently, but to send an e-mail to the backup operators if any problems occur. This allows a problem that would cause the backup to fail (wrong tape loaded, network problem) to be rectified before the start of the backup.

12.2.4 Restoring

The amrecover program is the front-end to the Amanda recovery process, and it is invoked from the system you wish to restore files onto. It works similar to FTP, making the backup set appear like an FTP server.

Note: The amrestore program actually performs the restoration. If you know which backup file on the tape contains your required data, you can invoke amrestore directly. To make the restoration process easier, amrecover uses the backup indexes to feed the correct information to amrestore for you.

Amanda restores files relative to the root of the point the backup was taken from. For example, on our test system vmlinux2, `/dev/dasdb1` is the root file system, and `/dev/dasdc1` is mounted at `/home`. To restore files directly into the directory `/home/tot12/testing`, we would change to the `/home` directory and start amrecover from there. The amrestore program will expand the files into the correct directory.

You can choose to restore files into a different directory, so that you can migrate changes from a backup. For example, if you select `/home/tot12/backup` as the location to restore to, when you restore `/home/tot12/testing`, you'll find the restored files in the directory `/home/tot12/backup/tot12/testing`.

Example: An example of a single file recovery session is shown in “Single file or directory restoration with Amanda” on page 292.

12.2.5 Reporting

Amanda keeps extensive logs of the backup process, and comes with utilities to read the logs and report on attributes of the backup process.

amoverview

The amoverview program produces a summary of the history of the backup set.

```
# amoverview normal
```

	date	07 07 08 08
host	disk	30 31 01 02
vmlinux1	dasdb1	0 1
vmlinux1	dasdc1	0 1
vmlinux2	//tot12/vjc	0 11 10 1
vmlinux2	dasdb1	E 0 11 1
vmlinux2	dasdc1	0 11 01 1
vmlinux3	dasdb1	E 0
vmlinux3	dasdc1	0 1
vmlinux7	dasda1	0 11 11 1
vmlinux7	dasdb1	0 11 11 1

It doesn't look like much data, but there is a great deal of information in this output. The `amoverview` program prints the details of each disk in the backup set, when a backup was performed or attempted, and the backup level taken at that time. Let's look at the line for `vmlinux2:dasdc1`:

- ▶ A level 0 backup was taken on July 30.
- ▶ Two level 1 backups were taken on July 31.
- ▶ A level 0 backup, and then a level 1 backup, were taken on August 1. The level 0 taken on this day have made the previous backups redundant. These prior backups would not be required for a disaster recovery restoration, but may still be used if files from prior to August 1 were required.
- ▶ A level 1 backup was done on August 2.

An E indicates that a backup was attempted, but an error occurred. Both `vmlinux2:dasdb1` and `vmlinux3:dasdb1` experienced errors on their first attempt. A possible reason is that the size of the backup was too large to be taken with the other backups being done at the time. For both of these disks, you can see that the level 0 backup was done on the next run.

The `amoverview` tool gives you an easy way to check that your file systems are being backed up in a timely manner.

amplot

The `amplot` program analyses `amdump` files (produced during every `amdump` run) and produces a graphical analysis of the dump.

Note: amplot uses the gnuplot program, which in turn requires X. The amplot output displays in an X window. We had to download and compile gnuplot in order to use amplot. The gnuplot source can be obtained at:

<http://www.gnuplot.org>

The default configuration for gnuplot installs into /usr/local/bin, but amplot as packaged by SuSE expects gnuplot to be found in /usr/bin. You will need to take this into account if you build gnuplot for use with amplot.

If you do not have an X display, or if you prefer printed output, amplot can generate output as a Postscript file. Refer to the amplot man page for more information.

The graphs produced by amplot show statistics such as job queue length, network bandwidth utilization, holding disk utilization, tape idle time, and number of dumper tasks in use. A sample graph from amplot is shown in Figure 12-2 on page 288.

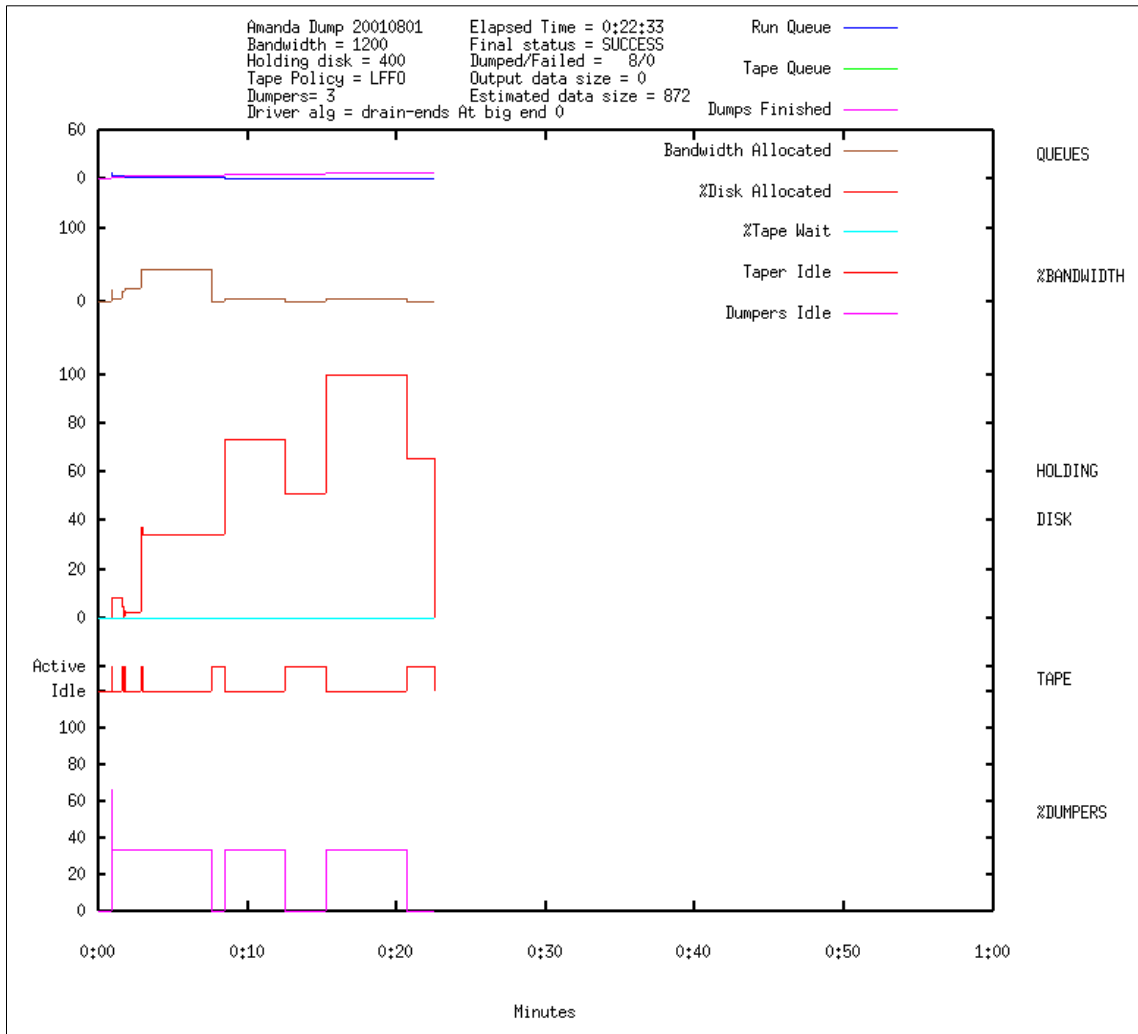


Figure 12-2 Sample *amplot* graph

Network bandwidth utilization is shown as a percentage of the bandwidth allowed in the `amanda.conf` file. On this graph, we see that network bandwidth does not appear to be a bottleneck. The graph also shows that the holding disk is full during the last part of the backup.

The graphs can be used to point out ways to improve performance. In our example, even though we only have three dumpers allocated, they are never all active at once. In fact, except for an instant after the estimates are performed, only one dumper is active at a time for the duration of the backup. This would seem to indicate that Amanda is being prevented from dumping more than one image at a time, which might be a combination of insufficient holding disk space and large backup images.

The e-mail reports sent to the Amanda users at the end of a backup run also contain useful information about the backup process.

Tip: The report sent at the end of an Amanda backup run is generated by the `amreport` program. You can run `amreport` at any time to get the summary of a backup run. Refer to the `amreport` man page for instructions.

Reporting with `amadmin`

Apart from the operational aspects of `amadmin` that we cover in earlier sections, there are reporting sub-commands that are very useful.

One of these sub-commands, `info`, summarizes the data compression and speed statistics of the last three backups for a disk, or for all disks on a host, or for all disks in the backup set. It also shows on which tapes the most recent backup data can be found.

```
# amadmin normal info vmlinux2 dasdc1
```

```
Current info for vmlinux2 dasdc1:
```

```
Stats: dump rates (kps), Full: 711.0, 679.0, -1.0
      Incremental: 16.0, 32.0, 16.0
      compressed size, Full: 33.3%, 33.3%, -100.0%
      Incremental: 43.8%, 43.8%, 43.8%
```

```
Dumps: lev datestamp tape file origK compK secs
      0 20010801 ITSODaily03 1 44931 14944 21
      1 20010802 ITSODaily06 4 73 32 2
```

This example shows information about our `vmlinux2:dasdc1`. The dump rates are shown, as well as the size of the data written to tape, for the last three incremental and full backups written (in our case, since there have been only two full backups, the last column of data for full backup is meaningless).

The `balance` sub-command gives an insight into the way that Amanda schedules full backups within a backup cycle. The purpose of the command is to view how balanced the tape runs have been during the backup cycle, but through interpreting the display, you can gain an understanding of part of Amanda's internal scheduling.

The following shows the output from **amadmin balance**, run early in the backup cycle.

```
# amadmin normal balance

due-date #fs orig KB out KB balance
-----
8/02 Thu 0 0 0 ---
8/03 Fri 0 0 0 ---
8/04 Sat 0 0 0 ---
8/05 Sun 0 0 0 ---
8/06 Mon 2 1041388 361120 -0.3%
8/07 Tue 1 1297449 425824 +17.6%
8/08 Wed 6 2922844 1023456 +182.7%
-----
TOTAL 9 5261681 1810400 362080 (estimated 5 runs per
dumpcycle)
```

This display tells us where full backups are currently due in the schedule. Amanda estimates the amount of data that will be backed up on these dates, based on previous backups, and uses these values to calculate the balance of the cycle.

According to the current plan, 6 out of the 9 file systems in the backup set are due on August 8. This means that the size of the backup on that date will be almost three times the average size of those backups. This creates a huge imbalance in the duration of the backup.

To minimize this, Amanda will promote some of these full backups to earlier in the cycle, in order to balance the workload more evenly throughout the backup cycle.

Important: Amanda will *never* postpone a full backup to balance the cycle.

When Amanda promotes a full backup, you will see messages like this in your backup report:

```
NOTES:
  planner: Full dump of vmlinux3:dasdc1 promoted from 2 days ahead.
  planner: Full dump of vmlinux7:dasda1 promoted from 4 days ahead.
```

In this case, Amanda decided to bring the full backups for these two file systems forward, in order to achieve a balanced backup cycle. Over the course of your backup cycle you may see these messages, especially if your file systems change in size over time.

Note: Refer to the `amadmin` man page for further information about the other sub-commands available.

12.2.6 Disaster recovery using Amanda

Amanda can be deployed in a disaster recovery role. With the aid of a small Linux system that loads using an initial root device (also known as a disaster recovery “bootstrap” system), Amanda can recover full systems up to the last incremental backup.

The system you use as a disaster recovery bootstrap would be like the installation starter system you first used to install Linux (unfortunately, you cannot use one of the installation systems because `amandad` is not present on these systems).

In 10.7, “Linux IPL from NSS” on page 228 we describe a way to build a VM NSS which can be used to IPL Linux images, and this is a useful way to implement a DR bootstrap. We describe other ways to build starter systems elsewhere in Chapter 10.

Important: Remember to install the `amanda` package as part of your disaster recovery bootstrap system, and to add the `amandad` line to `inetd.conf`.

The process would work as follows:

1. IPL the disaster recovery bootstrap image in your Linux guest.
2. Load the network driver and establish network connectivity.
3. Load the DASD driver, correctly mapping the DASDs as configured in the Linux instance to be restored (a standard disk configuration would help here).
4. Reformat the device which will contain your root file system.
5. Run `amrecover` to restore the root file system.
6. Execute step 4 for any “first level” file systems you have on separate devices (e.g. `/usr`, `/home`), and mount these empty file systems at their correct mount points.

Important: If you use LVM, this step will include a restoration of your LVM configuration using `vgcfgrestore` (assuming you backed-up your configuration using `vgcfgbackup`, and that the backup resides on a non-LVM file system!). Otherwise, manually recreate your LVM configuration.

7. Run `amrecover` on the “first level” file systems.

- Repeat steps 6 and 7 for any remaining file systems you have, stepping through the organization of your physical devices as required.

While Amanda can be used in this way to provide disaster recovery capability, it is not the most efficient method of providing full volume backup for Linux instances under VM. A better way would be to have VM perform backups of the minidisks all at once, and use Amanda to provide incremental backups only. Restoration would then involve a VM-level full volume restoration of the Linux system's minidisks, followed by incremental restoration of changed files using Amanda.

12.3 Backup and recovery scenarios

This section illustrates scenarios using the concepts discussed in this chapter.

12.3.1 Single file or directory restoration with Amanda

In Example 12-4, we show a file recovery session using `amrecover`. The file `mrtg_total.pl` has been deleted from the root user's home directory, and we want to restore that file from our Amanda backup.

Example 12-4 A file restore session using `amrecover`

```
vmlinux7:/ # amrecover normal -s vmlinux2 ❶
AMRECOVER Version 2.4.2. Contacting server on vmlinux2 ...
220 vmlinux2 AMANDA index server (2.4.2) ready.
200 Access OK ❷
Setting restore date to today (2001-08-01)
200 Working date set to 2001-08-01.
200 Config set to normal.
200 Dump host set to vmlinux7.
$CWD '/' is on disk 'dasda1' mounted at '/'.
200 Disk set to dasda1.
/
amrecover> history ❸
200- Dump history for config "normal" host "vmlinux7" disk "dasda1"
201- 2001-07-31 1 ITSODaily01 4
201- 2001-07-31 1 ITSODaily02 3
201- 2001-07-30 0 ITSODaily00 2
200 Dump history for config "normal" host "vmlinux7" disk "dasda1"
amrecover> setdate --07-31 ❹
200 Working date set to 2001-07-31.
amrecover> settape vmlinux2:default ❺
Using default tape from server vmlinux2.
amrecover> cd root ❻
/root
```



```

amrecover> ls 7
2001-07-30 .
2001-07-30 .bash_history
2001-07-30 .exrc
2001-07-30 .gnupg/
2001-07-30 .gtkrc-kde
2001-07-30 .kde/
2001-07-30 .kde2/
2001-07-30 .kxmlrpcd
2001-07-30 .mcoprc
2001-07-30 .xinitrc
2001-07-30 KDesktop/
2001-07-30 bin/
2001-07-30 dead.letter
2001-07-30 gd/
2001-07-30 lcs-2.4.5-s390-2.tar.gz
2001-07-30 linux-2.2.19.tar.gz
2001-07-30 linux-2.4.5.tar.gz
2001-07-30 linux/
2001-07-30 mrtg_total.pl
2001-07-30 netsaint/
2001-07-30 sieve
2001-07-30 sieve.c
amrecover> add mrtg_total.pl 8
Added /root/mrtg_total.pl
amrecover> list 9
TAPE ITSODaily00 LEVEL 0 DATE 2001-07-30
      /root/mrtg_total.pl
amrecover> extract 10

```

Extracting files using tape drive /dev/ntibm0 on host vmlinux2.
The following tapes are needed: ITSODaily00

Restoring files into directory /
Continue? [Y/n]: y

Load tape ITSODaily00 now
Continue? [Y/n]: y
restore: ./root: File exists
set owner/mode for '.'? [yn] n
amrecover> quit **11**

```

200 Good bye.
vmlinux7:/ # cd root
vmlinux7:~ # ls -l
total 45252
drwxr-xr-x  11 root   root       4096 Aug  1 03:31 .
drwxr-xr-x  18 root   root       4096 Jul 17 10:09 ..
-rw-----   1 root   root       8405 Jul 28 10:06 .bash_history
-rw-r--r--   1 root   root       1124 Feb 29  2000 .exrc

```

```

drwx--x--x  2 root    root      4096 Jul 17 10:07 .gnupg
-rw-r--r--  1 root    root      1105 Jul 18 04:47 .gtkrc-kde
drwx-----  2 root    root      4096 Jul 18 03:27 .kde
drwx-----  6 root    root      4096 Jul 18 03:58 .kde2
-r-----  1 root    root        21 Jul 18 04:47 .xmlrpcd
-rw-----  1 root    root        31 Jul 18 04:47 .mcporc
-rwxr-xr-x  1 root    root      2186 Apr 11 21:50 .xinitrc
drwx-----  3 root    root      4096 Jul 18 08:21 KDesktop
drwxr-xr-x  2 root    root      4096 Jul 17 10:07 bin
-rw-----  1 root    netsaint 208645 Jul 27 02:42 dead.letter
drwxr-x---  6 root    root      4096 Jul 26 08:49 gd
-rw-r--r--  1 root    root     18690 Jul 21 10:17 lcs-2.4.5-s390-2.tar.gz
drwxr-xr-x 14 1046    netsaint  4096 May 26 11:12 linux
-rw-r--r--  1 root    root    19343412 Jul 20 00:40 linux-2.2.19.tar.gz
-rw-r-----  1 root    root    26534489 Jul 27 08:07 linux-2.4.5.tar.gz
-rwxr-xr-x  1 root    root     27675 Jul 26 08:44 mrtg_total.pl
drwxr-xr-x  4 root    root      4096 Jul 20 08:14 netsaint
-rwxr-xr-x  1 root    root     16481 Jul 18 04:12 sieve
-rw-r-----  1 root    root      1293 Jul 18 04:12 sieve.c

```

1. The `amrecover` program is invoked, specifying the name of the backup set (`normal`) and the Amanda server to be used (`-s vmlinux2`).
2. `amrestore` reports that it successfully contacted the index server on `vmlinux2`. It sets defaults for the `amrecover` session based on current directory, today's date, etc.
3. We request a backup history of the disk, to check that we can get the file we are looking for at the date we need.
4. We want the file as at 31 July, and the backup covers this. The `setdate` command is used to set the point-of-reference for the restore.
5. The `settape` command specifies where the backup tapes (and the tape drive) are located.
6. We can now look through the backup set to locate the file to be restored. First, we change to the directory the file was located.
7. After changing directory, we issue the `ls` command to list the files and directories in the backup. Notice that the file we want to restore, `mrtg_total.pl`, does appear in the list. The date beside the file tells us the most recent version of this file available. Since a level 0 backup was done on July 30, and incremental backups on July 31, it appears that the file did not change between the full backup and the incrementals.
8. Having located the file, we add it to our extraction list using the `add` command.
9. Using the `list` command, we can check the details of the recovery we are about to do. `amrestore` tells us which tape it will be using, and the path to the file being restored.

10. The **extract** command commences the restoration. `amrecover` prompts us for information to complete the restore, including when to load the tape. Since we are recovering into our existing directory (`/root`), the attempt to create the directory fails (`restore: ./root: File exists`), and this is normal. Again, since the directory already exists, we do not need to change permissions.
11. The file recovery is complete, and we can exit `amrecover` and check that the file is correct.



System monitoring

In this chapter, we review methods by which an enterprise running Linux guest machines under VM can record the computing resources consumed by those guest systems. This information could then form the basis of a charge back system.

We also discuss how to monitor the availability of Linux guest machines. This includes system availability, response times and the availability of services such as DNS, mail servers and Web servers.

13.1 Why measure resource consumption

In the context of this redbook, there are essentially two reasons to measure resources consumed in a computing environment. Firstly, a service provider (whether an ASP, ISP or traditional enterprise) will often want to bill its users for their use of computing resources such as CPU time, disk space and network I/O bandwidth.

Secondly, there is a need to ensure that Service Level Agreements are being adequately met.

13.2 Charge back method

The charge back or billing methodology you choose will depend on the type of services you're providing to your customers. For example, the billing requirements of an ASP or ISP will probably be quite different from those used by an enterprise using Linux on VM as a server consolidation platform.

13.2.1 Service Provider accounting and billing

For an ASP or ISP, services and rates are the basis for a charge back system. Each service provided by the service provider has a rate (or fee) that falls into one of two categories: sign-up fees or usage fees.

The *sign-up fee* is a one-time flat fee charged to set up the user account for the service. The *usage fee* is a predetermined, recurring charge that occurs during each billing cycle. The usage criteria may be based on several models, ranging from a simple scheme where a flat fee is charged for the use of the service, to sophisticated schemes where the exact usage of each resource (CPU, memory, disk, network bandwidth etc.) is metered and billed to the user. Promotions and discounts are frequently offered to encourage new users to sign up and current users to use more services.

At the end of the billing cycle, the billing software computes the total charge for each user and mails an invoice or debits a credit card account, depending on the user's payment model.

There are a number of open source ISP billing and account administration packages available for Linux. One example is Freeside, which is available at:

<http://www.sisd.com/freeside>

The service provider must have an accurate way of billing the customer for such things as application usage and system resource usage. *Application usage* is easily tracked by methods as simple as using timestamp checkpoints embedded in the application programs. With this method, the customer signs on to the application, and the time is recorded. When the customer signs off, the time is once again recorded. To generate a bill, the start and end times are used to calculate the charge for that particular user's session.

Billing for *system resource usage* is more complex, as it requires a greater level of measurement and recording. The first part of this chapter focuses on system resource measurement.

13.2.2 Enterprise accounting and billing

Enterprises are using Linux under VM as a server consolidation platform. For example, you can consolidate many disparate file and print servers or infrastructure servers (such as DNS, firewall, e-mail) onto a single S/390 or zSeries machine.

These functions may be purely internal within an organization and as such, an ASP or ISP billing model would probably not apply. However, it's often necessary to charge back individual departments within an organization for their use of computing resources. This requirement has existed since the earliest days of computing, when precious computing resource had to be shared among many groups.

13.3 What can we measure

There are many measurement metrics available. However, not all of these are necessarily useful for charge back purposes. For that reason, we'll focus on CPU consumption, DASD utilization and network bandwidth usage. Given that in the context of this redbook we ran multiple Linux guest systems under VM, we'll use a combination of VM and Linux tools to derive the resource measurements.

13.4 CPU time accounting

In the following sections, we detail the various aspects of CPU time accounting, including how to set up virtual machines for accounting purposes, and how to set up Linux process accounting.

13.4.1 VM accounting

The VM operating system has the capability to generate accounting records that can be used for charge back.

The VM Control Program (CP) creates and records accounting records when particular system events occur. Once accounting is running, CP creates an accounting record whenever one of the following events occurs:

- ▶ A virtual machine logs off, or detaches a virtual processor.
- ▶ A user detaches a dedicated device.
- ▶ A user releases temporary disk space.
- ▶ A virtual machine issues a DIAGNOSE code X'4C'
- ▶ A SNA/CCS terminal session ends.
- ▶ The system checkpoints (during shutdown, for example).
- ▶ You enter the **ACNT** command.

When one of these events occurs, CP creates and stores an accounting record that describes the event. Then CP notifies the accounting virtual machine of the new record. Accounting records remain in storage until the accounting virtual machine retrieves them. The default limit for accounting is 20 records. If the number of records in storage reaches that number, CP notifies the primary system operator. The buildup of records in storage indicates that retrieval is not active. You can change the limit with the RECORDING command.

13.4.2 Setting up virtual machines for accounting

Note: If VM accounting has not been enabled at your installation, this section will show you how to set up this process. For more detailed information on setting up a virtual machine for accounting, refer to the latest version of the VM Planning and Administration publication.

The VM installation media supplies a sample directory entry for an accounting virtual machine. This entry contains the required IUCV authorization for connecting to the CP accounting system service. Also supplied is a sample system configuration file that defines the user ID for the accounting virtual machine as DISKACNT.

The user ID for the accounting virtual machine is defined as part of the SYSTEM_USERIDS statement in the system configuration file so that it is automatically logged on by CP at IPL. A sample PROFILE EXEC for the accounting virtual machine is also supplied.

To set up a virtual machine to begin recording accounting information automatically, you must have the proper PROFILE EXEC and user directory set up. The following steps show this procedure:

1. Log on as MAINT.
2. Determine the write password of the accounting virtual machine's 191 disk.

Note: The accounting virtual machine has been specified in either the SYSTEM_USERIDS ACCOUNT1 or ACCOUNT2 system configuration file statement, or the SYSACNT macroinstruction. Before linking to the accounting virtual machine's 191 disk, find out its write password by examining its user directory entry.

If the accounting virtual machine's 191 disk does not have a write password, you must supply one and update the directory.

Verify that the directory entry for this virtual machine contains the required IUCV authorization for connecting to the CP accounting system service (for example, IUCV *ACCOUNT).

3. Link to the accounting virtual machine's 191 disk by entering:

```
link to diskacnt 191 as 391 wr
```

When CP responds with ENTER WRITE PASSWORD: enter, for example, the following:

```
wpass
```

where *wpass* is the write password of the accounting virtual machine. The accounting virtual machine's 191 disk is now your 391 disk.

4. Access the 391 disk by entering:

```
access 391 x
```

If you receive a message that says X' 391 'DEVICE ERROR, you must format the 391 disk by entering:

```
format 391 x
```

CMS responds as follows:

```
FORMAT WILL ERASE ALL FILES ON DISK X (391).DO YOU WISH TO CONTINUE?  
(YES|NO).
```

Answer *yes* and when CP responds with ENTER DISK LABEL, enter:

```
acct
```

You can use any 1- to 6-character label name.

5. Copy the file named DVM PROFILE from MAINT's 193 disk (we have accessed the 193 disk as k) to the 391 disk by entering:

```
copyfile dvm profile k profile exec x
```

Note: The DVM PROFILE is the PROFILE EXEC for the accounting, symptom record recording, and error recording virtual machines. The RETRIEVE utility, which does the IUCV connect to the *ACCOUNT system service, is invoked from this PROFILE EXEC.

6. Release and detach the 391 disk by entering:

```
release x (det
```

7. If the accounting virtual machine is not logged on, use the XAUTOLOG command to log on the accounting virtual machine automatically. To do this for the DISKACNT user ID, enter:

```
xautolog diskacnt
```

8. You can use the CP command QUERY RECORDING to ensure that accounting is active, for example, by entering the following:

```
query recording
```

Example 13-1 shows an example of the output.

Example 13-1 Query recording output

RECORDING	COUNT	LMT	USERID	COMMUNICATION
EREP	ON 00000000	002	EREP	ACTIVE
ACCOUNT	ON 00001155	020	DISKACNT	ACTIVE
SYMPTOM	ON 00000000	002	OPERSYMP	ACTIVE

13.4.3 Virtual machine resource usage - record type 01

There are a number of VM accounting records available, but for Linux guest CPU consumption data, we're primarily interested in record type 01. This record is produced whenever a user logs off or whenever the ACNT command is entered. Among other things the record contains information on the following:

- ▶ User ID (Linux guest name)
- ▶ Number of seconds connected to CP
- ▶ Milliseconds of processor time used, including time for supervisor functions
- ▶ Milliseconds of virtual CPU time used
- ▶ Number of page reads
- ▶ Number of page writes
- ▶ Number of requested virtual I/O starts for non-spoiled I/O

13.4.4 Processing accounting records

Over time, the accounting virtual machine's A disk fills with accounting records. CP sends a message to the primary system operator when the A disk is 75% full, when it is 90% full, and when it is completely full. You can also log on the accounting virtual machine and check the disk yourself. When the disk is full, you must process some of the old records and erase some files to make room for new ones.

The CMS Utility *ACCOUNT* can be used to process accounting records. Since z/VM 4.1, the CMS utilities have been bundled into the base z/VM installation and are no longer a separate, chargeable product.

Note: Refer to *CMS Command and Utility Reference* for complete information about the *ACCOUNT* utility.

Example 13-2 from the *ACCOUNT* command illustrates that we can use VM accounting to gather data on CPU consumption for all Linux guests running under VM.

Example 13-2 Output from the VM ACCOUNT command

VM SYSTEM USAGE OVER THE PERIOD				07/12/01 TO 07/12/01			ALL SHIFTS	
USERID	SESS	CONNECT	RATIO	REAL-CPU	VIRT-CPU	PG READ	PG WRITE	SIO
TUXOMSTR	1	000004:35	00974	0000:00:17	0000:00:10	6082	10874	33682
VMLINUXA	1	000844:50	00001	0596:31:23	0234:57:01	38644	95891	107830
VMLINUXB	1	000702:14	00663	0001:03:31	0000:43:40	237642	235045	49061
VMLINUXC	1	000702:14	00582	0001:12:23	0000:48:30	230075	256174	62389
VMLINUX2	1	000697:30	00309	0002:15:07	0001:54:22	621859	634902	2553131
VMLINUX3	1	000031:43	00638	0000:02:59	0000:01:50	10490	16233	9429
VMLINUX4	1	000702:16	00453	0001:32:53	0001:08:39	60941	80382	240205
VMLINUX5	1	000001:33	*****	0000:00:00	0000:00:00	0	0	397
VMLINUX6	1	000863:18	00562	0001:32:07	0001:12:14	142996	230437	4847862
VMLINUX7	4	000078:35	00605	0000:07:47	0000:05:19	17	3395	70930
VMLINUX8	1	000080:20	*****	0000:00:00	0000:00:00	0	876	26
VMLINUX9	1	000573:23	00548	0001:02:44	0000:45:39	39402	115050	704388
TOTALS	41	020860:28	00033	0615:45:31	0242:59:18	1915393	2265598	10406917

13.4.5 Linux process accounting

Process accounting is the method of recording and summarizing processes executed on an individual Linux guest machine. Process accounting collects metrics such as the elapsed CPU time, average memory use, I/O information, and the name of the user who ran the process. The kernel will log process accounting information after a process terminates.

Note: Depending on the model of implementing Linux servers under VM, it might be sufficient to use VM accounting to record CPU consumption at a guest level rather than recording at a process level with individual Linux guests.

If you do require Linux process accounting, you should first install the `acct` rpm package. If you are running SuSE, this package resides in the `ap1` package directory, with the filename `acct.rpm`.

After installing the rpm, you can edit `/etc/rc.config` to enable accounting at Linux boot time with the following command:

```
# rpm -ivh acct.rpm
```

The parameter to edit in `/etc/rc.config` is named `START_ACCT`. Make sure that it is set as follows:

```
START_ACCT=yes
```

Important: Remember to run the `SuSEconfig` command after you have edited `/etc/rc.config`.

Once the Linux system has been rebooted, process accounting will be started automatically.

The `lastcomm` command can be used to show the last commands that have been executed in a Linux system. The information displayed includes the command name, who ran the command, and the amount of CPU time consumed.

The `sa` command is a tool for summarizing and reporting on logged process accounting data stored in the `acct` file. Refer to the `sa` man page for a complete description of the syntax available.

13.5 Disk space utilization

In the context of this redbook, we are running many Linux guest systems under the VM operating system. As such, each Linux guest will have a number of minidisks. A simple approach to billing customers for the amount of disk space they consume would be to use existing VM utilities to report on DASD space utilization.

If the VM installation uses the User Directory (i.e., not DIRMAINT), then we can use the CP utility `DISKMAP` to provide us with space utilization information for Linux guests.

Example 13-3 Output from DISKMAP

VOLUME	USERID	CUU	DEVTYPE	START	END	SIZE	
				0	0	1	GAP
LIS32A	LINMNT2	208	3390	00001	03338	03338	

From this example we can see that the Linux guest system LINMNT2 has 3338 cylinders of DASD allocated to it.

If the VM installation is using the Directory Maintenance (DIRMAINT) utility, then the systems programmer can issue the command:

```
dirm dirmap
```

This command will generate a report detailing the current DASD utilization on the VM system. Example 13-4 illustrates the output that is generated:

Example 13-4 Output from DIRM DIRMAP

USER	DIRECT	Map of Minidisks			14:38:05	20010713		
Volser	Type	Ownerid	Addr	SysAffin	Start	End	Length	Flags
LIUSR1	3390				0	0		1 Gap
		LI2000	0191	*	1	100	100	
		LI2000	0200	*	101	3100	3000	
		MONWRITE	0203	*	3101	3338	238	
LIUSR2	3390				0	0		1 Gap
		LI2000	0201	*	1	3000	3000	
		MONWRITE	0200	*	3001	3338	338	

13.6 Network bandwidth usage

There are two options for attributing bandwidth consumption to individual Linux guests in a VM environment. You can either use the SNMP server that is provided as part of VM's TCP/IP stack, or you can use SNMP services provided within Linux. In our case, we've chosen to focus on SNMP services within a Linux environment. For detailed information on configuring an SNMP virtual machine under VM's TCP/IP stack, refer to *z/VM TCP/IP Planning and Customization*, SC24-5981.

13.6.1 An introduction to SNMP

Simple Network Management Protocol (SNMP) is an application-layer protocol that facilitates the exchange of management information between network devices. It is part of the TCP/IP protocol suite. There are two standard levels of SNMP: SNMPv1 and SNMPv2. There is a third version of SNMP SNMPv3, but acceptance of this as a standard is still pending.

The two primary components of an SNMP implementation are the SNMP agent and the Network Management Application. It is a client server architecture where the SNMP agent is the server and the SNMP manager is the client.

An *agent* is a software component that resides on a managed device and collects management information. A managed device could be a UPS, a router, a server, or one of a multitude of other device types. In our context, a managed device will be one or more Linux guest machines. The Network Management application can monitor and control devices on which an SNMP agent is running.

The three commands that are most commonly used in SNMP communications are read, write, and trap; they have the following characteristics:

Read	This command is used by the network management application to query SNMP agents for management information.
Write	This command is used by the network management application to modify variables maintained by the SNMP agent.
Trap	This command is used by SNMP agents to send alerts to network management applications when defined thresholds are met, or specific events occur.

The collection of management information that an agent is responsible for is called the Management Information Base (MIB). MIBs are organized hierarchically in a tree structure and are comprised of managed objects. Managed objects are the leaf nodes of the MIB tree.

SNMP is typically used to gauge network performance, find and resolve network problems, and plan for network growth. However, you can also use SNMP to monitor vendor-specific hardware such as the current load on a UPS, the CPU utilization on routers, hubs, and servers, and even disk I/O and free space.

13.6.2 SNMP installation

Most Linux distributions should include some version of SNMP. We chose to use the UCD-SNMP package for this redbook. We started on the Web at:

<http://net-snmp.sourceforge.net/>

Note: Although the RPM package is called `ucdsnmp`, the project has now been renamed to Net-SNMP.

UCD-SNMP includes various SNMP tools: an extensible agent, an SNMP library, tools for requesting or setting information from SNMP agents, tools for generating and handling SNMP traps, a version of the `netstat` command which uses SNMP, and a Tk/Perl MIB browser. You will probably also want to install the `ucd-snmp-utils` package, which contains UCD-SNMP utilities.

Note: The following example illustrates how we installed and configured SNMP using a SuSE system, with UCD-SNMP 4.2.1. The steps may differ if you are running a different Linux distribution, or if you are running a different level of UCD-SNMP.

To install the UCD-SNMP package, enter the following command:

```
# rpm -ivh /suse/cd1/n2/ucdsnmp.rpm
```

Once the package is installed, an example configuration file can be found in `/usr/share/doc/packages/ucdsnmp/EXAMPLE.conf`. Copy the `EXAMPLE.conf` file to the `/etc` directory as follows:

```
# cp /usr/share/doc/packages/ucdsnmp/EXAMPLE.conf /etc/ucdsnmpd.conf
```

13.6.3 SNMP configuration

We now want to configure SNMP for our local environment by editing the `/etc/ucdsnmp.conf` file. Example 13-5 shows the simple modifications we made to the file:

Example 13-5 Changes to `/etc/ucdsnmp.conf`

#	sec.name	source	community
com2sec	local	localhost	localitso
com2sec	mynetwork	9.0.0.0/8	itso

As shown, we set the community name (which is synonymous with a password) to `localitso`, in order to access the SNMP data from our local system. If we had wanted to access this machine's SNMP data from another machine in the network (limited to users with an IP address of `9.x.x.x`), we would've used the password `itso`.

These modifications will be enough to get SNMP working in your environment; however, you should spend some time reviewing the configuration file to ensure you have the correct parameters set for your installation.

Now we edited the file `/etc/rc.d/snmpd`. We wanted to change the startup command so that the SNMP daemon uses our `/etc/ucdsnmpd.conf` as the configuration file.

Therefore, we changed the following line:

```
startproc /usr/sbin/snmpd -f || return=$rc_failed
```

to read:

```
startproc /usr/sbin/snmpd -f/etc/ucdsnmpd.conf || return=$rc_failed
```

Finally, before starting the SNMP services, we edited `/etc/rc.config` by changing the following line:

```
START_SNMPD="no"
```

to read:

```
START_SNMPD="yes"
```

As a result, the SNMP daemon will start automatically from now on when Linux is booted. However, it is not actually running yet.

Note: Always remember to re-run `SuSEconfig` after making any changes to `/etc/rc.config`.

We were now ready to start SNMP services manually by using this command:

```
# rcsnmpd start
```

To test our SNMP implementation, we used the `snmpget` command. This command queries SNMP agents on specified hosts for one or more OID values. The syntax is as follows:

```
snmpget HOST COMMUNITY OID
```

Try the following command and you should get a similar response:

```
# snmpget localhost localitso .1.3.6.1.2.1.1.1.0
system.sysDescr.0 = Linux tux390 2.2.16 #1 SMP Wed Nov 8 10:57:03 GMT 2000 s390
```

The OID `.1.3.6.1.2.1.1.1` maps to the system description. To see all of the available objects in our tree, we used the `snmpwalk` command. This command queries an entire tree, instead of individual OIDs.

The basic syntax is the same as `snmpget` (although the two commands have several different options):

```
# snmpwalk localhost public .1
```


With this command, you “walk” the entire tree of OIDs that are available to you. You can use the `snmpwalk` and `snmpget` commands from a remote Linux host on the network and get the same result.

This is a very basic implementation of SNMP. Included with the example `ucdsnmpd.conf` file are methods for monitoring CPU utilization, disk space, and several other useful examples. With these packages, you are also able to set traps to be sent to a specified host.

13.6.4 Network bandwidth monitoring

There are many tools available to monitor bandwidth consumption. In our case, we focus on just one of those tools, MRTG. This is not an endorsement of that product; rather, we picked one of the many available open source tools in this area merely to demonstrate how easy it is to install and configure.

13.6.5 MRTG

The Multi Router Traffic Grapher (MRTG) is an open source tool that utilizes SNMP to monitor the traffic load on servers, routers, or virtually anything that generates SNMP records. It can be found on the Internet at:

<http://people.ee.ethz.ch/~oetiker/webtools/mrtg/>

It is licensed for use under the terms of the GNU General Public License.

MRTG generates HTML pages containing PNG images which provide a snapshot visual representation of this traffic. MRTG is an excellent example of what you can do with SNMP. MRTG can be used to report on more than just network traffic; in our example, we also report on CPU consumption.

Note: The following example illustrates how we installed and configured MRTG using a SuSE system, using MRTG 2.9.10. The steps may differ if you are running a different Linux distribution or if you are running a different level of MRTG. Also note that we ran with UCD-SNMP 4.2.1.

13.6.6 MRTG installation and customization

If you are using SuSE, you can get the MRTG package from the `n1` packages directory, filename `mrtg.rpm`

To install the MRTG package, enter the following command:

```
# rpm -ivh mrtg.rpm
```

In our example, we created a configuration file to monitor the network traffic on the localhost and provide us with CPU statistics. We first used the **cfgmaker** tool to create the configuration file:

```
# cfgmaker localitso@localhost > /etc/mrtg.conf
```

The **cfgmaker** program will discover the network interfaces that are defined to your Linux guest, and write this information (along with appropriate HTML tags) into the configuration file. In our example, the Linux guest has an OSA-Express Fast Ethernet interface.

Note: We encountered problems when MRTG tried to discover interface information for virtual CTC or IUCV devices. Refer to 13.6.8, “MRTG reporting for Virtual CTC or IUCV devices” on page 314 for a discussion of the extra steps needed to get bandwidth reporting to function using these devices.

Example 13-6 Ethernet interface as defined in /etc/mrtg.conf

```
Target[localhost_3]: 3:localitso@localhost:
SetEnv[localhost_3]: MRTG_INT_IP="9.12.6.73" MRTG_INT_DESCR="eth0"
MaxBytes[localhost_3]: 1250000
Title[localhost_3]: Traffic Analysis for 3 -- vmlinux7
PageTop[localhost_3]: <H1>Traffic Analysis for 3 -- vmlinux7</H1>
<TABLE>
  <TR><TD>System:</TD>      <TD>vmlinux7 Guest Machine</TD></TR>
  <TR><TD>Maintainer:</TD> <TD>CCW <Caroline@javadog.org></TD></TR>
  <TR><TD>Description:</TD><TD>eth0  </TD></TR>
  <TR><TD>ifType:</TD>      <TD>ethernetCsmacd (6)</TD></TR>
  <TR><TD>ifName:</TD>     <TD></TD></TR>
  <TR><TD>Max Speed:</TD>  <TD>1250.0 kBytes/s</TD></TR>
  <TR><TD>Ip:</TD>        <TD>9.12.6.73 (vmlinux7.itso.ibm.com)</TD></TR>
</TABLE>
```

We now needed to edit the newly created `mrtg.conf` file. At the top of the file, we added an entry for the working directory where MRTG will place the HTML and .PNG files. Because we were using Apache as the Web server in our example, we elected to use a subdirectory called `mrtg` off the default Apache `DocumentRoot`. We added the following `WorkDir` entry in the file `/etc/mrtg.conf`:

```
WorkDir: /usr/local/httpd/htdocs/mrtg
```

Before running MRTG, we also decided to add some additional CPU reporting definitions into the `/etc/mrtg.conf` file. This is an example of the extra reporting that can be achieved using SNMP—we are not limited to simply network statistics; instead CPU, memory, disk and many other resource measurements are available.

Example 13-7 CPU reporting definitions in /etc/mrtg.conf

```
LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.cpu]:ssCpuRawUser.0&ssCpuRawIdle.0:localitso@localhost
RouterUptime[localhost.cpu]: localitso@localhost
MaxBytes[localhost.cpu]: 100
Title[localhost.cpu]: CPU LOAD
PageTop[localhost.cpu]: <H1>User CPU Load %</H1>
Unscaled[localhost.cpu]: ymwd
ShortLegend[localhost.cpu]: %
YLegend[localhost.cpu]: CPU Utilization
Legend1[localhost.cpu]: User CPU in % (Load)
Legend2[localhost.cpu]: Idle CPU in % (Load)
Legend3[localhost.cpu]:
Legend4[localhost.cpu]:
LegendI[localhost.cpu]: User
LegendO[localhost.cpu]: Idle
Options[localhost.cpu]: nopercent

LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.usrsys]:ssCpuRawUser.0&ssCpuRawSystem.0:localitso@localhost
RouterUptime[localhost.usrsys]: localitso@localhost
MaxBytes[localhost.usrsys]: 100
Title[localhost.usrsys]: CPU LOAD
PageTop[localhost.usrsys]: <H1>CPU (user and system) Load %</H1>
Unscaled[localhost.usrsys]: ymwd
ShortLegend[localhost.usrsys]: %
YLegend[localhost.usrsys]: CPU Utilization
Legend1[localhost.usrsys]: User CPU in % (Load)
Legend2[localhost.usrsys]: System CPU in % (Load)
Legend3[localhost.usrsys]:
Legend4[localhost.usrsys]:
LegendI[localhost.usrsys]: User
LegendO[localhost.usrsys]: System
Options[localhost.usrsys]: nopercent

LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.cpusum]:ssCpuRawUser.0&ssCpuRawUser.0:localitso@localhost +
ssCpuRawSystem.0&ssCpuRawSystem.0:localitso@localhost /+
ssCpuRawNice.0&ssCpuRawNice.0:localitso@localhost
MaxBytes[localhost.cpusum]: 100
Title[localhost.cpusum]: CPU LOAD
PageTop[localhost.cpusum]: <H1>Active CPU Load %</H1>
Unscaled[localhost.cpusum]: ymwd
ShortLegend[localhost.cpusum]: %
YLegend[localhost.cpusum]: CPU Utilization
Legend1[localhost.cpusum]: Active CPU in % (Load)
Legend2[localhost.cpusum]:
Legend3[localhost.cpusum]:
```

```
Legend4[localhost.cpusum]:  
LegendI[localhost.cpusum]: Active  
Legend0[localhost.cpusum]:  
Options[localhost.cpusum]: nopercnt
```

13.6.7 MRTG reporting

We were now ready to run MRTG. In your case, you should first run such a tool manually, and then, when you're happy with the reporting, you can use cron to automate the recording. The first couple of times MRTG is run, it gives warning messages such as those shown in Example 13-8. These messages are normal and can be safely ignored.

Example 13-8 Warning messages when first running MRTG

```
Rateup WARNING: /usr/bin//rateup could not read the primary log file for tux390.au.ibm.com  
Rateup WARNING: /usr/bin//rateup The backup log file for tux390.au.ibm.com was invalid as well  
Rateup WARNING: /usr/bin//rateup Can't remove tux390.au.ibm.com.old updating log file  
Rateup WARNING: /usr/bin//rateup Can't rename tux390.au.ibm.com.log to tux390.au.ibm.com.old  
updating log file  
Rateup WARNING: /usr/bin//rateup could not read the primary log file for localhost.3  
Rateup WARNING: /usr/bin//rateup The backup log file for localhost.3 was invalid as well  
Rateup WARNING: /usr/bin//rateup Can't remove localhost.3.old updating log file  
Rateup WARNING: /usr/bin//rateup Can't rename localhost.3.log to localhost.3.old updating log  
file
```

MRTG must be run regularly to capture SNMP statistics, with the recommended interval being every 5 minutes. You can update the `/etc/crontab` file to automate the running of MRTG, as shown in Example 13-9:

Example 13-9 An /etc/crontab definition to run MRTG automatically every 5 minutes

```
* /5 * * * * root /usr/bin/mrtg /etc/mrtg.conf > /dev/null 2>&1
```

Using our example, we now saw a number of HTML pages populating the directory `/usr/local/httpd/htdocs/mrtg`.

An example of one of the pages MRTG generates is shown in Figure 13-1 on page 313:

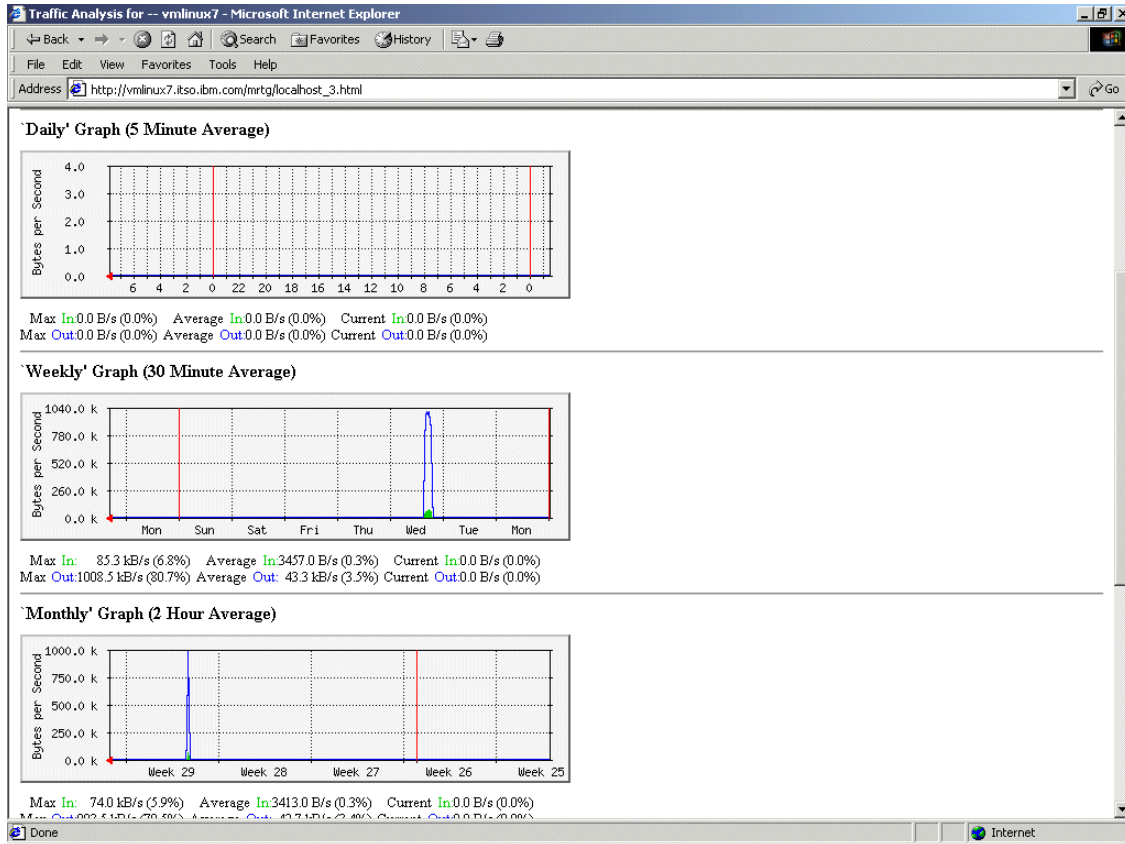


Figure 13-1 MRTG Traffic Analysis page

It would be useful to see all the graphs for traffic analysis, CPU consumption, etc. on a single Web page. The MRTG package includes a utility called **indexmaker** which can be used to create an `index.html` page incorporating all the reports on to a single Web page.

```
# indexmaker /etc/mrtg.conf --output=/usr/local/httpd/htdocs/mrtg/index.html
```

In this example, we ran `indexmaker` against our MRTG configuration file, telling it to write the newly created `index.html` file to the MRTG HTML directory.

An example of the `index.html` page is shown in Figure 13-2 on page 314.

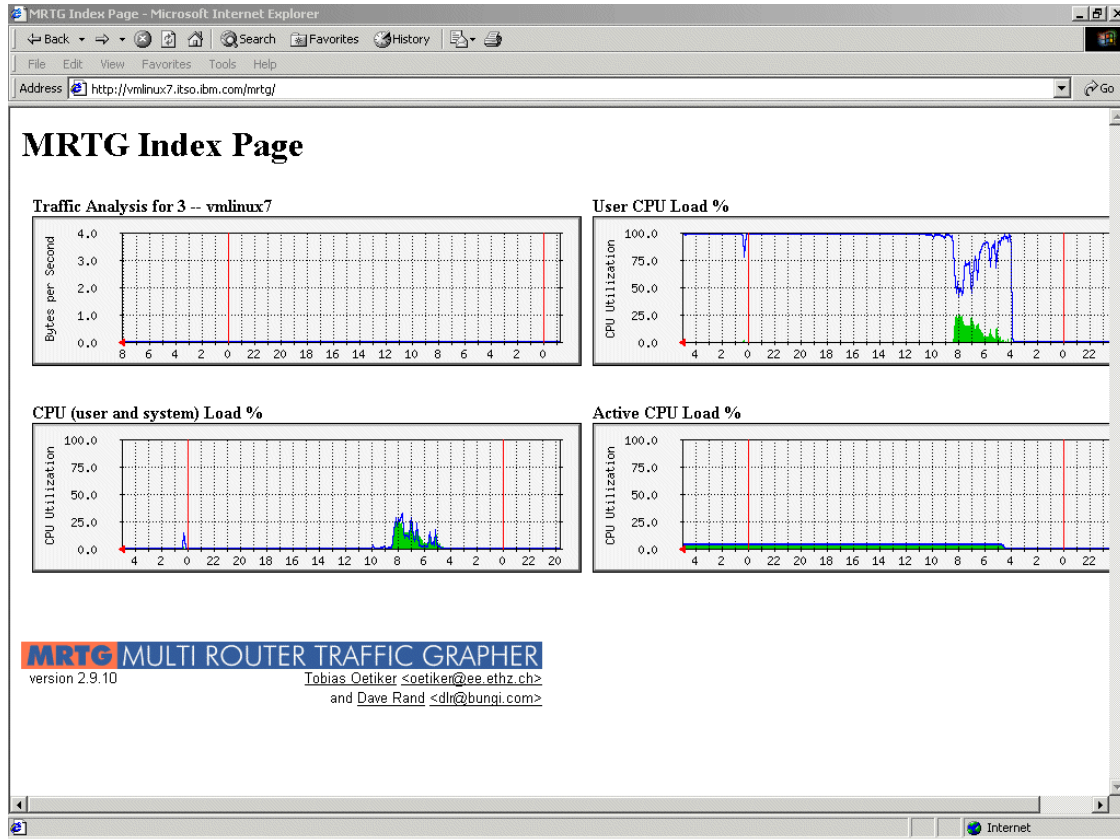


Figure 13-2 MRTG index page created by indexmaker

13.6.8 MRTG reporting for Virtual CTC or IUCV devices

When running many Linux guests under VM, it is highly probable that many of these guests will not have dedicated network interface cards. Instead, they will gain network connectivity via one or more virtual CTC or IUCV point-to-point connections with a VM TCP/IP stack.

We found that the MRTG program `cfgmaker` does not correctly recognize Virtual CTC or IUCV devices as their `ifSpeed` value in the MIB tree is set to 0. To get around this, we manually added a speed value into the MIB tree by adding a parameter into `/etc/ucdsnmpd.conf` as follows:

```
interface interface_name interface_type speed_in_bits/sec
```

Example 13-10 Changes to /etc/ucdsnmpd.conf for IUCV or CTC devices

```
interface ctc0 111 10000000
interface iucv0 111 10000000
```

In this example, we specified a maximum speed of 10Mbps. The 111 refers to the interface as a “Stack-to-Stack” device. For a complete listing of all available interface types, we referred to the file `/usr/share/snmp/mibs/IANAifType-MIB.txt`

Note: The speed value you specify will not affect the actual speed of the device; it will only alter the scaling of the MRTG graphs. You may need to alter this speed value to suit your environment to ensure the graph scaling is meaningful.

With these changes in place, we recycled the SNMP daemon as follows:

```
# rcsnmpd restart
```

At the level of SuSE we were running at time of writing (SuSE 7.2 beta, kernel 2.2.19), we found that adding a speed value in the MIB tree for the Virtual CTC or IUCV devices was not enough to get bandwidth information. The CTC and IUCV drivers at that level did not perform the necessary byte recording.

To resolve this issue, we made patches to both the CTC and IUCV drivers for the 2.2.19 level of the Linux kernel. These patches are in the file `ctc-iucv-bytestat.patch` and can be downloaded from:

```
ftp://www.redbooks.ibm.com/redbooks/SG246299
```

Ensure that you have the source code for the 2.2.19 Kernel in `/usr/src/linux` (this patch was only tested at 2.2.19). Copy the patch that you've downloaded from the Internet to the directory `/usr/src/linux/drivers/s390/net/`. You should see (among others) the files `ctc.c` and `netiucv.c`. These are the source files for the CTC and IUCV drivers. To apply the patch from the shell, type:

```
# patch -b -i ctc-iucv-bytestat.patch
```

You should see the output:

```
patching file ctc.c
patching file netiucv.c
```

If there are no errors listed, the module source code is now patched. Now compile the modules. To do this, go to the `/usr/src/linux` directory.

Note: This assumes that you have already selected that you want CTC and IUCV module support. If you have not already made this selection, then you need to first run the command `make menuconfig` from `/usr/src/linux`.

Then, under the selection:

```
S/390 Network device support
```

make sure the following selections are made:

```
<M> CTC device support  
<M> IUCV device support (VM only)
```

Exit and save the configuration.

Compile the module support into binaries by typing:

```
# make modules
```

Once this has finished, type:

```
# make modules_install
```

This will install the modules into their runtime directories.

You can now use the modules; refer to *Linux for S/390 Device Drivers and Installation Commands* for information on module syntax. You should now have accurate byte recording for the Virtual CTC and IUCV devices.

Note: Because Linux kernel and device driver development are such fast-moving and dynamic fields, we will not be providing patches for any other level of the CTC/IUCV device drivers. From reviewing the source code for the CTC and IUCV drivers at the 2.4.5 level of the Linux kernel, it appears that the byte-recording limitation has been removed.

13.6.9 Monitoring multiple Linux guests

So far we've outlined how to set up SNMP and MRTG for a single Linux guest. However, as previously mentioned, in the context of this redbook we ran multiple Linux guest systems, so now we describe how to add more Linux guests into the MRTG reporting model.

To add more Linux guests into the MRTG reporting model, you first need to set up UCD-SNMP on any additional Linux guests that you wish to monitor. If you have established UCD-SNMP on a master Linux system that you use for cloning, then no further installation or configuration requirements may be necessary. However, if you do not use a cloning technique, then you have to set up UCD-SNMP according to the guidelines in this chapter.

Once the Linux systems that you wish to monitor have SNMP running, you can then run the MRTG **cfgmaker** utility to **snmpwalk** their MIB tree. This will determine what network interfaces they are using.

The syntax of *cfgmaker* is as follows:

```
cfgmaker community_name@host_name > config_file_name
```

Run it against another Linux guest, as follows:

```
# cfgmaker itso@vmlinux2.itso.ibm.com > mrtg.conf.vmlinux2
```

Now edit the `mrtg.conf.vmlinux2` file, extracting the necessary interface information and adding it to your existing `/etc/mrtg.conf` file.

In our example, we added the definitions shown in Example 13-11 to the `/etc/mrtg.conf` file:

Example 13-11 Interface information gathered from walking vmlinux2's MIB tree

```
Target[vmlinux2.itso.ibm.com_4]: 4:itso@vmlinux2.itso.ibm.com:
SetEnv[vmlinux2.itso.ibm.com_4]: MRTG_INT_IP="9.12.6.99" MRTG_INT_DESCR="eth1"
MaxBytes[vmlinux2.itso.ibm.com_4]: 1250000
Title[vmlinux2.itso.ibm.com_4]: Traffic Analysis for -- vmlinux2
PageTop[vmlinux2.itso.ibm.com_4]: <H1>Traffic Analysis for -- vmlinux2</H1>
<TABLE>
  <TR><TD>System:</TD>      <TD>vmlinux2 Guest machine.</TD></TR>
  <TR><TD>Maintainer:</TD> <TD>SEW <simon@42.org></TD></TR>
  <TR><TD>Description:</TD><TD>eth1 </TD></TR>
  <TR><TD>ifType:</TD>      <TD>ethernetCsmacd (6)</TD></TR>
  <TR><TD>ifName:</TD>      <TD></TD></TR>
  <TR><TD>Max Speed:</TD>   <TD>1250.0 kBytes/s</TD></TR>
  <TR><TD>Ip:</TD>          <TD>9.12.6.99 (vmlinux2.itso.ibm.com)</TD></TR>
</TABLE>
```

Figure 13-3 on page 318 shows an example of an MRTG Web page reporting on network traffic for four Linux guest systems.

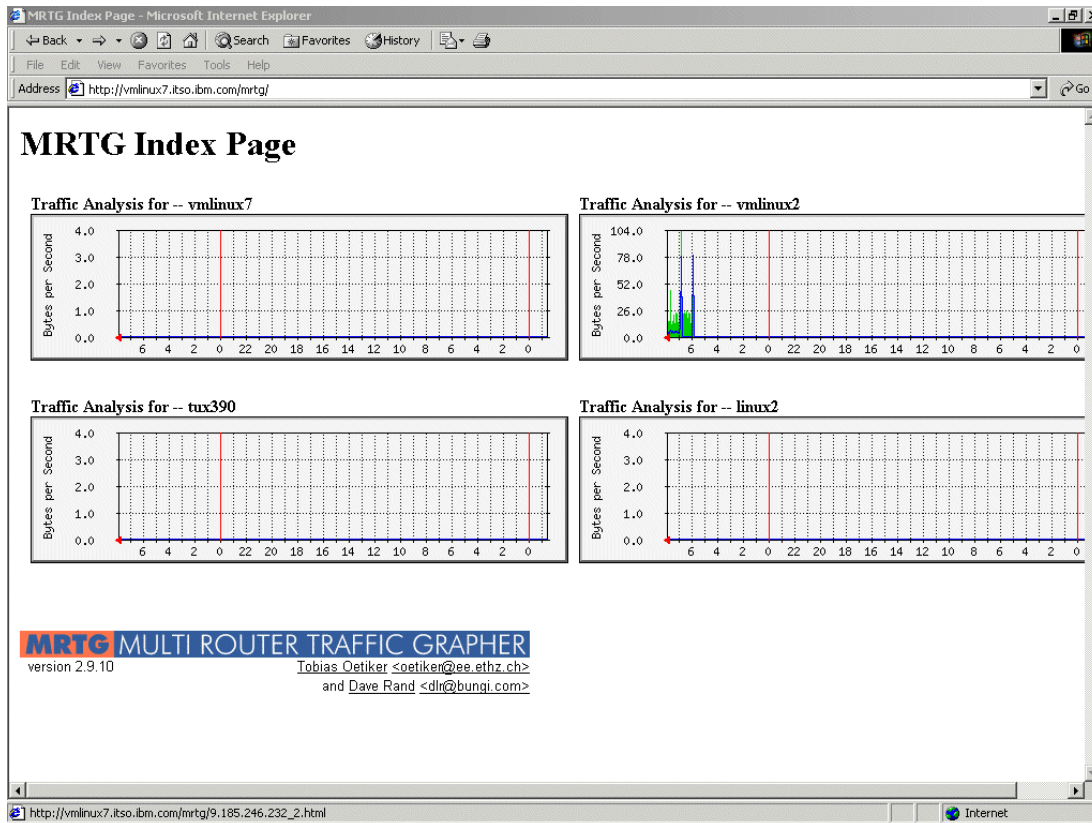


Figure 13-3 Multiple Linux guests on a single page

So far, we only looked at the traffic throughput reporting capabilities of MRTG. In the following section, we show how to determine how much total bandwidth over time is being used by the individual Linux guest machines.

13.6.10 Total bandwidth reporting

To report on the total network traffic profile for our Linux guests, we used the `mrtg_total` Perl script written by Josef Wendel. This script generates HTML reports for bandwidth usage on a per day and per month basis. The script is available on the Internet at:

http://www.geocities.com/josef_wendel/mrtg_total.html

There are a number of prerequisite Perl modules which you may not have installed on your system.

These modules are available from the Comprehensive Perl Archive Network (CPAN) Web site at the following URL:

<http://cpan.valueclick.com/modules/by-module/GD>

The modules we downloaded were:

- ▶ GD-1.33
- ▶ GDGraph-1.33
- ▶ GDTextUtil-0.80
- ▶ GDGraph3d-0.55

To use these modules, place the `mrtg_total` script and all the modules into a suitable directory on the Linux guest that is running MRTG. In our case, we placed the files in a directory called `/home/gd`.

Untar all of the files, then go into each subdirectory in the module order listed above and install that particular module. If you do not install them in the order listed, you may be trying to install a module that has a dependency on a module that hasn't been installed yet. Following is the first module installation, as an example:

```
$ cd /home/GD-1.33
$ perl Makefile.PL
$ make
$ make install
```

Now copy the `mrtg_total.pl` script to the `/usr/local/bin` directory.

Note: If you run `mrtg_total.pl` and get an error message stating `bad interpreter: No such file or directory`, then you need to change the location of the Perl program as referenced in the `mrtg_total.pl` script. The script expects to find the Perl executable in `/usr/local/bin/perl`. In a default SuSE system, that executable resides in `/usr/bin/perl`.

The `mrtg_total` script should be run once a day to generate its report. We ran it automatically by placing an entry into the `/etc/crontab` file.

Following is an `/etc/crontab` definition to run `mrtg_total` automatically once a day:

```
10 0 * * * root /usr/local/bin/mrtg_total.pl /etc/mrtg.conf
```

We were now ready to edit the MRTG configuration file to add extra parameters for each network interface so that the `mrtg_total` script will generate cumulative traffic information about each interface.

Using one of the interfaces from our previous example, we added the lines in italics to the `/etc/mrtg.conf` file; see Example 13-12 on page 320.

Example 13-12 Added parameters to include cumulative traffic information about each interface

```
### Interface 3 >> Descr: 'eth0' | Name: '' | Ip: '9.12.6.73' | Eth: '00-06-29-6c-cb-ce' ###
Target[localhost_3]: 3:localitso@localhost:
##-#Total[localhost_3]:      Traffic Totals for 9.12.6.73
##-#Total-Unit[localhost_3]: M
##-#Total-Ratio[localhost_3]:yes
SetEnv[localhost_3]: MRTG_INT_IP="9.12.6.73" MRTG_INT_DESCR="eth0"
MaxBytes[localhost_3]: 1250000
Title[localhost_3]: Traffic Analysis for 3 -- vmlinux7
PageTop[localhost_3]: <H1>Traffic Analysis for 3 -- vmlinux7</H1>
<TABLE>
  <TR><TD>System:</TD>      <TD>vmlinux7 Guest machine.</TD></TR>
  <TR><TD>Maintainer:</TD> <TD>SEW <Simon@42.org></TD></TR>
  <TR><TD>Description:</TD><TD>eth0  </TD></TR>
  <TR><TD>ifType:</TD>      <TD>ethernetCsmacd (6)</TD></TR>
  <TR><TD>ifName:</TD>      <TD></TD></TR>
  <TR><TD>Max Speed:</TD>  <TD>1250.0 kBytes/s</TD></TR>
  <TR><TD>Ip:</TD>         <TD>9.12.6.73 (vmlinux7.itso.ibm.com)</TD></TR>
```

We would add additional lines, with relevant labels and comments, for each device we are reporting on in the MRTG configuration file.

The graphs generated by the `mrtg_total` script reside in the same directory as all the other MRTG HTML files. Figure 13-4 on page 321 and Figure 13-5 on page 322 are two examples of traffic reporting by day and by month for one of our OSA-Express Fast Ethernet cards (as you can see, this was a test system and the traffic was only heavy for a short period).

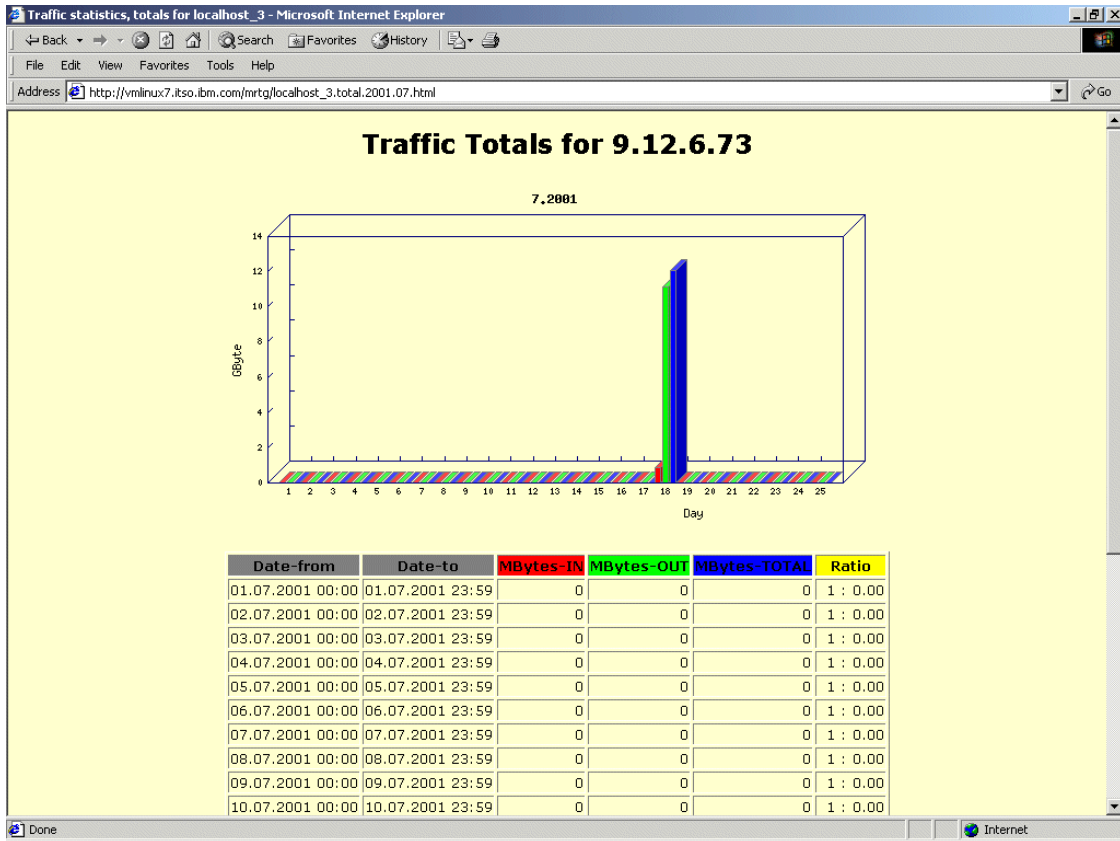


Figure 13-4 Traffic totals by day

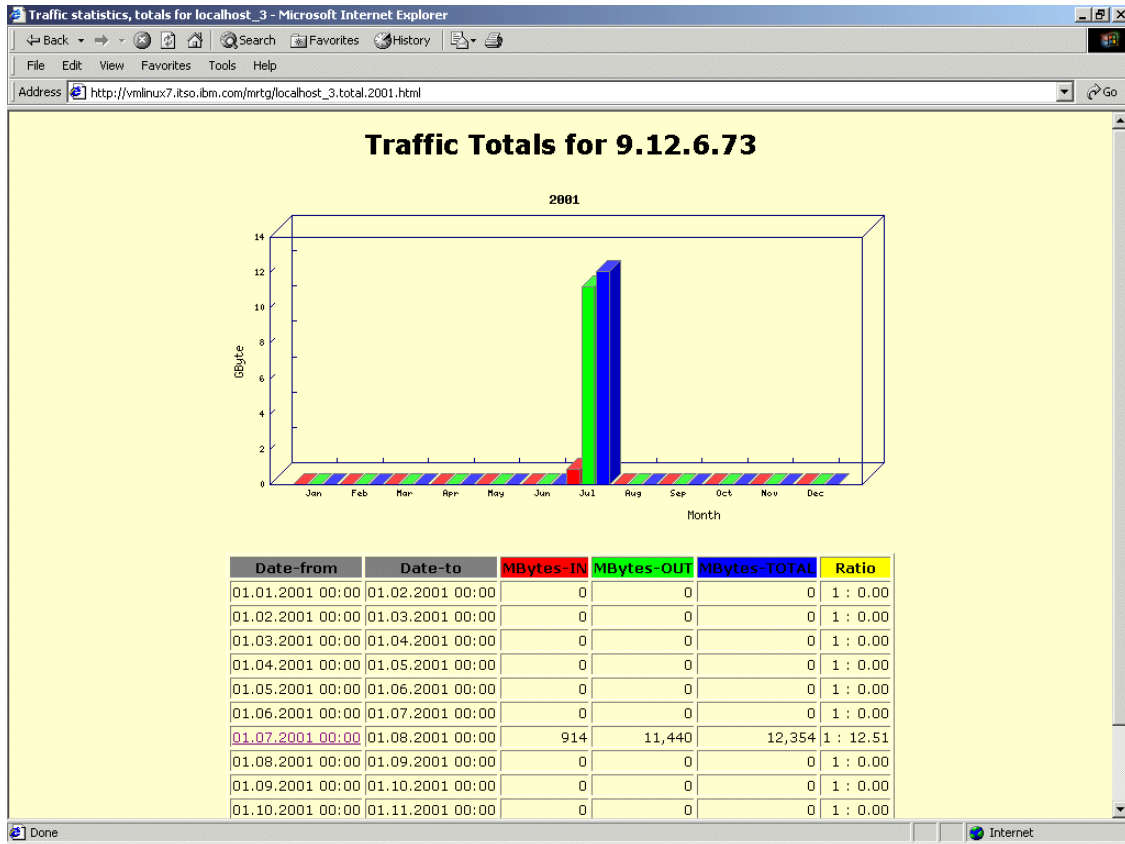


Figure 13-5 Traffic totals by month

13.7 Availability monitoring

In any computing environment, a mechanism is needed to monitor the health of the systems. In an environment with hundreds of Linux guest machines running under VM, it would quickly prove impractical to monitor these systems manually.

As is often the case in the open source community, there are many availability monitoring packages to choose from. We focused on a tool called NetSaint, which is available on the Web at:

<http://www.netsaint.org>

The tool appeared to be straightforward to install and configure, and provided us with enough monitoring function to be useful in an environment with many Linux systems.

13.7.1 NetSaint

Note: The following example illustrates how we installed and configured NetSaint 0.0.6 using a SuSE system. The steps may differ if you are running a different Linux distribution or if you are running a different level of NetSaint.

NetSaint is an open source software package that carries the GPL license. It can be used to monitor not only Linux hosts, but also many other types of servers or routers. It can also monitor individual services that those machines are running.

For example, if a server is unavailable, or network traffic is running slow or disk drives are filling up, NetSaint can be used to e-mail or page support staff. NetSaint uses a modular, “plugin” architecture to perform the actual service checks, which allows you to choose from a large number of plugins and also allows you to develop your own plugins for the specific needs of your environment.

The NetSaint package can be downloaded from the following URL:

<http://www.netsaint.org/download>

The version we used in this redbook was NetSaint 0.0.6. As you may surmise from the version numbering, this was still very much a work in progress; nevertheless, from our testing we found it to be very useful.

We downloaded the package `netsaint-0.0.6.tar.gz`, which includes the core program, CGIs, and documentation from the URL:

<http://www.netsaint.org/download/netsaint-0.0.6.tar.gz>

We also downloaded the NetSaint plugins package (at level 1.2.9-4) from the Source Forge Web site at the following URL:

<http://prdownloads.sourceforge.net/netsaintplug/netsaint-plugins-1.2.9-4.tar.gz>

13.7.2 Installing NetSaint

Note: As long as you are running Linux and have a copy of the GNU C Compiler (gcc), the installation of NetSaint should prove to be simple. However, if you wish to see the graphs produced by the statusmap CGI, you will also need to install the GD library. If the GD library does not come as part of your distribution, you can download it from the following URL:

<http://www.boutell.com/gd>

To check whether or not you have the GD library installed, execute the following **find** command from the shell:

```
# find / -name libgd.a
```

If you get a match, it means that you have the GD library installed.

The installation manual that comes with the NetSaint package is the definitive guide to all aspects of installing and configuring the product. We include the following sections simply to give you a “jump-start” guide to getting the product and up and running quickly.

First decide which Linux guest machine will act as the NetSaint server. This machine will interrogate other machines, as defined in its configuration file, to determine their availability and service level. Once you have selected a suitable Linux guest, upload the two files mentioned above into a suitable directory. We used /home/netsaint/. Untar the files as follows:

```
# tar -zxvf netsaint-0.0.6.tar.gz
# tar -zxvf netsaint-plugins-1.2.9-4.tar.gz
```

You should now have two new subdirectories, netsaint-0.0.6 and netsaint-plugins-1.2.9-4. In our case, we first configured and compiled the core NetSaint program, and then configured and compiled the plugins.

Next, we created a base directory to be used as the runtime directory for NetSaint:

```
# mkdir /usr/local/netsaint
```

Before starting the compilation, we needed to create a NetSaint group and user ID to be used for file ownership. The following two commands illustrate how to do this:

```
# groupadd netsntg
# useradd netsaint -g netsntg
```


We were now ready to run the configuration script to initialize variables and create the makefile:

```
# ./configure --prefix=/usr/local/netsaint --with-cgiurl=/cgi-bin/netsaint \  
--with-htmurl=/netsaint/ --with-netsaint-user=netsaint \  
--with-netsaint-grp=netsntg --with-gd-lib=/usr/local/lib/ \  
--with-gd-inc=/usr/local/include/
```

When that completed successfully, a makefile had been built and we were ready to compile NetSaint core code and the CGIs:

```
# make all
```

After that completed successfully, we installed the binaries, documentation, and sample HTML files:

```
# make install
```

We created and installed sample configuration files using the following commands:

```
# make config  
# make install-config
```

We needed to change directory to the plugins install directory, and run the configuration script for the NetSaint plugins, as follows:

```
# cd /home/netsaint/netsaint-plugins-1.2.9-4  
# ./configure --prefix=/usr/local/netsaint --with-netsaint-user=netsaint \  
--with-netsaint-group=netsntg --with-cgiurl=/cgi-bin/netsaint
```

After that completed successfully, we compiled the plugins with the following command:

```
# make all
```

Finally, we installed the binaries to the directory `/usr/local/netsaint/libexec/` as follows:

```
# make install
```

13.7.3 Configuring the Web interface

The next step is to configure the Web interface so that you can access NetSaint status pages and run the NetSaint CGI programs from your Web browser.

Note: The following section makes the assumption that you are running Apache as your Web server.

We edited our Apache configuration file (by default, this usually resides in a file called `/etc/httpd/httpd.conf`).

We added the following line (in our example, we added it at the bottom of the file):

```
Alias /netsaint/ /usr/local/netsaint/share/
```

Next we needed to create an alias for the NetSaint CGIs:

```
ScriptAlias /cgi-bin/netsaint/ /usr/local/netsaint/sbin/
```

Important: The `ScriptAlias` definition must *precede* the default Apache `cgi-bin` `ScriptAlias` entry in the `httpd.conf` file.

Now we restarted the Apache Web server; if running SuSE, this can be done as follows:

```
# rcapache restart
```

You should now be able to go to the NetSaint Web interface by pointing your Web browser at the following URL:

```
http://whatever_your_hostname_is/netsaint
```

You should be greeted with the screen shown in Figure 13-6 on page 327:

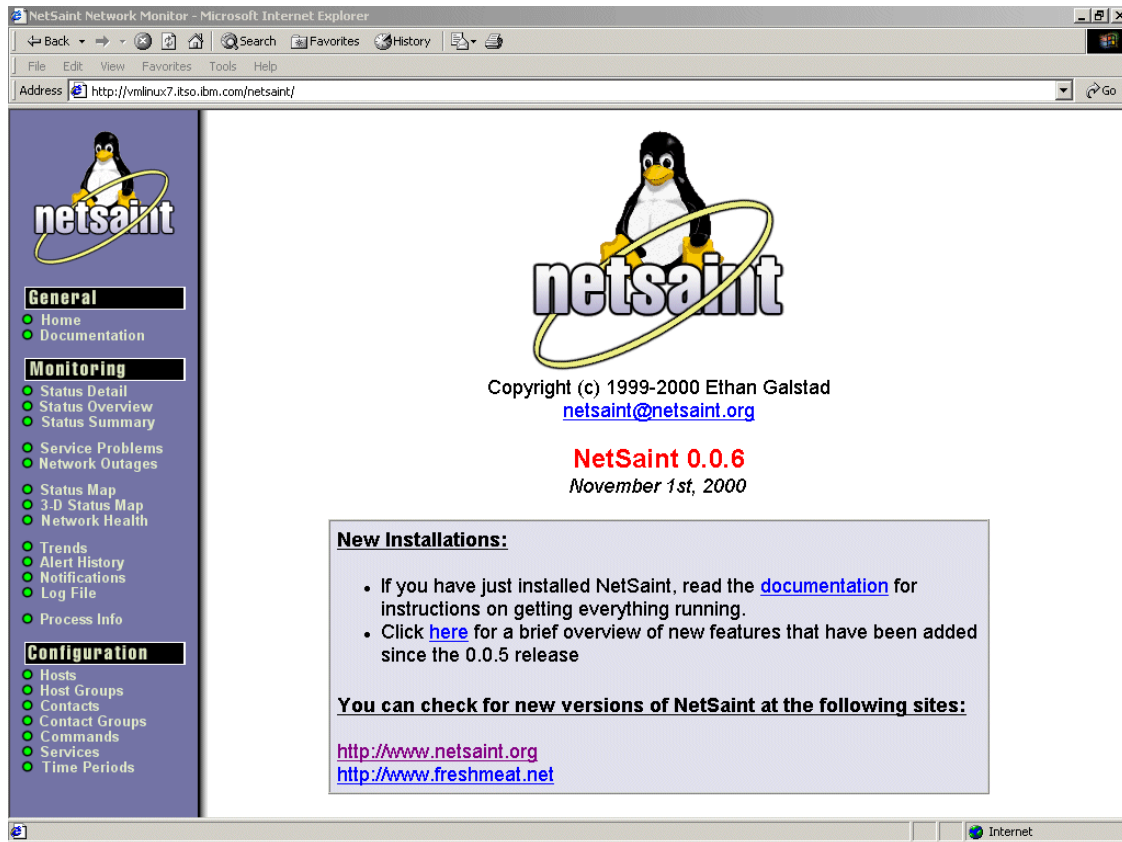


Figure 13-6 NetSaint welcome screen

13.7.4 User authorization

We secured our NetSaint services, so that only authorized staff can use the tool, by adding the lines shown in Example 13-13 to the `/etc/httpd/httpd.conf` file:

Example 13-13 User Authentication statements in `httpd.conf`

```
<Directory /usr/local/netsaint/sbin>
AllowOverride AuthConfig
order allow,deny
allow from all
Options ExecCGI
</Directory>

<Directory /usr/local/netsaint/share>
AllowOverride AuthConfig
```

```
order allow,deny
allow from all
</Directory>
```

At this point we've only told Apache that access to the NetSaint CGI's and HTML files requires authorization. We now needed to define the users that are authorized to access those files using the Apache **htpasswd** program. The command syntax is as follows:

```
# htpasswd -c /usr/local/netsaint/etc/htpasswd.users netsaintadmin
```

This will create a file called `htpasswd.users`, with the first user ID being `netsaintadmin`. The **htpasswd** program will prompt you for a password for that user ID. To create additional users, use the following command:

```
# htpasswd /usr/local/netsaint/etc/htpasswd.users <username>
```

We now needed to create a file called `.htaccess` (yes, that is a period at the front of the file name). A copy of the file must reside in two locations: `/usr/local/netsaint/sbin`, and `/usr/local/netsaint/share`. The contents of the file should be as follows:

```
AuthName "NetSaint Access"
AuthType Basic
AuthUserFile /usr/local/netsaint/etc/htpasswd.users
require valid-user
```

Finally we needed to make some modifications to the CGI configuration file `/usr/local/netsaint/etc/nscgi.conf`.

The following changes were made for our configuration:

```
use_authentication=1
authorized_for_system_information=*
authorized_for_configuration_information=*
authorized_for_system_commands=*
authorized_for_all_services=*
authorized_for_all_hosts=*
authorized_for_all_service_commands=*
authorized_for_all_host_commands=*
```

The `use_authentication=1` parameter enables authorization checking. The `authorized_for` parameters all have a value of asterisk (*). This means that any user who has successfully been authenticated by Apache will have access to these CGIs.

13.7.5 Configuring NetSaint

Since we've configured the NetSaint infrastructure, we now turn to configuring NetSaint itself. NetSaint has three configuration files: a "main" configuration file, `netsaint.cfg`; a "host" configuration file, `hosts.cfg`; and a "CGI" configuration file, `nscgi.cfg`. The configuration files reside in the directory `/usr/local/netsaint/etc/`.

Unless you have made any changes to the default installation steps previously described above, there's no need to make changes to the main configuration file unless you specifically want to.

The bulk of the configuration occurs in the `hosts.cfg` file. It is here that you define the systems that you wish to monitor. This file is quite complex, so we'll use an example environment to describe how to configure it.

Figure 13-7 uses the NetSaint package to display our network and server configuration:

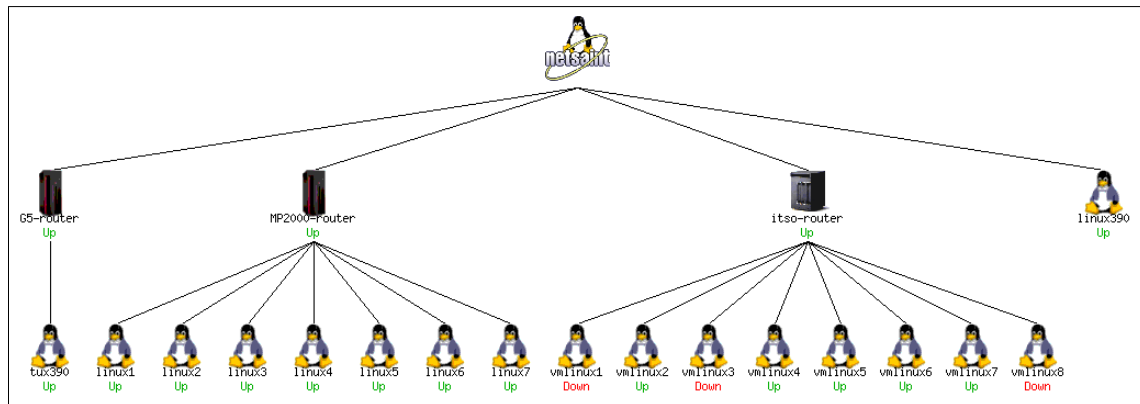


Figure 13-7 Example configuration - diagram generated by NetSaint

We focus on the two machines at the far left of the diagram, the G5-router and tux390. The definitions for all the other machines in the diagram will be very similar to those two.

We start by opening the file `/usr/local/netsaint/etc/hosts.cfg` in an editor.

Host definitions

We first need to discuss the host definition section. The syntax of that section is shown in Example 13-14:

Example 13-14 Host definitions syntax

```
host[<host_name>]=<host_alias>;<address>;<parent_hosts>;<host_check_command>;
```

```
<max_attempts>;<notification_interval>;<notification_period>;  
<notify_recovery>;<notify_down>;<notify_unreachable>;  
<event_handler>
```

Example 13-15 shows our coding:

Example 13-15 Host definitions

```
host[G5-router]=G5 VM 3.1 Router;9.185.122.219;;check-router-alive;20;60;24x7;1;1;1;  
host[tux390]=tux390 VM Guest;9.185.122.217;G5-router;check-host-alive;10;120;24x7;1;1;1;
```

The G5-router is actually a VM TCP/IP stack on a z/VM LPAR running on a 9672 G5 processor. The tux390 host is a Linux VM guest connected to the VM TCP/IP stack via Virtual CTC links.

In the host definition for the G5-router, we say that we want to run the **check-router-alive** command against this machine. The **check-router-alive** command is defined in the file `/usr/local/netsaint/etc/commands.cfg`. It sends a single ICMP ping to the defined machine every 60 seconds.

To change the checking interval from 60 seconds to another value, you must change the `interval_length` parameter; this is set in the `netsaint.cfg` file. If there is 100% packet loss, or if the round trip average is 5000ms (5 seconds) or longer, an error is flagged. These settings are can be configured in the `commands.cfg` file.

Note: For a complete description of all the available configuration options, refer to NetSaint documentation.

Similar definitions were been made for the tux390 host. Note, however, that we defined the G5-router as the parent host of the tux390 machine. This is particularly useful when we use the status map and 3-D status map CGIs, because we can easily see the relationship between different machines in our network.

Host groups

Next, we need to review the host groups section. As the name implies, this allows us to group together one or more hosts for the purposes of notifying support staff when an outage is detected.

The syntax for this section is:

```
hostgroup[<group_name>]=<group_alias>;<contact_groups>;<hosts>
```

In our example, we coded the following:

```
hostgroup[G5Penguins]=All Linux Guests under VM on G5;linux-admins;tux390
hostgroup[routers]=All routers;linux-admins;itso-router,MP2000-router,G5-router
```

Note that we only defined the tux390 host under the G5Penguins group; if we had more Linux guests under this VM system, then we would add their names here.

Command configuration

The command configuration section can be used to define the functions you wish to run when an exception occurs. For example, we chose to use the default e-mail notification, which notifies the defined user whenever a problem is detected:

Example 13-16 Send an e-mail when an exception is detected

```
command[notify-by-email]=/bin/echo -e '***** NetSaint 0.0.6 *****\n\nNotification Type:
$NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState:
$SERVICESTATE$\n\nDate/Time: $DATETIME$\n\nAdditional Info:\n\n$OUTPUT$' | /bin/mail -s '**
$NOTIFICATIONTYPE$ alert - $HOSTALIAS$/SERVICEDESC$ is $SERVICESTATE$ **' $CONTACTEMAIL$
```

Contact configuration

The contact section defines who to contact with the notification that an outage has occurred. We defined the *netsaintuser* as the recipient of all contact messages:

Example 13-17 Contact section

```
contact[netsaintuser]=NetsaintUser;24x7;24x7;1;1;1;1;1;1;notify-by-email,notify-by-epager;host-
notify-by-email,host-notify-by-epager;root
```

When there is a problem, the *netsaintuser* will be notified via e-mail, and also via a pager message sent through an e-mail pager gateway.

Service configuration

NetSaint not only monitors the availability of whole host machines, but it can also monitor individual services running within a machine. For example, we can define services to be monitored such as POP3, HTTP, and DNS. Netsaint will check that these services are indeed active on a machine.

The syntax of the service sections is shown in Example 13-18:

Example 13-18 Service section

```
service[<host>]=<description>;<volatile>;<check_period>;<max_attempts>;
<check_interval>;<retry_interval>;<notification_group>;
<notification_interval>;<notification_period>;<notify_recovery>;
```

```
<notify_critical>;<notify_warning>;<event_hander>;<check_command>
```

For the G5-router, we defined the following service definition:

Example 13-19 G5-router service definition

```
service[G5-router]=PING;0;24x7;3;5;1;linux-admins;240;24x7;1;1;0;;check_ping
```

Because this machine was simply acting as a router to our Linux guest, all we needed to do was ping the machine regularly to verify that it was still up. For our tux390 host, we added several more service definitions, as follows:

Example 13-20 tux390 host service definition

```
service[tux390]=PING;0;24x7;3;5;1;linux-admins;240;24x7;1;1;0;;check_ping
service[tux390]=HTTP;0;24x7;3;2;1;linux-admins;240;24x7;1;1;1;;check_http
service[tux390]=DNS;0;24x7;3;2;1;linux-admins;240;24x7;1;1;1;;check_dns
```

Along with regular pings to check machine availability, we also checked that the Web server and DNS were running.

The final step to complete before running NetSaint is to edit the nscgi.conf file. In our case, we wanted to add definitions for site-specific graphics and enable NetSaint process checking by making the following additions:

Example 13-21 Changes to nscgi.conf file

```
hosttextinfo[tux390]=/serverinfo/tux390.html;bluetux.gif;bluetux.jpg;bluetux.gd2;Linux 390;
hosttextinfo[G5-router]=/serverinfo/g5.html;G5icon.gif;G5icon.jpg;G5icon.gd2;System 390 G5;
```

```
netsaint_check_command=/usr/local/netsaint/libexec/check_netsaint \
/usr/local/netsaint/var/status.log 5 '/usr/local/netsaint/bin/netsaint' #uncomment this line
```

We added graphics (.gif, .jpg and .gd2 files) so that the various screens available in the NetSaint Web interface will display our unique icons.

Note: You need to create these graphics yourself. To create .gd2 files, you can use the utility pngtogd2, which is part of the GD package.

13.7.6 Starting and stopping NetSaint

Note: Refer to the NetSaint documentation for a complete description of all options available for starting, stopping, and restarting NetSaint.

There are four methods of starting NetSaint: manually from a shell in the foreground; manually from a shell, but running NetSaint as a background task; manually as a Daemon; or automatically, as the Linux system boots.

While initial testing is carried out, we recommend that you manually run NetSaint as a foreground task. When you're comfortable that everything is configured correctly, you can automate NetSaint to start at boot time with the following command:

```
/usr/local/netsaint/bin/netsaint /usr/local/netsaint/etc/netsaint.cfg
```

To stop NetSaint when running in foreground mode, simply press <CTRL-C> to get out of the program. Refer to NetSaint documentation for other methods.

13.7.7 Using NetSaint

Once you have completed the configuration, you're ready to start NetSaint. Go to the main Web page to begin verifying that it is indeed working as you expect.

Type in the valid URL from your Web browser (it should be something like the following):

```
http://your_hostname/netsaint
```

You'll be prompted for a valid user ID and password; if you followed the example in this chapter, the user ID should be netsaintadmin. You will then be greeted by the Web page as displayed in 13.7.3, "Configuring the Web interface" on page 325.

To verify that your configuration definitions are successful, select the option **Status Summary**. Using that screen, as shown in Figure 13-8 on page 334, you can then drill down to individual servers.

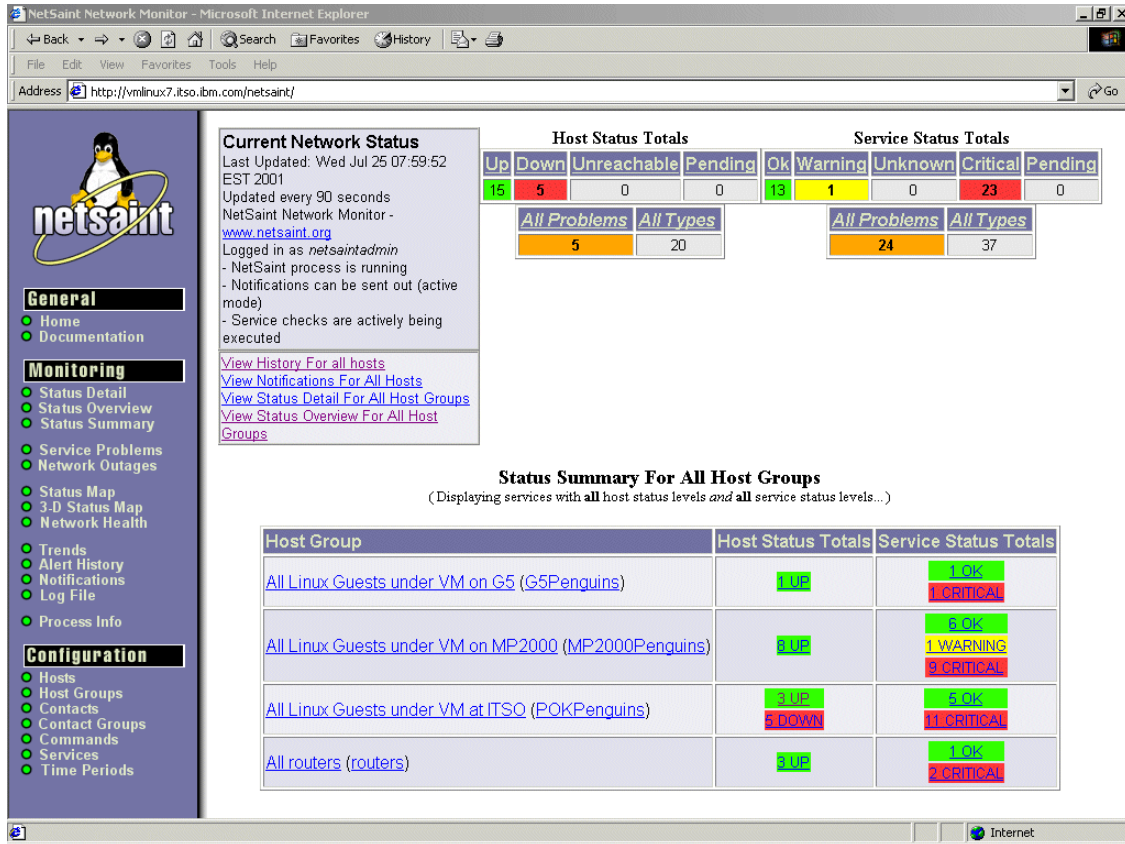


Figure 13-8 NetSaint Status Summary screen

NetSaint has two very useful graphic-based CGIs: `statusmap.cgi`, and `statuswrl.cgi`. `statuswrl.cgi` produces VRML output.

Important: If you select the 3D-Status Map option from the Web page and it prompts you to save a file to disk, then it means you do not have a VRML plugin installed for your Web browser. The NetSaint documentation recommends using a VRML plugin such as Cortona, Cosmo Player or WorldView.

Figure 13-9 on page 335 and Figure 13-10 on page 336 show examples of these CGIs.

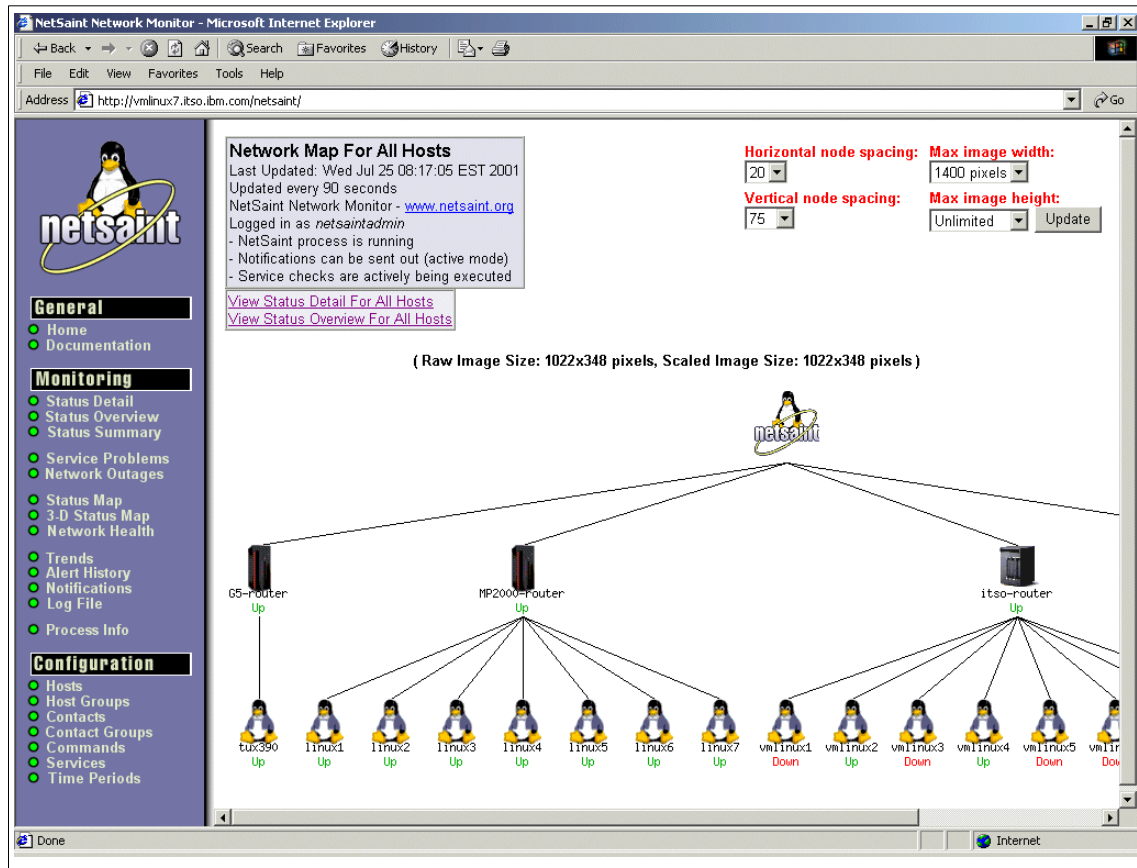


Figure 13-9 Status map CGI

You can select individual machines and gather more information about their service status.

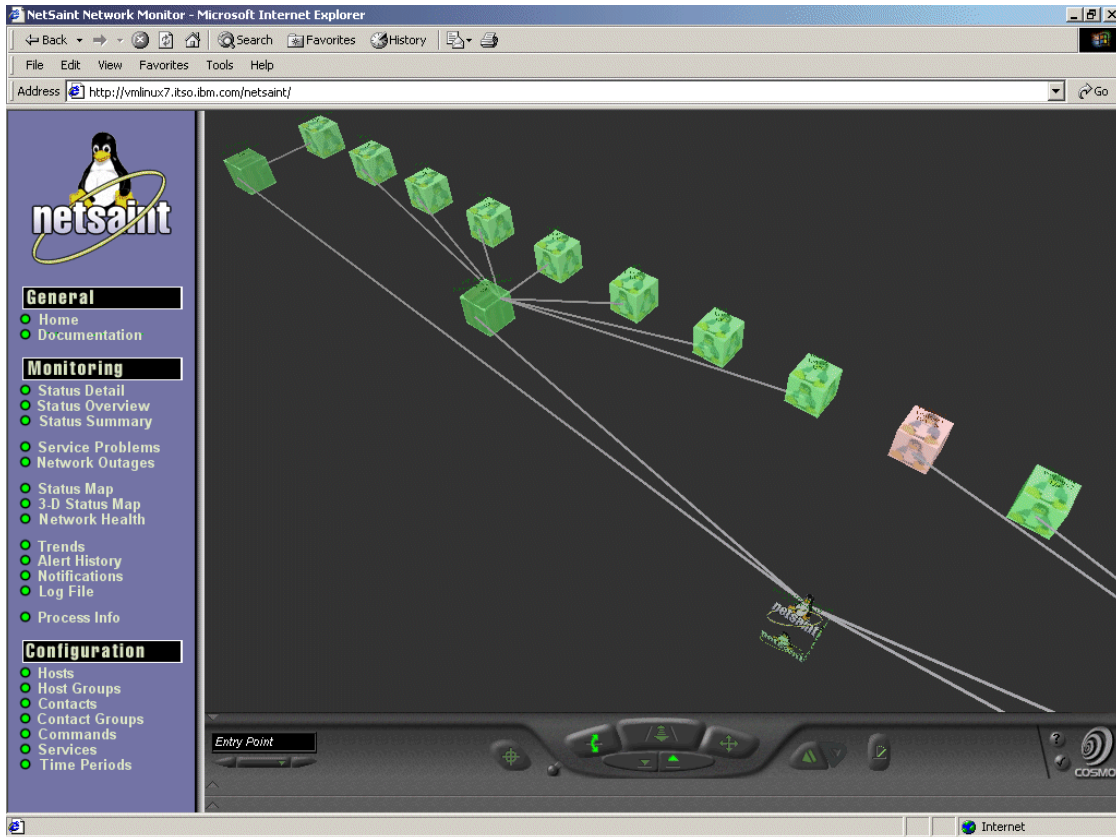


Figure 13-10 3D status map

With the 3D status map, you can literally fly around the Linux guests, identifying ones that may have problems (in our example, the problem guest displays as red).

If you find a machine that has a service problem, you can click that guest and be transferred to the status page for that specific machine. From there you can drill down further, thus pinpointing exactly what the problem is.

In Figure 13-11 on page 337, we can see that the Linux guest vmlinux1 is currently down:

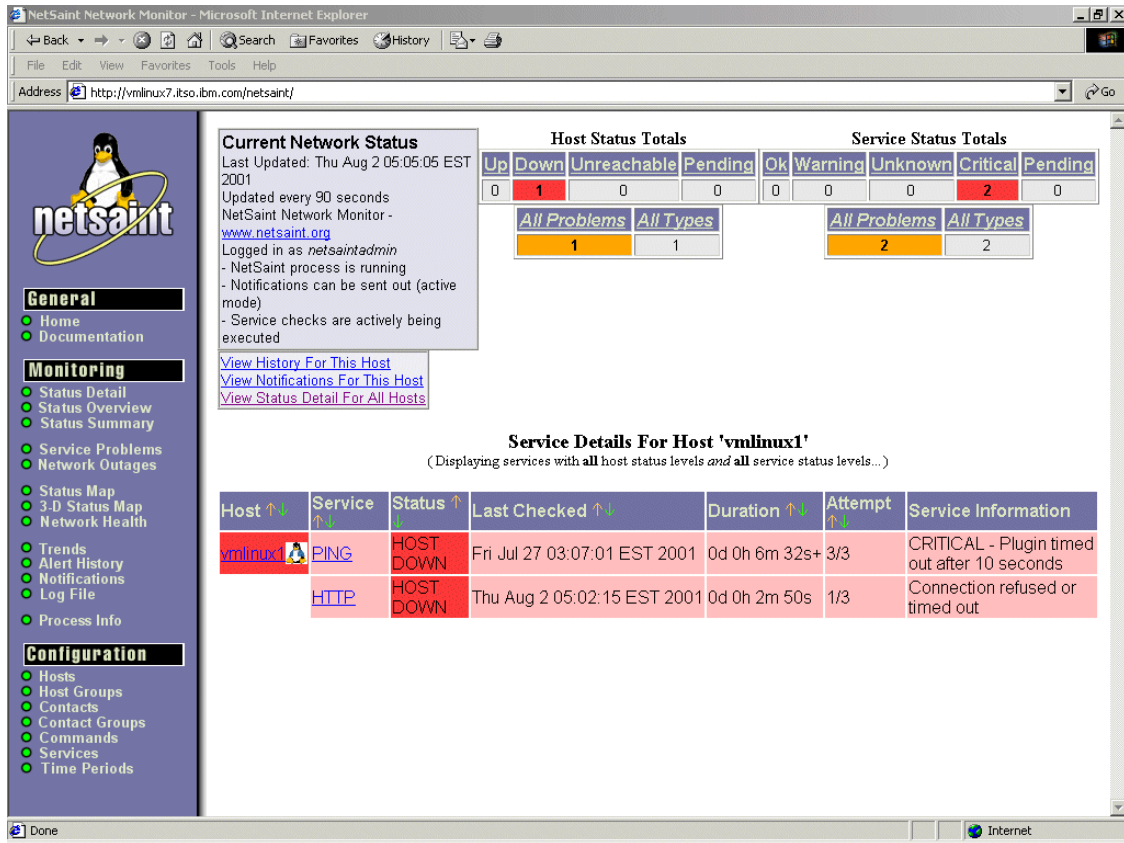


Figure 13-11 Service details screen

13.8 Summary

In this chapter we have outlined some of the methods by which an organization can account for resources consumed by Linux/390 VM Guest systems. We have also examined just a few of the many ways in which organizations can monitor their Linux systems.

Comprehensive monitoring can be done very cheaply through many, very professional, Open Source packages. And as the popularity of Linux grows, more and more tools are becoming available every month. These tools are being written both by the open source community and by an ever-increasing number of software vendors, such as Tivoli.

Undoubtedly the methods and sophistication of accounting for Linux resource consumption will continue to mature, as we have seen on other platforms.



Web application servers

One of the most significant developments to appear recently in the software world is the Web application server. Although technically “application server” is a generic term, in common usage it refers to a server that implements the Java 2 Enterprise Edition (J2EE) standard. Some Java application servers include IBM WebSphere Application Server, BEA WebLogic, Lutris Enhydra, and Tomcat and Resin from the Apache Project.

Properly deploying WebSphere Application Server or any other application server is a very complex undertaking with many key decision points that are dependent on the particulars of each application. Fully exploring such a deployment is beyond the scope of this book.

There are issues with WebSphere Application Server (and other application servers) that require special consideration in the Linux for zSeries and S/390 environment. In this chapter, we discuss some of these issues and provide possible solutions.

14.1 WebSphere issues

In order for you to effectively deploy WebSphere on Linux for zSeries and S/390, we need to explain how WebSphere workloads differ from the other workloads we've discussed. These differences have a major effect on your deployment options and the relative merits of those options.

14.1.1 Java Virtual Machine (JVM)

As an implementation of the J2EE standard, WebSphere is based completely on Java technology. This means that all of WebSphere (both internal and application components) runs in a Java Virtual Machine (JVM). In some sense, the JVM can be viewed as its own small virtual server, which emulates a sort of system with a limited set of instructions and functions. Compiled Java byte-code is executed by the JVM, which runs as a process under the host operating system.

Some implementations of the JVM include Just-In-Time compilation (JIT). This technique involves compiling the Java byte-code into native instructions just before execution. This technique potentially provides significant improvements in execution time.

As a result of this interpreted implementation, Java applications tend to be somewhat more CPU-intensive than would otherwise be expected. For starters, the byte-code interpretation can take up to half of the execution time. There is also some overhead associated with Java's garbage collection routines. In the case of JIT compilation, some extra CPU time is still required to perform the translation to native code, though the total is usually still much less than for fully interpreted byte-code.

The JVM is also a pure stack-based machine; there are no "registers" to pass parameters, so data is always allocated on the stack (and thus uses memory).

14.1.2 Objects

Java is considered an object-oriented language. Object-oriented programming is a tool for improving software engineering methods and programmer effectiveness, not performance. Several artifacts of Java's object-orientation affect the resource demands of Java code:

- ▶ Loading of class hierarchies - loading one class requires loading all that class's ancestors, as well
- ▶ Indirection - most data accesses are through at least two levels of indirection, which makes memory access a more significant part of overall performance

- ▶ Dynamic binding - most methods are looked up by name (i.e. a string) rather than by an address, thus requiring lots of string parsing overhead at runtime¹
- ▶ Cache-unfriendly behavior - objects tend to be less easily kept in cache, especially for architectures with a large number of distinct objects

In general, these behaviors mean that Java code tends to be a large consumer of memory, both in terms of usage and bandwidth.

It is important to also recognize that the architecture of WebSphere applications themselves can have a significant impact on the run-time behavior. Applications that are written as a few, relatively large objects can take advantage of a deep private cache (such as that provided by the pSeries) and thus are not as efficient on a shared-cache architecture such as the zSeries. Conversely, applications composed of many smaller objects will tend to be more efficient on the zSeries.

14.2 Options for running WebSphere

IBM WebSphere Application Server presents several different options for deployment in the Linux for zSeries and S/390 environment. Each deployment option has its own advantages and disadvantages, and is more appropriate for some environments than others. In fact, the particular behavior of a specific WebSphere Application Server application may make it more appropriately implemented in a particular way.

Note that because of the relatively high memory and CPU demands of WebSphere, we do not currently recommend running many (tens or hundreds) of separate WebSphere Application Server instances on Linux for zSeries and S/390.

14.2.1 WebSphere Application Server for zSeries and S/390 Linux

One option is to use WebSphere Application Server V3.5 for Linux for zSeries and S/390. This version is shipped with the Java 1.2.2 JVM rather than the Java 1.3, and thus will not have the performance enhancements offered by the new JVM. However, this option allows the user to leverage their existing expertise in Linux

We hope that eventually WebSphere Application Server V4 will be released for Linux for zSeries and S/390, and that some of the performance enhancements will carry over to this platform; however, at the time of writing no such release has been announced.

¹ This is not to say that this technique is necessarily a negative; it actually simplifies relocation, for one. But nothing comes for free, and there is a runtime cost associated with it.

14.2.2 WebSphere Application Server for z/OS

Another configuration option is to deploy WebSphere Application Server for z/OS on a separate LPAR running z/OS. At the time of writing, the current released version WebSphere Application Server for z/OS is V4, which includes Java 1.3 JVM. This JVM is expected to have significant performance gains over the Java 1.2.2 JVM included in WebSphere Application Server V3.5.

An additional cost will be the software and staffing costs associated with z/OS. This option may be most appropriate if z/OS is already installed in the organization and personnel are available to support it.

14.2.3 Separate servers

A third option, of course, is to deploy WebSphere Application Server on separate servers - pSeries AIX servers, for example. This method allows WebSphere Application Server to have exclusive access to all the memory and CPU resources on the machine. It also significantly increases the complexity of an installation, requiring additional hardware, power, network, and other resources. However, in situations where WebSphere Application Server is running extremely complex applications at very high utilizations, this may be the best way to handle the workload.

14.3 Planning for WebSphere Application Server

A key factor in planning for WebSphere Application Server is minimizing the impact it has on other z/VM guests while ensuring it has sufficient resources to meet its performance targets. Fortunately, the zSeries architecture provides powerful tools ideally suited to achieving these goals. By combining z/VM and LPARs, we can deploy WebSphere Application Server in ways that maximize performance and manageability.

We *strongly* recommend that WebSphere Application Server servers be set up in their own LPAR, separate from the other Linux for zSeries and S/390 guests (Web servers and such). This applies to WebSphere Application Server for Linux for zSeries and S/390, as well as to WebSphere Application Server for z/OS. In both cases, having a dedicated set of resources with the same general access patterns makes management simpler and minimizes unexpected interactions.

We also recommend that WebSphere Application Server servers be deployed as z/VM guests. The additional flexibility conferred by z/VM in administering the servers is significant, especially in a production ISP or ASP environment.

14.4 Test environments

One particular advantage of running WebSphere Application Server as a VM guest is that it allows the easy creation of development, test, and staging servers. This becomes especially important since it is not currently feasible to run individual WebSphere Application Servers for hundreds of clients. If customer applications need to be run on a shared WebSphere Application Server environment, then it is critical that there be cordoned-off areas where applications can be thoroughly tested before being deployed on the production server.

This is one area where the zSeries architecture has a distinct advantage. By creating test and staging servers as additional z/VM guests, it is possible to have a test environment that is essentially identical to the production environment, without additional equipment costs. These guests should be created within the WebSphere Application Server-dedicated LPAR to maintain segregation from other Linux for zSeries and S/390 guests.

An example deployment scenario is presented in Figure 14-1. Note that while the example shows the WebSphere Application Server LPAR as running on CPs rather than IFLs, this is only necessary in the case of WebSphere Application Server for z/OS; the processors for that LPAR can be IFLs if running WebSphere Application Server for Linux for zSeries and S/390.

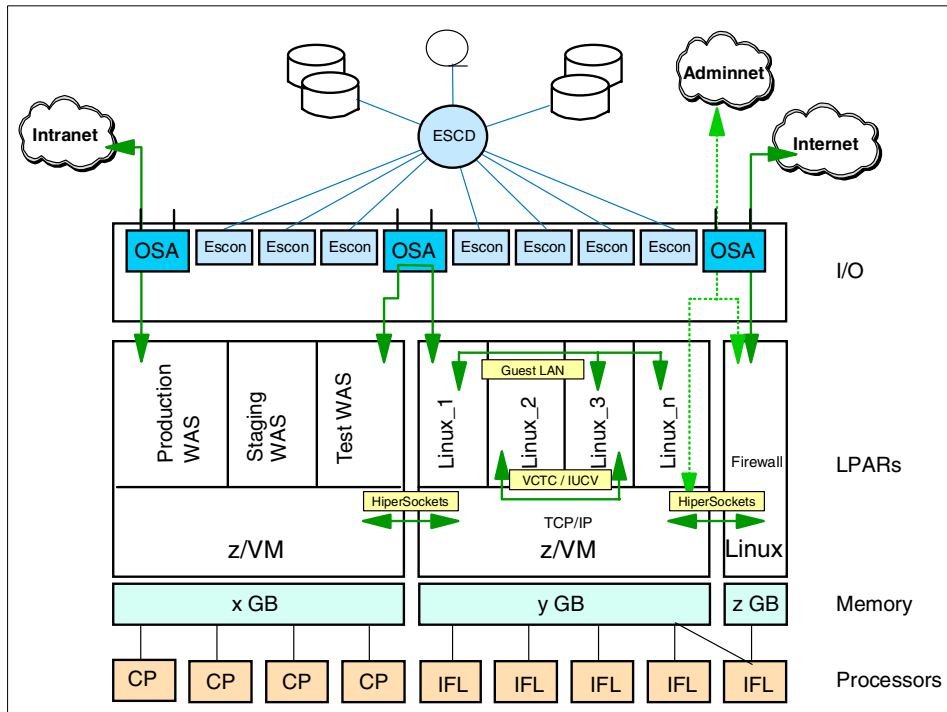


Figure 14-1 Sample WebSphere deployment



Integrating and replacing Microsoft servers

In this chapter we will discuss how to use you VM Linux images to replace existing Microsoft servers or integrate into the existing Microsoft Windows network.

15.1 Using Samba as a domain controller

Samba can also act as Windows NT 4.0 Primary Domain Controller (PDC) for Windows clients. When set up in this fashion, it has the following capabilities:

- ▶ Perform domain logons for Windows NT 4.0/2000 clients
- ▶ Place Windows 9x clients in user level security
- ▶ Retrieve a list of users and groups from a Samba PDC to Windows 9x/NT/2000 clients
- ▶ Supply roaming user profiles
- ▶ Allow Windows NT 4.0 style system policies

Note: Samba 2.2.1 is required for PDC functionality when using Windows 2000 SP2 clients

15.1.1 Setting up a Samba PDC

The following settings should be made in the `/etc/smb.conf` file for Samba to act as Windows PDC:

- ▶ In the `[global]` section, we add the following values:
 - `netbios name = VMLINUX8` // NETBIOS name of the server
 - `workgroup = SAMBAITSO` // domain name
 - `os level = 64`
 - `preferred master = yes` // we want to act as preferred master
 - `domain master = yes` // we want to act as domain master
 - `local master = yes` // we want to act local master
 - `security = user` // passwords are kept on the Samba server
 - `encrypt passwords = yes`
 - `domain logons = yes` // we allow domain logons
 - `logon path = \\%N%\%U\profile` // where the user profiles are stored
 - `logon drive = H:` // where user home directories will be mounted
 - `logon home = \\%N%\%U` // user home directory
 - `logon script = logon.cmd` // generic logon script for all users
- ▶ In the `[netlogon]` section we define the *netlogon* share. The `logon script` value in the global section is relative to this share.
 - `path = /sambashares/netlogon` // directory for *netlogon* share
 - `writeable = no`
 - `writelist = ntadmin` // only administrators can write here
- ▶ `[profiles]` section, here we define the share for storing user profiles:
 - `path = /sambashares/ntprofiles`
 - `writeable = yes`
 - `create mask = 0600`
 - `directory mask = 0700`

The `/etc/smb.conf` from our test system is as follows:

```
# Samba config file created using SWAT
# from tot67.itso.ibm.com (9.12.6.133)
# Date: 2001/08/07 14:05:20

# Global parameters
[global]
    workgroup = SAMBAITSO
    netbios name = VMLINUX8
```

```
server string = Samba PDC 2.2.1a on VMLINUX8
encrypt passwords = Yes
map to guest = Bad User
keepalive = 30
domain logons = Yes
os level = 64
preferred master = True
domain master = True
wins support = Yes
kernel oplocks = No
winbind uid = 10000-20000
winbind gid = 10000-20000
template shell = /bin/bash
winbind separator = +
```

[homes]

```
comment = home-directory
read only = No
create mask = 0750
browseable = No
```

[printers]

```
comment = All Printers
path = /tmp
create mask = 0700
printable = Yes
browseable = No
```

[windowsshare]

```
comment = Share for W2KSAMBA Domain Users group
path = /sambashares/windowsshare
```

[netlogon]

```
path = /sambashares/netlogon
write list = ntadmin
```

[profiles]

```
path = /sambashares/ntprofiles
read only = No
create mask = 0600
directory mask = 0700
```

15.1.2 Creating a machine trust account

For each machine participating in a domain, a machine trust account has to be created on the Domain controller. The password for the machine account acts as a secret for secure communication with the Domain controller. This means that the PDC cannot be spoofed by another computer with the same NETBIOS name trying to access the data. Windows 9x computers are never true members of a domain because they do not have a machine trust account and thus there is no secure communication with the Domain controller.

In the current release, Samba requires a UNIX user ID to exist because Samba computes the Windows NT SIDs from the UNIX UIDs. This means that all machine accounts must have an entry in `/etc/passwd` and `/etc/smbpasswd`. You can create a machine trust account in two ways:

1. Manually

Using the `useradd` command, we added a machine (in our example) with the NETBIOS name TOT11:

```
# useradd -g 100 -d /dev/null -c Peters_W2K -m -s /bin/false TOT11$
```

The following entry is now added to the `/etc/passwd` file:

```
TOT11$:x:501:100:Peters_W2K:/dev/null:/bin/false
```

Now you need to create the `/etc/smbpasswd` entry with the following command:

```
# smbpasswd -a -m TOT11
Added user TOT11$
```

Now you should join the domain from the client computer.

Important: Manually creating is the same as creating an account with Server Manager in Windows NT. From the time you create a machine trust account manually to the time you join to the domain, every client with the NETBIOS name can join the domain. This means that it can get a lot of data from the domain, because PDC inherently trusts the members of a domain.

2. Automatically

The recommended way of creating an account is to add it on the fly. For this you need to add the following parameter to your `/etc/smb.conf` file:

```
add user script = /usr/bin/useradd -d /dev/null -g 100 -s /bin/false -M %u
```


Important: In Samba 2.2.1 only the *root* can be used to add the machine accounts. Therefore, it is necessary to create an entry in `/etc/smbpasswd` for root. For security reasons we recommend that you use a different password from the one used for the UNIX root account.

15.2 Using Samba in Windows domains

Integration of the Linux and Windows worlds is a big challenge of today's IT industry. One of the aspects of this challenge is how to use the same account created in a Windows Active Directory database on the Linux servers. For example, if we incorporate a Samba server into the Windows 2000 Active Directory and want to share the disks from this Samba server, we would normally need to reproduce all user IDs, group IDs, and passwords on the Linux server. To avoid this work, the package Winbind was written.

Winbind combines the worlds of UNIX and Windows NT/2000 by allowing a UNIX server to become a full member of a Windows domain. After joining the domain, the UNIX server sees all users and groups as if they were native UNIX users and groups. This means that whenever the UNIX programs query for the user ID or group ID, they ask the Windows Domain controller for the specified domain to validate the user or group ID.

Winbind hooks into the operating system at a low level, via the NSS name resolution modules in the C library. This redirection to the Windows Domain controller is transparent. Users on the UNIX machine can use Windows user and group names as they would use "native" UNIX names. For example, they can own the files with their user ID, log in to the system and even run an X-window session as Windows Domain users. The only difference in user names is that they have the Domain name incorporated into the user name, for example `DOMAIN\username` or `DOMAIN\groupname`. This is necessary for Winbind so it can determine which Domain controller is responsible for authentication.

Winbind also provides the Pluggable Authentication Module (PAM) to provide authentication via a Windows 2000 domain to any PAM-enabled application. This solves the problem of synchronizing passwords between the systems, because all the passwords are stored in a single location on the Windows Domain controller or Active Directory.

15.2.1 Recompiling the latest Samba package

In our environment we were using the SuSE 7.2 31 bit version of Linux. Before recompiling the newest version you should install the version which comes with the distribution so that the `/etc/rc.config` file gets updated with the correct configuration for Samba. You also need to ensure that the Samba daemon is started automatically by specifying “yes” in `START_SMB` in `/etc/rc.config` as shown in the following:

```
# start samba? ("yes" or "no")
# Windows 95 / NT - File- and Printservices
#
START_SMB="yes"
```

After changing the `START_SMB` setting, run the following command:

```
# SuSEconfig
```

To recompile the latest Samba package with the additional packages needed for integration into Windows domains you need to get the files `samba-latest.tar.gz` and `samba-appliance-0.5-src.tar.gz`. Start at the Samba home page:

<http://www.samba.org>

Then find a download mirror. The latest Samba package is in the top directory and the Samba appliance package is in the subdirectory named `appliance`. Copy those two files into the `/usr/src` directory. Before recompiling the two packages, check if you have installed the following packages:

- ▶ `pam_devel`

To successfully recompile the latest Samba (in our example we used version 2.2.1a), follow these steps:

1. Unpack the latest source:

```
# cd /usr/src
# tar xzf samba-latest.tar.gz
```

2. Start the configuration script for Makefile:

```
# cd samba-2.2.1a/source
# ./configure --prefix=/usr --libdir=/etc --with-privatedir=/etc \
--localstatedir=/var/log --with-codepagedir=/usr/lib/samba/codepages \
--sbindir=/usr/sbin --with-smbmount --with-automount --with-vfs \
--with-quotas --with-profile --with-msdfs --mandir=%{_mandir} \
--with-swatdir=/usr/lib/samba/swat \
--with-sambabook=/usr/lib/samba/swat/using_samba --with-pam \
--with-pam_smbpass
```

3. After the Makefile is created, compile the package:

```
# make LOCKDIR=/var/lock/samba
```

4. Stop the Samba daemon:

```
# /etc/init.d/smb stop
```

5. Install the newly compiled Samba:

```
# make install LOCKDIR=/var/log/samba
```

6. Start the newly installed Samba:

```
# /etc/init.d/smb start
```

To check if you are really running the new version, execute the following command; the output should be similar to the following:

```
# smb -V  
Version 2.2.1a
```

If you managed to do all the tasks just described, you are now running the latest version of Samba. Congratulations!

Now we need to compile utilities from the Samba appliance source code. To do this, follow these steps:

1. Unpack the latest source:

```
# cd /usr/src  
# tar xzf samba-appliance-0.5-src.tar.gz
```

2. Start the configuration script for Makefile:

```
# cd samba-appliance-0.5/source/tng  
# cp /usr/share/automake/config.sub .  
# ./configure --prefix=/usr --libdir=/etc --with-privatendir=/etc \  
--localstatedir=/var/log --with-codepagedir=/usr/lib/samba/codepages \  
--sbindir=/usr/sbin --with-smbmount --with-automount --with-vfs \  
--with-quotas --with-profile --with-msdfs --mandir=%{_mandir} \  
--with-swatdir=/usr/lib/samba/swat -enable-static=yes -enable-shared=no \  
--with-sambabook=/usr/lib/samba/swat/using_samba --with-pam \  
--with-pam_smbpass
```

3. After the Makefile is created, compile the following packages: samedit, nsswitch and winbind:

```
# make nsswitch LOCKDIR=/var/lock/samba  
# make bin/samedit LOCKDIR=/var/lock/samba
```

4. Copy the following files into your Samba bin directory (in SuSE 7.2, this is /usr/sbin):

```
# cp bin/windindd /usr/sbin  
# cp bin/wbinfo /usr/sbin  
# cp bin/sanmedit /usr/sbin  
# cp nsswitch/libnss_winbind.so /lib/libnss_winbind.so.2  
# cp nsswitch/pam_winbind.so /lib/security/pam_winbind.so
```

15.2.2 Joining the Active Directory

By using the tools we compiled, we now join our Linux Samba server to the Windows 2000 Active Directory. In our example, we installed Windows 2000 Server with the following setup:

- ▶ Windows 2000 server with SP2 installed
- ▶ Domain name: itso.ibm.com
- ▶ NETBIOS domain name: ITSO
- ▶ Host name: itsont1.itso.ibm.com
- ▶ IP Address: 9.12.0.60/255.255.255.0, gateway 9.12.0.1

The VM Linux Samba server has the following attributes:

- ▶ Samba 2.2.1a with Winbind extensions from Samba-appliance-0.5 version
- ▶ Host name: vmlinux8.itso.ibm.com
- ▶ IP Address: 9.12.6.72/255.255.255.0, gateway 9.12.6.75

To join the Windows 2000 Active Directory we use the `samedit` command. Follow these steps to join the Linux Samba server to the Windows 2000 Active Directory:

1. Modify the `/etc/smb.conf` file to include the following parameters:

```
workgroup = ITSO
security = DOMAIN
password server = 9.12.0.60
encrypt passwords = yes
```

2. Connect to the Windows 2000 server:

```
# samedit -S ITSONT1 -W ITSO -U Administrator
```

Type in the Administrator password; the output of the command should be similar to the following:

```
# samedit -S ITSONT1 -W ITSO -U Administrator
added interface ip=9.12.6.72 bcast=9.12.6.255 nmask=255.255.255.0
Enter Password:
Server: \\ITSONT1:    User: Administrator    Domain: ITSO
Connection:    1st session setup ok
2nd session setup ok
OK
```

If the NETBIOS server name (in our example ITSONT1) could not be resolved into the IP address by your DNS server, you can add the NETBIOS server name into the `/etc/lmhosts` file, which provides the NETBIOS-to-IP address resolution. You can see the example of the `/etc/lmhosts` file in Example 15-1.

Example 15-1 The `/etc/lmhosts` file

```
# This file provides the same function that the
# lmhosts file does for Windows.
```

```
# It provides another way to map netbios names to ip addresses.
# See the section on 'name resolve order' in the manual page to
# smb.conf for more information.
```

```
# Sample entry:
# 192.168.1.1 samba
9.12.0.60 ITSONT1
```

-
3. After successfully logging into the Windows 2000 server, use the following commands (Example 15-2) to add your Linux Samba server to the Active Directory:

Example 15-2 Adding a user to a domain

```
# samedit -S ITSONT1 -W ITS0 -U Administrator
added interface ip=9.12.6.72 bcast=9.12.6.255 nmask=255.255.255.0
Enter Password:
Server: \\ITSONT1:      User: Administrator   Domain: ITS0
Connection:      1st session setup ok
2nd session setup ok
OK
[ITSONT1\Administrator@ITS0]$ createuser VMLINUX8$ -L
createuser VMLINUX8$ -L

SAM Create Domain User
Domain: ITS0 Name: vmlinux8$ ACB: [W      ]
Resetting Trust Account to insecure, initial, well-known value: "vmlinux8"
vmlinux8 can now be joined to the domain, which should
be done on a private, secure network as soon as possible
Create Domain User: OK
[ITSONT1\Administrator@ITS0]$
```

Note: In the case that you get message “Create Domain User: FAILED”, this means that account was created, but it is disabled. To start using the account you need to reset and enable it in “Active Directory Users and Groups” tool on the Windows 2000 server. You will find this account under Computer accounts.

There is an alternative way to join the Windows 2000 Active Directory by using **smbpasswd**. To use this approach, follow these steps:

1. Create a computer account with the “Active Directory Users and Groups” tool on the Windows 2000 server, with the name vmlinux8.
2. Reset the account by right-clicking it and select “Reset Account.”
3. On VM Linux server, join the domain by executing the command:

```
# smbpasswd -j ITS0 -r ITSONT1
```

15.2.3 Setting up Winbind

Before starting Winbind, add the following lines to the `/etc/smb.conf` file:

- ▶ `winbind separator = +`
This is the separator for separating the Windows Domain name from the user name; for example, `ITSO+Administrator`.
- ▶ `winbind uid = 10000-20000`
This means that Windows Domain users will be mapped to this range of UNIX user IDs.
- ▶ `winbind gid = 10000-20000`
This means that Windows Domain groups will be mapped to this range of UNIX group IDs.
- ▶ `winbind cache time = 15`
- ▶ `winbind enum users = yes`
- ▶ `winbind enum groups = yes`
- ▶ `template homedir = /home/%D%U`
This is the definition for the home directory of Windows Domain users.
- ▶ `template shell = /bin/bash`
This is the login shell for Windows Domain users logging into the VM Linux Samba server. If you do not want to allow remote logins for Windows Domain users, use `/bin/false`.

Now you can start Winbind with the command:

```
# windind
```

You can test the Winbind functionality by listing users and groups from the Windows 2000 Active Directory.

Users:

```
# wbinfo -u
```

You will see output similar to that shown in Example 15-3.

Example 15-3 Windows Domain users

```
# wbinfo -u
ITSO+Administrator
ITSO+Guest
ITSO+IUSR_NF5500W2K
ITSO+ivo
ITSO+IWAM_NF5500W2K
```

```
ITSO+krbtgt
ITSO+mikem
ITSO+openldap
ITSO+tot67
ITSO+TsInternetUser
#
```

Users:

```
# wbinfo -g
```

You will see output similar to that shown in Example 15-4.

Example 15-4 Windows Domain groups

```
# wbinfo -g
ITSO+Domain Admins
ITSO+Domain Users
ITSO+Domain Guests
ITSO+Domain Computers
ITSO+Domain Controllers
ITSO+Cert Publishers
ITSO+Schema Admins
ITSO+Enterprise Admins
ITSO+Group Policy Creator Owners
ITSO+DnsUpdateProxy
#
```

15.2.4 Setting up `/etc/nsswitch.conf`

To fully incorporate Winbind functionality into the Linux security layout, you should also modify `/etc/nsswitch.conf` with the following entries:

- ▶ `passwd: files winbind`
- ▶ `groups: files winbind`

You can check the setup of the `nsswitch` configuration with the `id` command as shown in Example 15-5.

Example 15-5 Checking the `id` data for Windows user name

```
# id ITSO+ivo
uid=10000(ITSO+ivo) gid=10000(ITSO+Domain Users) groups=10000(ITSO+Domain
Users)
#
```

With this setup you can now share a directory from your Samba server and assign the Windows Domain group permission to it. In this case, Windows Domain users can use Samba shared directories. In our example, we created the directory /sambashares/windowsshare with the commands:

```
# mkdir /sambashares
# mkdir /sambashares/windowsshare
```

Then we assigned full permissions for the Domain Users group from ITSO Domain as is shown in Example 15-6.

Example 15-6 Setting the permissions for the Windows Domain Users

```
# chgrp "ITSO+Domain Users" windowsshare
# ls -l
total 12
drwxr-xr-x  3 root    root      4096 Aug  3 12:46 .
drwxr-xr-x 19 root    root      4096 Aug  3 12:46 ..
drwxr-xr-x  2 root    ITSO+Dom  4096 Aug  3 12:47 windowsshare
# chmod 770 windowsshare
# ls -l
total 12
drwxr-xr-x  3 root    root      4096 Aug  3 12:46 .
drwxr-xr-x 19 root    root      4096 Aug  3 12:46 ..
drwxrwx---  2 root    ITSO+Dom  4096 Aug  3 12:47 windowsshare
#
```

To share /sambashares/windowsshare, add the following lines to the /etc/smb.conf file as shown in Example 15-7.

Example 15-7 Defining a windows share

```
[windowsshare]
    comment = Share for ITSO Domain Users group
    path = /sambashares/windowsshare
    read only = No
```

15.2.5 Setting up the PAM authentication

With the Winbind package you also get the Pluggable Authentication Modules (PAM) for Winbind. This module allows the setup of your Linux server to authenticate the users against the Windows Domain controller. This means that you can log on to the Linux server running Winbind with the username from the Windows Active Directory. In Figure 15-1 on page 357 we show how this process is done.

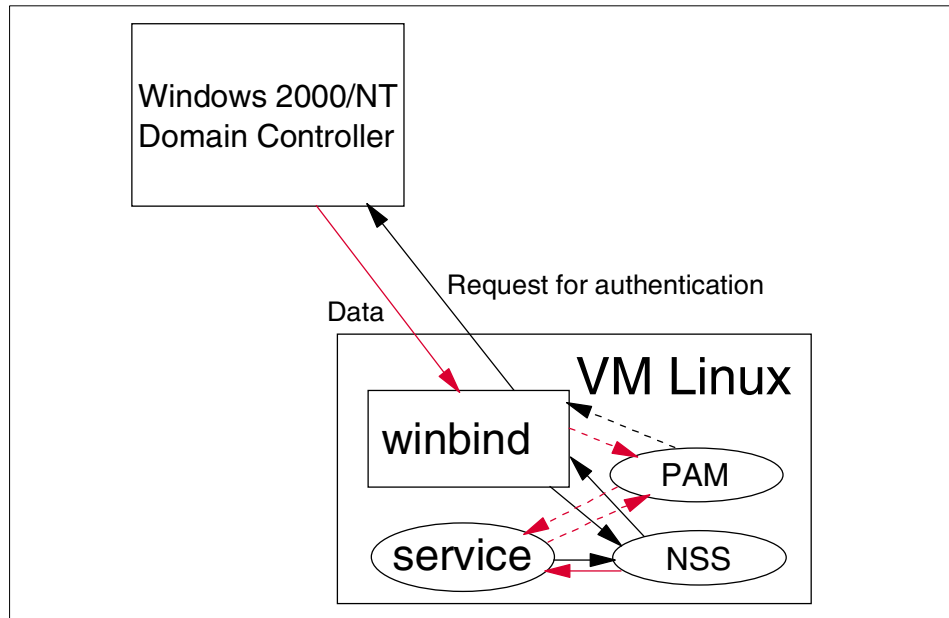


Figure 15-1 Using Winbind for authentication

As you can see in Figure 15-1, the process of authentication is as follows:

1. Service gets the request.
2. Because service is PAM-aware, it passes the request handling to the underlying PAM modules.
3. The PAM module for Winbind passes the request to the Winbind daemon.
4. The Winbind daemon passes the request to the Windows Domain controller.
5. If the user exists in the Windows Active Directory database, the information is passed back to the Winbind daemon.
6. The Winbind daemon then passes this information back to the PAM module.
7. PAM then informs service that this user is allowed to use it.

If the user does not exist, this information comes back to the service and service denies the access.

To achieve this, modify the file for the service which you want to access with this user ID. In SuSE 7.2, these files are located in the `/etc/pam.d` directory as shown in the following:

```
# ls -l /etc/pam.d/
total 92
drwxr-xr-x  2 root  root    4096 Aug  2 17:50 .
```

```

drwxr-xr-x 38 root    root    4096 Aug  7 09:52 ..
-rw-r--r--  1 root    root    305 Jun 17 22:23 chfn
-rw-r--r--  1 root    root    305 Jun 17 22:23 chsh
-rw-r--r--  1 root    root    631 Jun 18 01:03 ftp
-rw-r--r--  1 root    root     95 Jun 18 06:30 imap
-rw-r--r--  1 root    root    749 Aug  3 11:37 login
-rw-r--r--  1 root    root    623 Aug  2 17:50 login.org
-rw-r--r--  1 root    root    259 Jun 18 01:04 netatalk.pamd
-rw-r--r--  1 root    root    517 Jun 17 20:33 other
-rw-r--r--  1 root    root    305 Jun 17 22:23 passwd
-rw-r--r--  1 root    root     95 Jun 18 06:30 pop
-rw-r--r--  1 root    root    311 Jun 18 01:23 ppp
-rw-r--r--  1 root    root    322 Jun 18 12:34 proftpd
-rw-r--r--  1 root    root    263 Jun 17 22:11 rexec
-rw-r--r--  1 root    root    455 Jul 18 09:38 rlogin
-rw-r--r--  1 root    root    292 Jun 17 22:11 rsh
-rw-r--r--  1 root    root    216 Aug  3 13:59 samba
-rw-r--r--  1 root    root    439 Aug  2 17:50 sshd
-rw-r--r--  1 root    root    352 Jun 17 22:36 su
-rw-r--r--  1 root    root    108 Jun 17 23:05 su1
-rw-r--r--  1 root    root     60 Jun 17 23:16 sudo
-rw-r--r--  1 root    root    318 Jun 17 23:39 xdm

```

1. To allow use of the Windows Active Directory user names to log on to your Linux system, modify the `/etc/pam.d/login` file as shown in the following:

```

# cat /etc/pam.d/login
#%PAM-1.0
auth    required    /lib/security/pam_nologin.so
auth    required    /lib/security/pam_env.so
auth    required    /lib/security/pam_mail.so
auth    sufficient  /lib/security/pam_winbind.so
account required    /lib/security/pam_unix.so      audit
account required    /lib/security/pam_winbind.so
password required   /lib/security/pam_pwcheck.so   nullok
password required   /lib/security/pam_unix.so      nullok \
    use_first_pass use_authtok
password required   /lib/security/pam_winbind.so
session required    /lib/security/pam_unix.so
session required    /lib/security/pam_limits.so

```

Now you can try to log on to the Linux server running the Winbind daemon using the Windows Active Directory account. The following was our successful logon:

```

# telnet vmlinux8
Trying 9.12.6.72...
Connected to vmlinux8.
Escape character is '^]'.
Welcome to SuSE Linux 7.2 (S390) - Kernel 2.4.5 (2).

```

```
vmlinux8 login: ITS0+ivo
Password:
Last login: Fri Aug  3 17:15:32 from localhost
Have a lot of fun...
No directory /home/ITS0/ivo!
Logging in with home = "/".
```

15.3 Replacing Microsoft Exchange Server

An Internet Services Provider has to offer at least a basic mail serving functionality to its customers. Once messages have been delivered into the mailbox of the recipient's message store, the recipient needs message access methods to retrieve and work with the messages. These mail servers usually are based on the simple Post Office Protocol, version 3 (POP3). POP3 treats the message store as a single in-box. The user agent can retrieve and delete messages from this in-box. Once messages are retrieved and deleted from the POP3 server, it is the user agent's responsibility, if necessary, to retain messages in some local message store. While a POP3 client can leave mail on the server (by not deleting it), the POP3 protocol lacks mechanisms to categorize, file, or search the mail, so the POP3 server message store can quickly become unmanageable. Also, most large-scale POP3 servers enforce a storage limit, refusing to accept new mail for a user whose limit has been exceeded.

Thus, the POP3 model strongly encourages the complete transfer of mail to the client, where a well-designed client can provide many more capabilities to the user. This has the advantage that the communication with the server is simple, but it has the disadvantage that the user cannot conveniently use more than one computer to read mail: the mail remains on whichever computer the user reads it.

Enterprise customers, which are thinking about consolidating their servers on Linux on zSeries, usually are using more complex mail serving applications, based on or at least supporting the Internet Mail Access Protocol, version 4 (IMAP4). This newer access protocol defines a much richer message store, allowing mail to be stored in multiple mailboxes. A rich set of message and mailbox manipulation functions exists. While a POP3 message can be handled only as a single block, IMAP4 allows access to individual parts of the message. Provisions exist to allow message stores to be replicated to a local store (and resynchronized later) for the mobile user. The IMAP4 model, in contrast to the POP3 model, involves storing mail on the server, where it may be accessed by any client, and using the client's storage only for caching messages for efficiency or for traveling.

The standards described so far allow messages to be transmitted through the Internet, but only “in the clear”. Features such as authentication and encryption are needed to make message transmission secure. Authentication allows messages to be signed, so the recipient can confirm that the sender is the person claimed. Encryption allows data to be sent in such a fashion that only a recipient with a key can decrypt the data. For e-mail, directory services are needed to access user information, such as a given user's e-mail address. Lightweight Directory Access Protocol (LDAP), is the standard that describes how to access directory data. Directory services will play an even greater role for storing and accessing public keys to enable secure messaging.

Earlier e-mail systems were developed for a homogeneous group of users on a single network. They typically have a large set of features allowing the creation and manipulation of compound documents. Their delivery systems often support guaranteed deliveries and receipt notifications. Additional integrated functions for calendars and schedules are not uncommon. On the other hand, they often do not scale well to large user communities, because they were developed for a small, homogeneous domain. They cannot exchange mail with other systems except through specially designed gateways, which lose information in the process of converting between mail formats.

For these reasons, widely accepted groupware and e-mail systems (such as Microsoft Exchange or Lotus Notes) are now designed to also support the common Internet standards like POP3 and IMAP4. Of course, the rich document layout offered, for example, by Lotus Notes is lost when sending an e-mail to a server that is not of the same kind, and advanced functions like calendars can only be used between two Lotus Notes servers. But the pure mailing functionality is still sustained, even when communicating with simple POP3 servers.

So a mail serving application on a Linux server running on zSeries is expected to at least support the IMAP4 protocol, and to communicate with Lotus Notes or Exchange clients. Additionally it is preferred that a functionality comparable to those of the major groupware applications is offered. Various applications are available that meet these requirements, some commercial, some open source. We will have a closer look at one example of each group.

Naturally, from an IBM point of view, Lotus Domino is the recommended application for mail serving and collaboration. However, although there is an Intel-based Linux version, until now no Linux version for zSeries is available, so we will not discuss this software now.

15.3.1 The Bynari Insight Server

In April 2001, Bynari Inc. announced that its messaging and collaboration product, Insight Server, runs on the IBM zSeries and S/390 under the Linux operating system ports for those platforms. See also:

<http://www.bynari.net>

Insight is a commercial UNIX/Linux collaboration tool for enterprises. It was developed due to the computer industry's need of a UNIX/Linux client for MS Exchange with Outlook functionality. Insight bridges the gap between Outlook users and UNIX/Linux workstations, allowing them to collaborate in a seamless fashion. It enables a number of messaging protocols to communicate so that various platforms can work together—the way systems designers intended.

Bynari began offering Insight Server as an entry level platform for small to medium size businesses wanting functionality found in Microsoft Exchange. Soon, demand for larger user populations began, asking for more robust hardware platforms. Having already scaled Insight Server for Compaq's non-stop clusters, Bynari's developers began looking at IBM mainframes as the next step on its product roadmap.

Originally designed as a platform for the large populations of Microsoft Outlook users not connected to Exchange, Insight Server has also proven its value as a cost effective replacement for Exchange. Insight Server supports IMAP, POP3, SMTP, and LDAP protocols for numerous e-mail clients including Outlook, Outlook Express, Eudora and Netscape Messenger. Additionally, Insight Server offers meeting management, shared calendars and folders, and other collaboration functions for workgroup oriented clients including Microsoft Outlook and Bynari's own Insight client.

15.3.2 The Cyrus IMAP server

The Cyrus IMAP server, a scalable enterprise mail system designed for use in small to large enterprise environments using standards-based technologies, is developed by the Cyrus Project at Carnegie Mellon University. It provides access to personal mail and system-wide bulletin boards through the IMAP protocol. Documentation is provided at :

<http://asg2.web.cmu.edu/cyrus>

The software can be downloaded from:

<ftp://ftp.andrew.cmu.edu/pub/cyrus-mail>

A full Cyrus IMAP implementation allows a seamless mail and bulletin board environment to be set up across multiple servers. It differs from other IMAP server implementations in that it is run on “sealed” servers, where users are not normally permitted to log in. The mailbox database is stored in parts of the file system that are private to the Cyrus IMAP system. All user access to mail is through software using the IMAP, POP3, or KPOP protocols. The private mailbox database design gives the server large advantages in efficiency, scalability, and in the ease of administration. Multiple concurrent read/write connections to the same mailbox are permitted. The server supports access control lists on mailboxes and storage quotas on mailbox hierarchies.

A brief description of how to install the Cyrus IMAP server in a Linux environment, together with Sendmail and OpenLDAP, is provided in “The Exchange Server Replacement HOWTO” by Curt Johnson, and in the “Cyrus IMAP HOWTO” by Aurora Skarra-Gallagher, both of which can be found at:

<http://www.linuxdoc.org>

15.4 Using AFS in an enterprise environment

In this section we talk about the Andrew File System (AFS) and how it can participate and be used in an IT VM Linux environment. We will also outline the installation instructions for setting up an AFS server and client in a VM Linux environment. In our test environment we used OpenAFS, which was donated by IBM from its version of commercial AFS implementation. IBM branched the source of the AFS product, and made a copy of the source available for community development and maintenance.

15.4.1 What is AFS

AFS makes it easy for people to work together on the same files, no matter where the files are located. AFS users do not have to know which machine is storing a file, and administrators can move files from machine to machine without interrupting user access. Users always identify a file by the same path name and AFS finds the correct file automatically, just as happens in the local file system on a single machine. While AFS makes file sharing easy, it does not compromise the security of the shared files. It provides a sophisticated protection scheme.

AFS uses a client/server computing model. In client/server computing, there are two types of machines: Server machines store data and perform services for client machines; client machines perform computations for users and access data and services provided by server machines. Some machines act as both client and server. In most cases, you work on a client machine, accessing files stored on a file server machine.

15.4.2 Building OpenAFS

You can get the latest OpenAFS from:

<http://www.openafs.org>

In our test environment we used OpenAFS version 1.1.1. After downloading the package, you can unwind it with the command:

```
# tar xzf openafs-1.1.1-src.tar.bz2
```

Then move to the source directory and run the configuration program with the commands:

```
# cd openafs-1.1.1
# mv config.sub config.sub.org
# cp /usr/share/automake/config.sub .
# ./configure --with-afs-sysname=s390_linux24 \
  --with-linux-kernel-headers=/usr/src/linux
```

Important: You have to install the kernel source before compiling. In our example we installed the source in `/usr/src/linux`.

Compile the package with the command:

```
# make
```

After compilation, all the binary and configuration files reside in the `openafs-1.1.1/s390_linux24` directory. For a minimal configuration of your AFS system, you need to install at least one server. This server will then act as:

- ▶ File server machine
- ▶ Database server machine
- ▶ Binary distribution machine
- ▶ System control machine

15.4.3 Installing OpenAFS

To install and configure OpenAFS on the VMLinux server, follow the following steps.

Creating AFS directories

Create the AFS directories with the commands:

```
# mkdir /usr/afs
# mkdir /usr/vice
# mkdir /usr/vice/etc
```

Loading AFS modules into the kernel

1. Change to the directory as indicated (assuming that /usr/src/openafs-1.1.1 is the source code directory):

```
# cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.client/usr/vice/etc
```

2. Copy the AFS kernel library files to the local /usr/vice/etc/modload directory with the command:

```
# cp -rp modload /usr/vice/etc
```

3. Copy the initialization scripts to the local directory for initialization files (in our example, /etc/init.d):

```
# cp -p afs.rc /etc/init.d/afs
```

4. Run the AFS initialization script to load the AFS extensions into the kernel:

```
# /etc/init.d/afs start
```

Configuring server partitions

Each AFS file server must have at least one partition or logical volume dedicated to storing AFS volumes. Each partition is mounted on /vicep xx , where xx is one or two lowercase letters. The /vicep xx directories must reside in the machine's root directory. In our example, we selected the /dev/dasd/0209/part1 as the partition for AFS file serving. Follow these steps to configure the partitions:

1. Create the mount directory:

```
# mkdir /vicepa
```

2. Create the file system on the partition, add an entry to the /etc/fstab file, and mount the partition with the command:

```
# mke2fs /dev/dasd/0209/part1 -b 4096
```

An example of the /etc/fstab file after adding the mounting entry is as follows:

```
# cat /etc/fstab
/dev/dasd/0204/part1 swap swap defaults 0 0
/dev/dasd/0201/part1 / ext2 defaults 1 1
/dev/dasd/0209/part1 /vicepa ext2 defaults 0 2
proc /proc proc defaults 0 0
# mount /vicepa
```

Enabling AFS Login

AFS also provides the PAM authentication for PAM-capable clients. The following steps show you how to set up the configuration for each service for which you wish to use AFS authentication. You can skip this section if you do not want to use client functionality on this server.

1. Copy the PAM libraries into the /lib/security directory:

If you plan to use the AFS Authentication Server (**kaserver** process):

```
# cd /lib/security
# cp /usr/src/openafs-1.1.1/s390_linux24/dest/lib/pam_afs.so.1 .
# ln -s pam_afs.so.1 pam_afs.so
```

If you plan to use Kerberos implementation of AFS authentication:

```
# cd /lib/security
# cp /usr/src/openafs-1.1.1/s390_linux24/dest/lib/pam_afs.krb.so.1 .
# ln -s pam_afs.krb.so.1 pam_afs.krb.so
```

2. For each service from `/etc/pam.d`, put the following line into the auth section:

```
auth sufficient /lib/security/pam_afs.so try_first_pass ignore_root
```

Insert this line just after the entries that impose conditions under which you want the service to fail. The `ignore_root` parameter means that the AFS PAM module will ignore the local superuser `root` and also any user with UID 0. Following is our `/etc/pam.d/login` file for login service:

```
# cat /etc/pam.d/login
#%PAM-1.0
auth    required  /lib/security/pam_nologin.so
auth    required  /lib/security/pam_env.so
auth    required  /lib/security/pam_mail.so
auth    sufficient /lib/security/pam_afs.so    try_first_pass ignore_root
auth    requisite /lib/security/pam_unix.so    nullok #set_secrcp
account required  /lib/security/pam_unix.so
password required /lib/security/pam_pwcheck.so nullok
password required /lib/security/pam_unix.so nullok use_first_pass \
    use_authtok
session required  /lib/security/pam_unix.so    none # debug or trace
session required  /lib/security/pam_limits.so
```

Starting the BOS Server

The BOS (Basic OverSeer) Server is used to monitor and control other AFS server processes on its server. Follow these steps to install and start the BOS Server:

1. Copy the files:

```
# cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.server/usr/afs
# cp -rp * /usr/afs
```

2. Start the BOS Server, include the `-noauth` flag to disable authorization checking (authentication is not running yet):

```
# /usr/afs/bin/bosserver -noauth &
```

3. Verify that the BOS server created `/usr/vice/etc/ThisCell` and `/usr/vice/etc/CellServDB` as symbolic links to the corresponding files in the `/usr/afs/etc` directory:

```
# ls -l /usr/vice/etc
```

If the links do not exist, create them with the commands:

```
# cd /usr/vice/etc
# ln -s /usr/afs/etc/ThisCell ThisCell
# ln -s /usr/afs/etc/CellServDB CellServDB
```

Defining cell name and membership for the server process

Here we assign the cell name. You should know that changing the name is very difficult, so you should plan the name carefully. Usually, the cell name is the same as the name of the Internet domain you are using. There are two important restrictions: the name cannot include uppercase letters or more than 64 characters.

Use the following steps to set the cell name:

1. Change to the directory with the AFS programs with the command:

```
# cd /usr/afs/bin
```

2. With the **bos setcellname** command, set the cell name:

```
# ./bos setcellname vmlinux8.itso.ibm.com itso.ibm.com -noauth
```

As you can see, we issued **bos setcellname** with two parameters:

- machine name = vmlinux8.itso.ibm.com
- cell name = itso.ibm.com

3. Verify that the server you are installing is now registered as the cell's first database server:

```
# ./bos listhosts vmlinux8.itso.ibm.com -noauth
Cell name is itso.ibm.com
Host 1 is vmlinux8
```

Starting the database server process

Now we create four database server processes in the `/usr/afs/local/BosConfig` file and start them running. They run on the database server machine only.

- ▶ The Authentication Server (the **kaserver** process) maintains the Authentication Database.
- ▶ The Backup Server (the **busserv** process) maintains the Backup Database.
- ▶ The Protection Server (the **ptserver** process) maintains the Protection Database.
- ▶ The Volume Location (VL) Server (the **v1server** process) maintains the Volume Location Database (VLDB).

Note: AFS's authentication and authorization software is based on algorithms and other procedures known as *Kerberos*, as originally developed by Project Athena at the Massachusetts Institute of Technology. Some cells choose to replace the AFS Authentication Server and other security-related protocols with Kerberos as obtained directly from Project Athena or other sources. If you wish to do this, contact the AFS Product Support group now to learn about necessary modifications to the installation.

Follow these steps to create these server processes (we assume that you are in the `/usr/afs/directory`):

1. Start the Authentication Server:

```
# ./bos create vmlinux8.itso.ibm.com kaserver simple \  
/usr/afs/bin/kaserver -cell itso.ibm.com -noauth
```

2. Start the Backup Server:

```
# ./bos create vmlinux8.itso.ibm.com buserver simple \  
/usr/afs/bin/buserver -cell itso.ibm.com -noauth
```

3. Start the Protection Server:

```
# ./bos create vmlinux8.itso.ibm.com ptserver simple \  
/usr/afs/bin/ptserver -cell itso.ibm.com -noauth
```

4. Start the VL Server:

```
# ./bos create vmlinux8.itso.ibm.com vlserver simple \  
/usr/afs/bin/vlserver -cell itso.ibm.com -noauth
```

Initializing cell security

Now we initialize the cell's security mechanisms. We begin by creating two initial entries in the Authentication Database:

► A generic administrative (in our example we call it `admin`)

After installation, all administrators can use this account or you can create a separate account for each of them.

► The entry for the AFS server process, called `afs`

There are no logons under this user ID, but Authentication Server's Ticket Granting (TGS) module uses the associated key to encrypt the server tickets that it grants to AFS clients.

In the following steps we show how to create these two entries. Keep in mind that this process does not configure all of the security mechanisms related to the AFS Backup System. To do this you need to refer to the *IBM AFS Administration Guide* on the Web at:

<http://oss.software.ibm.com/developerworks/opensource/afs/docs.html>

1. Change to the directory with the AFS programs with the command:

```
# cd /usr/afs/bin
```

2. Enter the kas interactive mode with the -noauth option, because the server is in no-authorization checking mode. Then create admin and afs entries:

```
# ./kas -cell itso.ibm.com -noauth
ka> create afs
initial_password:
Verifying, please re-enter initial_password:
ka> create admin
initial_password:
Verifying, please re-enter initial_password:
```

3. Examine the afs entry checksum:

```
ka> examine afs
```

```
User data for afs
key (0) cksum is 824768179, last cpw: Wed Aug  8 09:42:29 2001
password will never expire.
An unlimited number of unsuccessful authentications is permitted.
entry never expires. Max ticket lifetime 100.00 hours.
last mod on Wed Aug  8 09:42:29 2001 by <none>
permit password reuse
```

4. Turn on the ADMIN flag for the admin entry and then examine the entry to verify that the admin flag appears:

```
ka> setfields admin -flags admin
ka> examine admin
User data for admin (ADMIN) - you can see the ADMIN flag is present
key (0) cksum is 824768179, last cpw: Wed Aug  8 09:42:39 2001
password will never expire.
An unlimited number of unsuccessful authentications is permitted.
entry never expires. Max ticket lifetime 25.00 hours.
last mod on Wed Aug  8 09:47:32 2001 by <none>
permit password reuse
```

5. Quit the kas server:

```
ka> quit
```

6. Now we need to add admin to the /usr/afs/etc/UserList file, to enable admin to issue privileged **bos** and **vos** commands:

```
# ./bos adduser vmlinux8.itso.ibm.com admin -cell itso.ibm.com -noauth
```

7. Next define the AFS server encryption key in `/usr/afs/etc/KeyFile`:

```
# ./bos addkey vmlinux8.itso.ibm.com -kvno 0 -cell itso.ibm.com -noauth
input key:
Retype input key:
```

For the input key, type in the password you used for creating the afs entry in step 1.

8. Verify that the checksum for the new key in the Keyfile is the same as the checksum defined for the key Authentication Database's afs entry that we displayed in step 2.:

```
# ./bos listkey vmlinux8.itso.ibm.com -cell itso.ibm.com -noauth
key 0 has cksum 824768179
Keys last changed on Wed Aug  8 12:03:59 2001.
All done.
```

As you can see in our example, the keys are the same.

9. Now we need to create the Protection Database Entry for the admin user. By default, the Protection Server assigns AFS UID 1 to the admin user, because it is the first entry you are creating. If the local password file (`/etc/passwd` or equivalent) already has an entry for admin that assigns it a UNIX UID other than 1, it is best to use the `-id` argument on the `pts createuser` command to make the new AFS UID match the existing UNIX UID. Otherwise, it is best to accept the default. In our example we already have the admin user on the system with a UID of 501:

```
# ./pts createuser -name admin -cell itso.ibm.com -id 501 -noauth
User admin has id 501
```

10. Next add the admin user to the `system:administrators` group and then check if this was successful:

```
# ./pts adduser admin system:administrators -cell itso.ibm.com -noauth
# ./pts membership admin -cell itso.ibm.com -noauth
Groups admin (id: 1) is a member of:
system:administrators
```

11. Now we need to restart the bos server with the `-all` flag to restart the database server processes, so that they start using the new server encryption key:

```
# ./bos restart vmlinux8.itso.ibm.com -all -cell itso.ibm.com -noauth
```

You can check whether the AFS server processes are running with the command:

```
# ps ax | grep afs
 9050 ?      S        0:00 /usr/afs/bin/bosserver -noauth
11419 ?      S        0:00 /usr/afs/bin/kaserver
11420 ?      S        0:00 /usr/afs/bin/buserver
11421 ?      S        0:00 /usr/afs/bin/ptserver
```

```
11422 ?          S          0:00 /usr/afs/bin/vlserver
```

Starting the file server, volume server, and salvager

To start the **fs** process, which consists of the File Server, Volume Server, and the Salvager (**fileserver**, **volserver** and **salvager** processes), follow these steps:

1. Change to the directory with the AFS programs with the command:

```
# cd /usr/afs/bin
```

2. Create the **fs** process:

```
# ./bos create vmlinux8.itso.ibm.com fs fs /usr/afs/bin/fileserver \  
/usr/afs/bin/volserver /usr/afs/bin/salvager -cell itso.ibm.com -noauth
```

You can verify that the **fs** process has started successfully with the command:

```
# ./bos status vmlinux8.itso.ibm.com fs -long -noauth  
Instance fs, (type is fs) currently running normally.  
Auxiliary status is: file server running.  
Process last started at Wed Aug 8 12:39:05 2001 (2 proc starts)  
Command 1 is '/usr/afs/bin/fileserver'  
Command 2 is '/usr/afs/bin/volserver'  
Command 3 is '/usr/afs/bin/salvager'
```

You can see that in our example the servers are running with no problems.

3. Because this is the first AFS file server in our cell, we need to create the first AFS volume, **root.afs**:

```
# ./vos create vmlinux8.itso.ibm.com /vicepa root.afs -cell itso.ibm.com \  
-noauth  
Volume 536870912 created on partition /vicepa of vmlinux8.itso.ibm.com
```

As you can see, we used our **/vicepa** partition for the **root.afs** AFS volume.

Starting the server portion of the update process

Start the server portion of the Update Server (the **upserver** process) to distribute the contents of directories on this machine to other server machines in the cell. It becomes active when you configure the client portion of the Update Server on additional server machines.

Distributing the contents of its **/usr/afs/etc** directory makes this server the cell's system control machine. The other servers in the cell run the **upclientetc** process (an instance of the client portion of the Update Server) to retrieve the configuration files. Use the **-crypt** argument to the **upserver** initialization command to specify that the Update Server distributes the contents of the **/usr/afs/etc** directory only in encrypted form. Several of the files in the directory, particularly the **KeyFile** file, are crucial to cell security and so must never cross the network unencrypted.

(You can choose not to configure a system control server, in which case you must update the configuration files in each server's `/usr/afs/etc` directory individually. The **bos** commands used for this purpose also encrypt data before sending it across the network.)

Distributing the contents of its `/usr/afs/bin` directory to other servers of its system type makes this server a binary distribution machine. The other servers of its system type run the **upclientbin** process (an instance of the client portion of the Update Server) to retrieve the binaries.

The binaries in the `/usr/afs/bin` directory are not sensitive, so it is not necessary to encrypt them before transfer across the network. Include the `-clear` argument to the upserver initialization command to specify that the Update Server distributes the contents of the `/usr/afs/bin` directory in unencrypted form unless an upclientbin process requests encrypted transfer.

Note that the server and client portions of the Update Server always mutually authenticate with one another, regardless of whether you use the `-clear` or `-crypt` arguments. This protects their communications from eavesdropping to some degree.

Start the Update Server process with the commands:

```
# cd /usr/afs/bin
# ./bos create vmlinux8.itso.ibm.com upserver simple \
  "/usr/afs/bin/upserver -crypt /usr/afs/etc -clear /usr/afs/bin" \
  -cell itso.ibm.com -noauth
```

Starting the Controller for NTPD

Keeping the clocks on all server and client machines in your cell synchronized is crucial to several functions, and in particular to the correct operation of AFS's distributed database technology, Ubik. The time skew can disturb Ubik's performance and cause service outages in your cell.

The AFS distribution includes a version of the Network Time Protocol Daemon (NTPD) for synchronizing the clocks on server machines. If a time synchronization program is not already running on the machine, then in this section you start the **runntp** process to configure NTPD for use with AFS.

Note: Do not run **runntp** process on top of another NTPD or another time synchronization protocol is already running on the machine.

In our example we did not have another time synchronization protocol running, so we decided to use **runntp** from the AFS server:

1. Create the **runntp** process:

If you have a reliable network connection to an outside time source:

```
# ./bos create vmlinux8.itso.ibm.com runntp \  
simple "/usr/afs/bin/runntp hostname+" -cell itso.ibm.com -noauth
```

If you plan to use the local clock as the time source (as we did in our example):

```
# ./bos create vmlinux8.itso.ibm.com runntp \  
simple "/usr/afs/bin/runntp -localclock" -cell itso.ibm.com -noauth
```

If you have a connection to an outside time source, but it is not reliable:

```
# ./bos create vmlinux8.itso.ibm.com runntp \  
simple "/usr/afs/bin/runntp -localclock hostname+" \  
-cell itso.ibm.com -noauth
```

Note: In the OpenAFS version we used, the NTP package was not compiled, because it is obsolete. The clients are getting time from the AFS servers anyway. You should install the NTP server on the AFS server. It is available on the Web at:

<http://www.eecis.udel.edu/~ntp>

Or use any other NTP server. In SuSE 7.2 they have included the XNTP package, which can be used for this purpose.

In our example we deleted the definition for the **runntp** process with the command:

```
# ./bos delete vmlinux8.itso.ibm.com runntp
```

15.4.4 Installing client functionality

The server which we just installed is the AFS file server, database server, system control server, and binary distribution server. Now we need to make this server also the client machine.

Copying client files to the local disk

Before installing and configuring the AFS client, we need to copy the necessary files from the build directory:

```
# cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.client/usr/vice/etc  
# cp -p * /usr/vice/etc  
cp: omitting directory `C'  
cp: omitting directory `modload'  
# cp -rp C /usr/vice/etc
```


Defining cell membership for client processes

Every AFS client machine has a copy of the `/usr/vice/etc/ThisCell` file on its local disk to define the machine's cell membership for the AFS client programs that run on it. The `ThisCell` file you created in the `/usr/afs/etc` directory (in “Defining cell name and membership for the server process” on page 366) is used only by server processes.

Among other functions, the `ThisCell` file on a client machine determines the following:

- ▶ The cell in which users authenticate when they log onto the machine, assuming it is using an AFS-modified login utility
- ▶ The cell in which users authenticate by default when they issue the `klog` command
- ▶ The cell membership of the AFS server processes that the AFS command interpreters on this machine contact by default

//TODO: Should “To define this” be “To define cell membership”?

To define this, remove the symbolic link created in “Starting the BOS Server” on page 365 and create the new `ThisCell` file by copying the server copy of this file from `/usr/afs/etc/ThisCell`. With this you define the same cell for both server and client processes, which gives you the most consistent AFS performance:

```
# cd /usr/vice/etc
# rm ThisCell
# cp /usr/afs/etc/ThisCell ThisCell
```

Creating the client `CellServDB` file

The `/usr/vice/etc/CellServDB` file on a client machine's local disk lists the database server machines for each cell that the local Cache Manager can contact. If there is no entry in the file for a cell, or if the list of database server machines is wrong, then users working on this machine cannot access the cell.

Because the `afsd` program initializes the Cache Manager, it copies the contents of the `CellServDB` file into kernel memory. The Cache Manager always consults the list in kernel memory rather than the `CellServDB` file itself. Between reboots of the machine, you can use the `fs newcell` command to update the list in kernel memory directly.

Follow these steps to create the `CellServDB` file:

1. Remove the symbolic link created in “Starting the BOS Server” on page 365:

```
# cd /usr/vice/etc/
# rm CellServDB
```

2. Create CellServDB with the local cell entry and display it to verify the file:

```
# cat /usr/afs/etc/CellServDB > CellServDB
# cat CellServDB
>itso.ibm.com #Cell name
9.12.6.72 #vmlinux8
```

Configuring the cache

The Cache Manager uses a cache on the local disk or in machine memory to store local copies of files fetched from file server machines. As the **afsd** program initializes the Cache Manager, it sets basic cache configuration parameters according to definitions in the local `/usr/vice/etc/cacheinfo` file.

The file has three fields:

1. The first field names the local directory on which to mount the AFS file space. The conventional location is the `/afs` directory.
2. The second field defines the local disk directory to use for the disk cache. The conventional location is the `/usr/vice/cache` directory, but you can specify an alternate directory if another partition has more space available. There must always be a value in this field, but the Cache Manager ignores it if the machine uses a memory cache.
3. The third field specifies the number of kilobyte (1024 byte) blocks to allocate for the cache.

The values you define must meet the following requirements:

- ▶ On a machine using a disk cache, the Cache Manager expects always to be able to use the amount of space specified in the third field. Failure to meet this requirement can cause serious problems, some of which can be repaired only by rebooting. You must prevent non-AFS processes from filling up the cache partition. The simplest way is to devote a partition to the cache exclusively.
- ▶ The amount of space available in memory or on the partition housing the disk cache directory imposes an absolute limit on cache size.
- ▶ The maximum supported cache size can vary in each AFS release; see the Release Notes for the current version.
- ▶ For a disk cache, you cannot specify a value in the third field that exceeds 95% of the space available on the partition mounted at the directory named in the second field. If you violate this restriction, the **afsd** program exits without starting the Cache Manager and prints an appropriate message on the standard output stream. A value of 90% is more appropriate on most machines. Some operating systems (such as AIX) do not automatically reserve some space to prevent the partition from filling completely; for them, a smaller value (say, 80% to 85% of the space available) is more appropriate.

- ▶ For a memory cache, you must leave enough memory for other processes and applications to run. If you try to allocate more memory than is actually available, the `afsd` program exits without initializing the Cache Manager and produces the following message on the standard output stream:

```
afsd: memCache allocation failure at number KB
```

The number value is how many kilobytes were allocated just before the failure, and so indicates the approximate amount of memory available.

Tip: Disk caches smaller than 10 MB do not perform well, and also memory caches smaller than 5 MB do not perform well. The cache size depends on the number of users using the client machine.

Configuring a disk cache

To configure the disk cache, perform the following steps:

1. Create a local directory for caching with the command:

```
# mkdir /usr/vice/cache
```

2. Create the `cacheinfo` file; in our example, we define a 50 MB disk cache:

```
# echo "/afs:/usr/vice/cache:50000" > /usr/vice/etc/cacheinfo
```

Configuring a memory cache

To configure the memory cache, do the following:

Create the `cacheinfo` file; in our example, we create a 25 MB memory cache:

```
# echo "/afs:/usr/vice/cache:25000" > /usr/vice/etc/cacheinfo
```

Configuring the Cache Manager

By convention, the Cache Manager mounts the AFS file space on the local `/afs` directory. The `afsd` program sets several cache configuration parameters as it initializes the Cache Manager, and starts daemons that improve performance. These options are stored in the `afsd` options file. In the `afs` configuration file there are three predefined cache sizes:

- ▶ `SMALL` is suitable for a small machine that serves one or two users and has approximately 8 MB of RAM and a 20 MB cache.
- ▶ `MEDIUM` is suitable for a medium-sized machine that serves two to six users and has 16 MB of RAM and a 40 MB cache.
- ▶ `LARGE` is suitable for a large machine that serves five to ten users and has 32 MB of RAM and a 100 MB cache.

By default the distributed `afs.conf` file options are set to `MEDIUM`.

Follow these steps to configure the Cache Manager:

1. Create the local directory on which to mount the AFS file space:

```
# mkdir /afs
```

2. Copy the AFS configuration option file to the /etc/sysconfig directory (in the case of the Suse 7.2 distribution we used, you need also to create this directory):

```
# mkdir /etc/sysconfig
# cp /usr/vice/etc/afs.conf /etc/sysconfig/afs
```

3. Edit the /etc/sysconfig/afs file if you want to incorporate any changes:

Change AFS_SERVER=off to AFS_SERVER=on.

In our example we added -nosettime, because this is a file server that is also a client. This flag prevents the machine from picking up the file server in the cell as its source for the correct time.

There are also two more parameters you can use:

- -memcache specifies that the machine will use a memory cache.
- -verbose specifies that the trace of Cache Manager's initialization will be displayed on the standard output stream.

Example 15-8 shows an example of the AFS configuration file.

Example 15-8 Our /etc/sysconfig/afs file

```
#!/bin/sh
# Copyright 2000, International Business Machines Corporation and others.
# All Rights Reserved.
#
# This software has been released under the terms of the IBM Public
# License. For details, see the LICENSE file in the top-level source
# directory or online at http://www.openafs.org/d1/license10.html

# Configuration information for AFS client

# AFS_CLIENT and AFS_SERVER determine if we should start the client and or
# the bossserver. Possible values are on and off.
AFS_CLIENT=on
AFS_SERVER=on

# AFS client configuration options:
LARGE="-stat 2800 -dcache 2400 -daemons 5 -volumes 128"
MEDIUM="-stat 2000 -dcache 800 -daemons 3 -volumes 70 -nosettime"
SMALL="-stat 300 -dcache 100 -daemons 2 -volumes 50"
OPTIONS=$MEDIUM

# Set to "-verbose" for a lot of debugging information from afsd. Only
# useful for debugging as it prints _a lot_ of information.
```

```

VERBOSE=

# OPTIONS are the options passed to afsd.
OPTIONS="$OPTIONS $VERBOSE"

# Sample server preferences function. Set server preferences using this.
# afs_serverprefs() {
#   /usr/afsws/etc/fs setserverprefs <host> <rank>
#}

# Either the name of an executable script or a set of commands go here.
# AFS_POST_INIT=afs_serverprefs
AFS_POST_INIT=

```

15.4.5 Completing the installation of the first AFS server

The machine is now configured as an AFS file server and client machine. In this final phase of the installation, we initialize the Cache Manager and then create the upper levels of AFS file space, among other procedures.

Verifying the AFS initialization script

Follow these step to complete this task:

1. Shut down the **bos** server:

```
# /usr/afs/bin/bos shutdown vmlinux8.itso.ibm.com -wait
```

2. Issue the **ps** command to learn the **bosserv** process's ID and then kill that process:

```
# ps ax | grep bosserv
9050 ?      S      0:00 /usr/afs/bin/bosserv -noauth
# kill -9 9050
```

3. Reboot the VM Linux server, log on as root, and then start the AFS initialization script and wait for the message that all daemons are started:

```
# cd /
# shutdown -h now
```

```
login: root
password: root_password
```

```
# /etc/init.d/afs start
Starting AFS services.....
afsd: All AFS daemons started.
```

4. As a basic test of correct AFS functioning, try to authenticate as admin:

```
# /usr/afs/bin/klog admin
Password: admin_passwd
```
5. Issue the **tokens** command to verify that the **klog** command was successful:

```
# /usr/afs/bin/tokens
```

Tokens held by the Cache Manager:

```
User's (AFS ID 501) tokens for afs@itso.ibm.com [Expires Aug  9 18:09]
--End of list--
```
6. Issue the **bos status** command to verify that the output of each process reads “currently running normally”:

```
# /usr/afs/bin/bos status vmlinux8.itso.ibm.com
Instance kaserver, currently running normally.
Instance buserver, currently running normally.
Instance ptserver, currently running normally.
Instance vlserver, currently running normally.
Instance fs, currently running normally.
Auxiliary status is: file server running.
Instance upserver, currently running normally.
```
7. Check the volumes with the command:

```
# cd /
# /usr/afs/bin/fs checkvolumes
All volumeID/name mappings checked.
```

Activating the AFS initialization script

After confirming that the AFS initialization script works correctly, we take the action necessary to have it run automatically at each reboot.

On the SuSE 7.2 distribution you can do this by creating two symbolic links into run level 3, which is the default run level used:

```
# cd /etc/init.d/rc3.d/
# ln -s ../afs S99afs
# ln -s ../afs K01afs
```

Configuring the top levels of the AFS file space

If you have not previously run AFS in your cell, you now configure the top levels of your cell's AFS file space. We created the root.afs volume in “Starting the file server, volume server, and salvager” on page 370. Now we set the Access Control List (ACL) on the /afs directory. Creating, mounting, and setting the ACL are the three steps required when creating any volume.

After setting the ACL on the root.afs volume, create your cell's root.cell volume, mount it as a subdirectory of the /afs directory, and set the ACL. Create both a read/write and a regular mount point for the root.cell volume. The read/write mount point enables you to access the read/write version of replicated volumes when necessary. Creating both mount points essentially creates separate read-only and read-write copies of your file space, and enables the Cache Manager to traverse the file space on a read-only path or read/write path as appropriate.

Then replicate both the root.afs and root.cell volumes. This is required if you want to replicate any other volumes in your cell, because all volumes mounted above a replicated volume must themselves be replicated in order for the Cache Manager to access the replica.

When the root.afs volume is replicated, the Cache Manager is programmed to access its read-only version (root.afs.readonly) whenever possible. To make changes to the contents of the root.afs volume (when, for example, you mount another cell's root.cell volume at the second level in your file space), you must mount the root.afs volume temporarily, make the changes, release the volume, and remove the temporary mount point.

To set up the ACL for the /afs directory, follow these steps:

1. Edit the ACL on the /afs directory with **fs setacl**. We add the entry that grants the l (lookup) and r (read) permissions to the system:anyuser group. With this we enable all AFS users who can reach your cell to traverse through the directory. If you prefer to enable access only to locally authenticated users, substitute the system:authuser group.

Note: By default the **system:administrators** have all seven rights. This is the default entry that AFS places on every new volume's root directory.

```
# /usr/afs/bin/fs setacl /afs system:anyuser rl
```

2. Create the root.cell volume and then mount it in the subdirectory of /afs, where it serves as the root of our cell's local AFS file space. At the end we create an ACL entry for the system:anyuser group:

```
# /usr/afs/bin/vos create vmlinux8.itso.ibm.com /vicepa root.cell
Volume 536870915 created on partition /vicepa of vmlinux8.itso.ibm.com
/usr/afs/bin/fs mkmount /afs/itso.ibm.com root.cell
/usr/afs/bin/fs setacl /afs/itso.ibm.com system:anyuser rl
```

3. To shorten the path names for users in the local cell we create a symbolic link to a shortened cell name:

```
# cd /afs
# ln -s itso.ibm.com itso
```

```
# ls -l
total 8
drwxrwxrwx  2 root    root      2048 Aug 8 20:44 .
drwxr-xr-x  22 root    root      4096 Aug 8 20:29 ..
lrwxr-xr-x   1 admin   root       12 Aug 8 20:44 itso -> itso.ibm.com
drwxrwxrwx  2 root    root      2048 Aug 8 20:41 itso.ibm.com
```

4. Create a read/write mount point for the root.cell volume (we created the regular mount point in step 2). By convention, the read/write mount point begins with a period:

```
# cd /usr/afs/bin
# ./fs mkmount /afs/.itso.ibm.com root.cell -rw
```

5. Define the replication site for the root.afs and root.cell volumes:

```
# ./vos addsite vmlinux8.itso.ibm.com /vicepa root.afs
Added replication site vmlinux8.itso.ibm.com /vicepa for volume root.afs
# ./vos addsite vmlinux8.itso.ibm.com /vicepa root.cell
Added replication site vmlinux8.itso.ibm.com /vicepa for volume root.cell
```

6. Verify that the Cache Manager can access both the root.afs and root.cell volumes before you attempt to replicate them:

```
# ./fs examine /afs
Volume status for vid = 536870912 named root.afs
Current disk quota is 5000
Current blocks used are 5
The partition has 6737196 blocks available out of 6737440

# ./fs examine /afs/itso.ibm.com
Volume status for vid = 536870915 named root.cell
Current disk quota is 5000
Current blocks used are 2
The partition has 6737196 blocks available out of 6737440
```

7. Release the replica of root.afs and root.cell you created in the previous steps:

```
# ./vos release root.afs
Released volume root.afs successfully
# ./vos release root.cell
Released volume root.cell successfully
```

8. Check the volumes to force the Cache Manager to notice that you have released read-only versions of the volumes, then examine the volumes again:

```
# ./fs checkvolumes
All volumeID/name mappings checked.
# ./fs examine /afs
Volume status for vid = 536870912 named root.afs
Current disk quota is 5000
Current blocks used are 5
```



```
The partition has 6737248 blocks available out of 6737440
```

```
# ./fs examine /afs/itso.ibm.com
Volume status for vid = 536870915 named root.cell
Current disk quota is 5000
Current blocks used are 2
The partition has 6737248 blocks available out of 6737440
```

Storing AFS binaries in AFS

In the conventional configuration, you make AFS client binaries and configuration files available in the subdirectories of the `/usr/afsws` directory on client machines (afsws is an acronym for AFS workstation). You can conserve local disk space by creating `/usr/afsws` as a link to an AFS volume that houses the AFS client binaries and configuration files for this system type.

In this section we create the necessary volumes. The conventional location to which to link `/usr/afsws` is `/afs/cellname/sysname/usr/afsws`.

Follow these steps to complete this task:

1. Create volumes for storing the AFS client binaries for this system type. In our example we create the volumes `s390_linux24`, `s390_linux24.usr` and `s390_linux.usr.afsws`:

```
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24
Volume 536870918 created on partition /vicepa of vmlinux8.itso.ibm.com
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24.usr
Volume 536870921 created on partition /vicepa of vmlinux8.itso.ibm.com
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24.usr.afsws
Volume 536870924 created on partition /vicepa of vmlinux8.itso.ibm.com
```

2. Now mount those volumes:

```
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24 -vol s390_linux24
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24/usr \
  -vol s390_linux24.usr
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24/usr/afsws \
  -vol s390_linux24.usr.afsws
```

3. Release the new `root.cell` replica and check the volumes so the local Cache Manager can access them:

```
# ./vos release root.cell
Released volume root.cell successfully
# ./fs checkvolumes
All volumeID/name mappings checked.
```

4. Grant the lookup and read access to the `system:anyuser` group on each new directory's ACL:

```
# cd /afs/.itso.ibm.com/s390_linux24
```

```
# /usr/afs/bin/fs setacl -dir . usr usr/afsws -acl system:anyuser rl
```

5. We set an unlimited quota on the s390_linux.usr.afsws volume so we do not have any problems copying appropriate files for distribution, without exceeding the quota:

```
# /usr/afs/bin/fs setquota /afs/.itso.ibm.com/s390_linux24/usr/afsws 0
```

6. Copy the necessary files:

```
# cd /afs/.itso.ibm.com/s390_linux24/usr/afsws/  
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/bin .  
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/etc .  
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/include .  
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/lib .
```

7. Set the permissions to allow system:authuser to look up and read on directories /etc, /include, and /lib and deny the access for the group system:anyuser to those directories. The group system:anyuser still needs the lookup and read permissions to the /bin directory to enable unauthenticated users to access the **klog** binary:

```
# cd /afs/.itso.ibm.com/s390_linux24/usr/afsws  
# /usr/afs/bin/fs setacl -dir etc include lib -acl system:authuser rl \  
system:anyuser none
```

8. Create the symbolic link /usr/afsws on the local disk to the directory /afs/itso.ibm.com/@sys/usr/afsws:

```
# ln -s /afs/itso.ibm.com/@sys/usr/afsws /usr/afsws
```

Tip: If you do not want to type the whole path to AFS suite commands, such as **fs**, you should include the following paths to the PATH environment variable: /usr/afsws/bin and /usr/afsws/etc.

Congratulations! You have just completed the installation of the AFS server on your VM Linux.

15.4.6 Installing clients on other servers

In this section we describe how to install the AFS client on the server from which you want to access the AFS files.

Transfer the installation files to the client server

Follow these steps to transfer the installation files from the server where you compiled the OpenAFS package to the client server. In our example, the server with compiled packages was vmlinux8.itso.ibm.com.

1. Create the gz package:

```
# cd usr/src/openafs-1.1.1/s390_linux24/
```

```
# tar -c -z dest > s390_linux24afs.gz
```

2. Transfer the file to the client computer with ftp; for example, create the directory and unpack into this directory:

```
# mkdir /afsinstall
# cd /afsinstall/
# ftp vmlinux8.itso.ibm.com
Connected to vmlinux8.itso.ibm.com.
...
ftp> cd /usr/src/openafs-1.1.1/s390_linux24
ftp> bin
ftp> get s390_linux24afs.gz
...
11253760 bytes received in 00:00 (24.35 MB/s)
ftp> bye
221 Goodbye.
# tar zxf s390_linux24afs.gz
```

Creating AFS directories on the local disk

Create the directories for holding binary and configuration files with the commands:

```
# mkdir /usr/vice
# mkdir /usr/vice/etc
```

Loading AFS into the kernel

Follow these steps to load the AFS modules into the kernel:

1. Copy the AFS kernel files into the `/usr/vice/etc/modload` directory:

```
# cd /afsinstall/dest/root.client/usr/vice/etc/
# cp -rp modload /usr/vice/etc
```

2. Copy the initialization script and start it:

```
# cp -p afs.rc /etc/init.d/afs
# /etc/init.d/afs start
Starting AFS services.....
```

Enabling AFS login

Follow the instructions in “Enabling AFS Login” on page 364 to enable AFS login on the client server. Keep in mind that the installation files are now in the `/afsinstall` directory, not in `/usr/src/openafs-1.1.1/s390_linux24`.

Loading and creating client files

Follow these steps to complete this task:

1. Copy the client files:

```
# cd /afsinstall/dest/root.client/usr/vice/etc/
# cp -p * /usr/vice/etc
cp: omitting directory `C'
cp: omitting directory `modload'
# cp -rp C /usr/vice/etc
```

2. Create the `/usr/vice/etc/ThisCell` file. With this cell you define the membership of this client server:

```
# echo "itso.ibm.com" > /usr/vice/etc/ThisCell
```

3. From your AFS server, copy `CellServDB` to `/usr/vice/etc/CellServDB`.

Configuring the cache

We already explained, in “Configuring the cache” on page 374, how the cache works and what the needed parameters are. Here we just outline the procedure to implement this on the client machine. In our example, we use the disk cache:

1. Create the directory for the cache:

```
# mkdir /usr/vice/cache
```

2. Create the `cacheinfo` file with a `cachesize` of 25000 KB:

```
# echo "/afs:/usr/vice/cache:25000" > /usr/vice/etc/cacheinfo
```

Configuring the Cache Manager

We explained the function of the Cache Manager in “Configuring the Cache Manager” on page 375. Follow these steps to set up the Cache Manager on the client server:

1. Create the directory for the cache:

```
# mkdir /afs
```

2. Copy the configuration file:

```
# mkdir /etc/sysconfig
# cp /usr/vice/etc/afs.
# cp /usr/vice/etc/afs.conf /etc/sysconfig/afs
```

3. Edit the configuration file to suit your needs. Following is our example file:

```
#!/bin/sh
# Copyright 2000, International Business Machines Corporation and others.
# All Rights Reserved.
#
# This software has been released under the terms of the IBM Public
# License. For details, see the LICENSE file in the top-level source
# directory or online at http://www.openafs.org/d1/license10.html

# Configuration information for AFS client
```

```

# AFS_CLIENT and AFS_SERVER determine if we should start the client and or
# the bossserver. Possible values are on and off.
AFS_CLIENT=on
AFS_SERVER=off

# AFS client configuration options:
LARGE="-stat 2800 -dcache 2400 -daemons 5 -volumes 128"
MEDIUM="-stat 2000 -dcache 800 -daemons 3 -volumes 70 -memcache"
SMALL="-stat 300 -dcache 100 -daemons 2 -volumes 50"
OPTIONS=$MEDIUM

# Set to "-verbose" for a lot of debugging information from afsd. Only
# useful for debugging as it prints _a lot_ of information.
VERBOSE=

# OPTIONS are the options passed to afsd.
OPTIONS="$OPTIONS $VERBOSE"

# Sample server preferences function. Set server preferences using this.
# afs_serverprefs() {
#   /usr/afsws/etc/fs setserverprefs <host> <rank>
#}

# Either the name of an executable script or a set of commands go here.
# AFS_POST_INIT=afs_serverprefs
AFS_POST_INIT=

```

Starting the Cache Manager

1. Reboot the VM Linux server, log on as root, and then start the AFS initialization script and wait for the message that all daemons are started:

```

# cd /
# shutdown -h now
...
login: root
password: root_password
...
# /etc/init.d/afs start
Starting AFS services.....
afsd: All AFS daemons started.

```

2. Follow the instructions in “Activating the AFS initialization script” on page 378 to enable the script to load automatically.

Setting up volumes and loading binaries into AFS

Here we create /usr/afsws on the local disk to the directory in AFS that houses AFS binaries for this system type. We prepared those binaries in the “Storing AFS binaries in AFS” on page 381.

1. Create /usr/afsws on the local disk as a symbolic link to the directory /afs/itso.ibm.com/@sys/usr/afsws with the command:

```
# ln -s /afs/itso.ibm.com/@sys/usr/afsws /usr/afsws
```

Congratulations! Now you are ready to use the AFS file system on your client. Next time you log on to your system, you will already be authenticated to the AFS server.

Important: On any Linux system on which you want to use the AFS logon with authentication to the AFS server, you still have to define a /etc/passwd entry with the same username and user ID as defined on the AFS server. This is required by the AFS PAM module. It authenticates you to the AFS server, but it still requires a local entry in the /etc/passwd file.

15.4.7 Installing Windows 2000 OpenAFS Client

In this section we explain how to install and configure Windows 2000 OpenAFS Client. You can obtain the compiled version of AFS client from:

<http://www.openafs.org/release/latest.html>

In our example we used version 1.0.4.a.

After installing the package, start the AFS client. You will see a window similar to Figure 15-2.

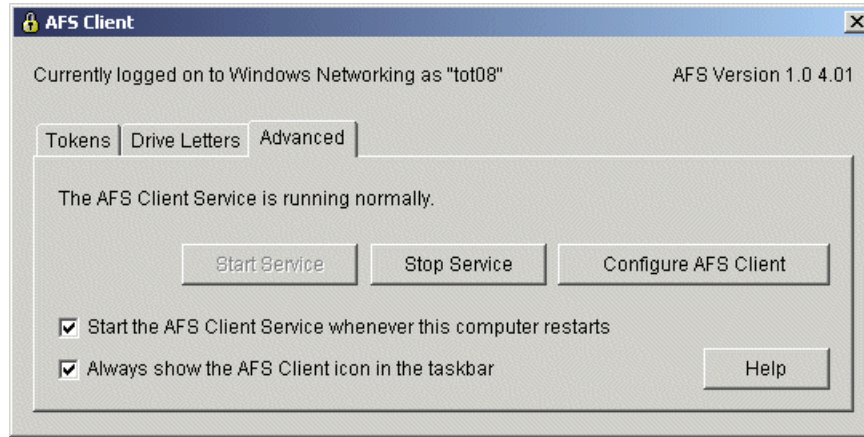


Figure 15-2 Windows 2000 AFS Client

On the **Advanced** tab, click **Configure AFS Client** and you will see a window similar to Figure 15-3 on page 388.

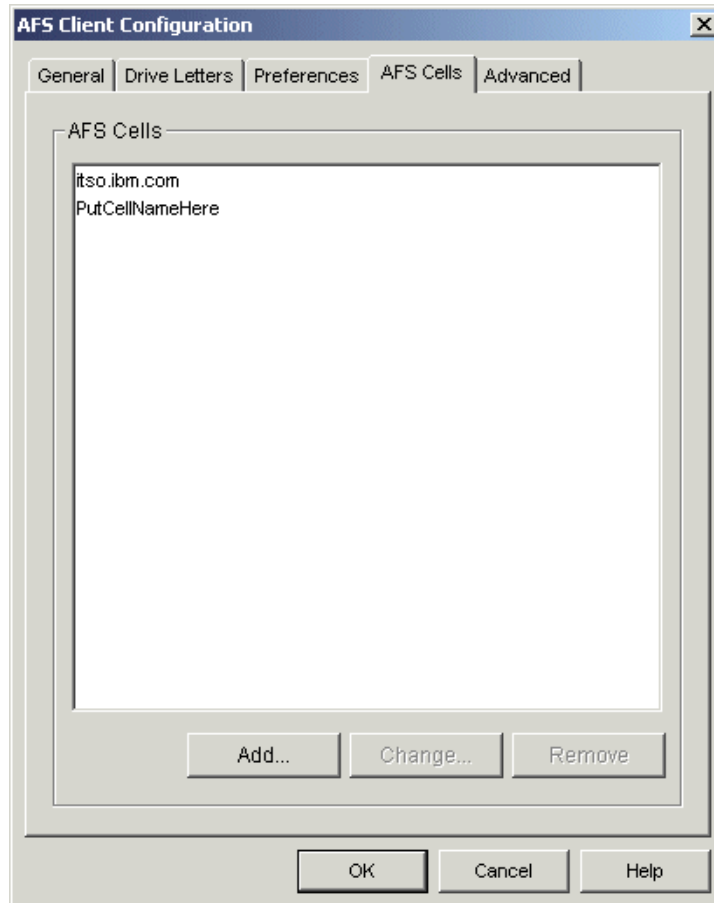


Figure 15-3 Configuring Windows 2000 AFS Client

Select the **AFS Cells** tab and click **Add...** , and you will see a window similar to Figure 15-4 on page 389.

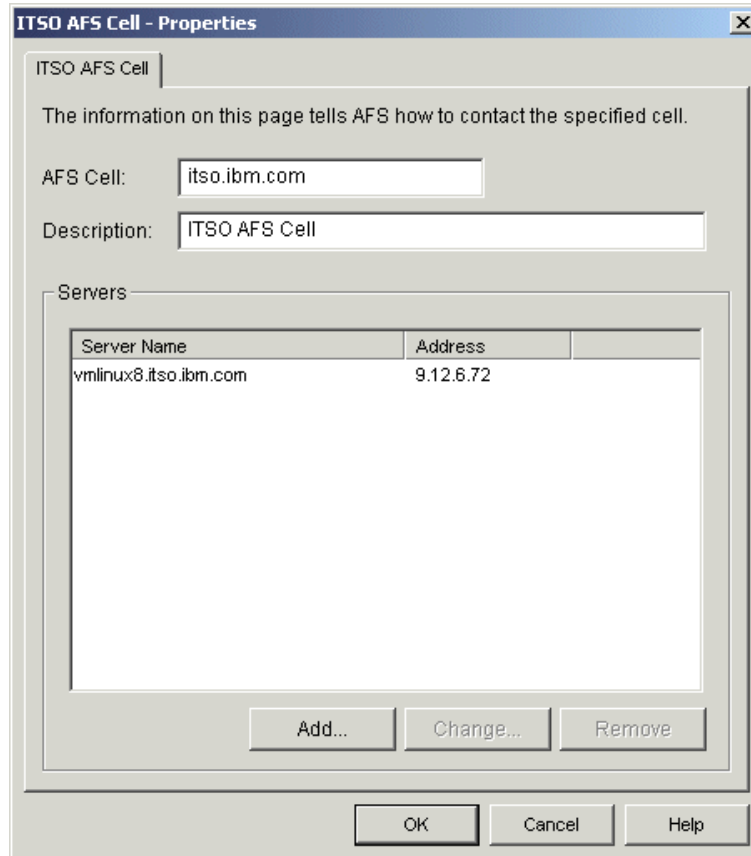


Figure 15-4 Defining the AFS Cell

Define your AFS Cell name here, and the server that is holding this cell. In our example you can see that the cell name is `itso.ibm.com` and that the server is `vmlinux8.itso.ibm.com`. Now start the AFS client service.

After defining the AFS Cell and starting the service, you can log on to get the tokens. In the main AFS client setup shown in Figure 15-2 on page 387, select the **Tokens** tab and click **Obtain New Tokens....** You will see a window similar to Figure 15-5 on page 390.



Figure 15-5 Logging on to the AFS server

Now you can map the AFS directory /afs to the drive letter on your Windows 2000 workstation. In the main AFS client setup shown in Figure 15-2 on page 387, select the **Drive Letters**, and you will see a window similar to Figure 15-6.

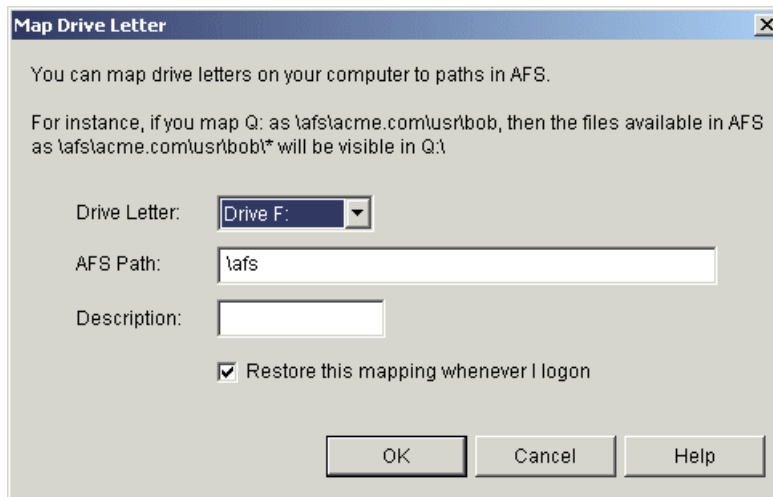


Figure 15-6 Mapping the /afs directory to the local drive letter

After you mapped the /afs directory to the drive letter, you can see the /afs content by exploring the drive you assigned. You will see a window similar to Figure 15-7 on page 391.

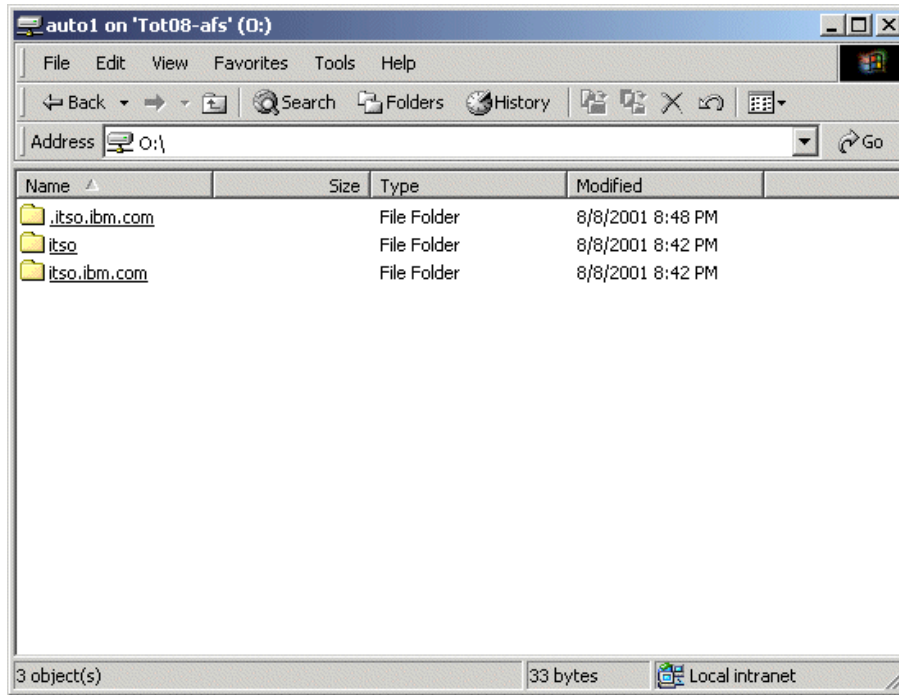


Figure 15-7 /afs directory mapped to the drive

Tip: If you define the same username and password on the AFS server as you use on your Windows 2000 workstation you can integrate the logon to AFS server with your workstation logon. With this you can use single logon to the workstation and AFS server.



z/VM 4.2 Linux features

In this chapter, we describe the enhancements shipped with IBM's z/VM 4.2 Operating System that can benefit customers running Linux guest virtual machines.

We review the System Administration Facility, which can help in the creation and management of multiple Linux guests. This facility can also be used to migrate from an existing Virtual Image Facility (VIF) environment.

We also review the VM LAN Support introduced in z/VM 4.2. This facility allows z/VM to support multiple internal, virtualized LAN segments. Individual guest machines can define a virtual network interface card (NIC) and then connect to the virtual or guest LAN segment using existing communications software, such as the Linux TCP/IP stack (we refer to these as “guest LANs” to avoid confusion with virtual LAN, since that term has been adopted by an IEEE standard, 802.1q).

16.1 System Administration Facility

The System Administration Facility is in part based on a number of ease-of-use functions developed for the S/390 Virtual Image Facility (VIF) product. These functions include the creation of Linux guest machines, the assignment of disk space for those guests, and the ability to start and stop Linux servers. The facility is comprised of a client component which runs in either a Linux or CMS guest. This client component communicates with a server component known as the VMADMIN server.

16.1.1 Who should use the System Administration Facility

The System Administration Facility is intended for use in a *new, non-tailored* z/VM system. It will not function if you want to use a pre-existing system. VMADMIN manages the user directory (which means that you can't do it—either manually, or through a directory management product such as DIRMAINT).

Before deciding on whether or not this facility is going to be useful in your environment, we recommend that you review the following flowchart, which comes from the *System Administration Facility Guide*, SC24-6034.

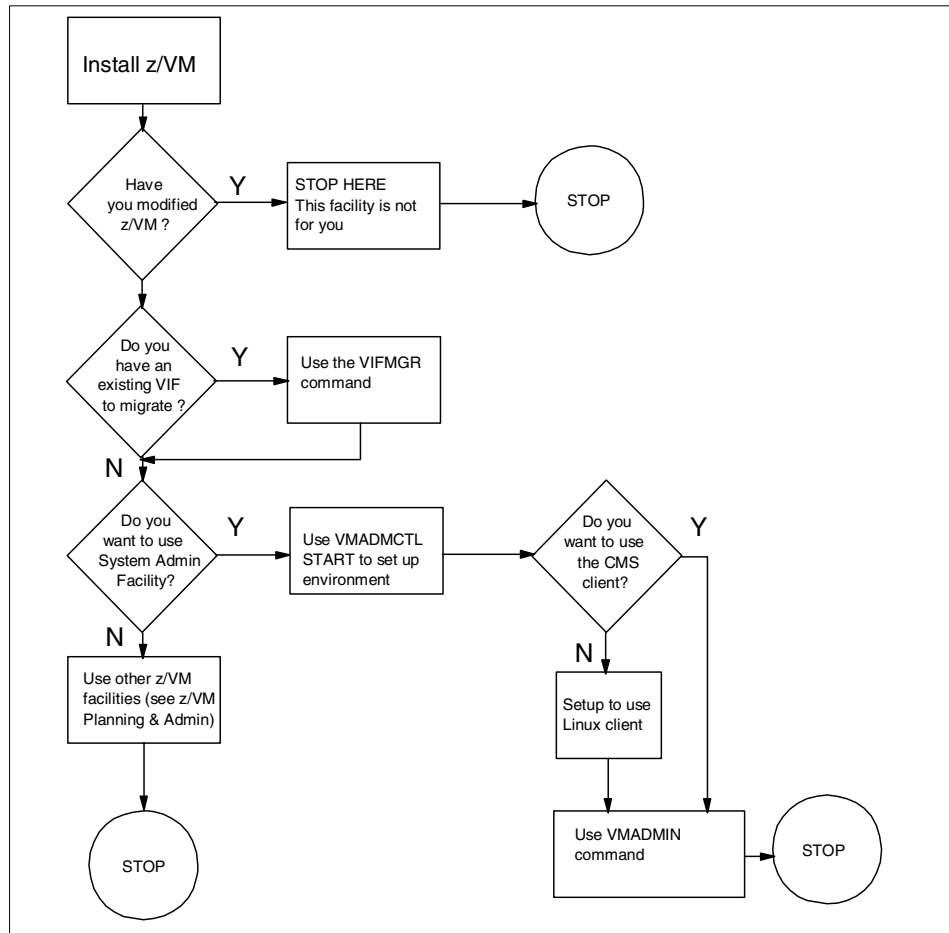


Figure 16-1 Figure 5-1 How to proceed after installing z/VM

16.1.2 Initializing the System Administration Facility

If you decide to use the System Administration Facility in your environment, execute the following steps to initialize the facility (before proceeding, however, we recommend you first read the instructions given in *System Administration Facility Guide*, SC24-6034).

1. Logon to CMS on the newly installed z/VM system, using the VMADMIN user ID.
2. Enter the command **VMADMCTL START**. (Note that this command can only be entered once; you cannot stop and then restart VMADMCTL.)
3. Reply to the prompts with your environment-specific responses.

You'll need the following information about your environment in order to answer the VMADMCTL initialization questions:

- Network device address for the VMADMIN server
Network devices such as OSA cards have two (and sometimes, three) device addresses assigned to them. There is always an even number and an odd device number, which represent read and write subchannels.
On an OSA Express card running in QDIO mode, there's a third device for the control subchannel. To answer this prompt, specify the even device address number of the network card you'll be using for VMADMIN. For example, if using an OSA card with device addresses 0x0500, 0x0501, and 0x0502, you'd reply with 0500.
- The network device's port number or port name
This is the number or name of the starting even port, or the name associated with the network device that is assigned to the VMADMIN server.
- VMADMIN server network type
- The type of local area network (LAN) to which the server is connected. For a QDIO device, specify either FastEthernet, FE, GigabitEthernet, or GB. For a non-QDIO device, specify either Ethernet, 802.3, TokenRing, TR, or FDDI.
- VMADMIN server network MTU size (576, 1492, 1500, 2000, 4096, 4352 or 8902)
- VMADMIN server IP address
- VMADMIN server subnet mask
- IP address of gateway to be used by the System Administration Facility

Once configuration has completed, the VMADMIN server will be initialized and a message will be generated, informing you that VMADMIN is now operational.

Following is an example of the initialization process that includes the prompts and example responses:

```
VMADMCTL START
HLEVMA0050I Reply RESTART at any time to start over or QUIT to terminate
HLEVMA0052R Enter Server network device address:
9.12.6.73
HLEVMA0036E Device address must be hexadecimal
HLEVMA0052R Enter Server network device address:
292C
HLEVMA0053R Enter Server network port number:
0
```



```

HLEVMA0054R Enter Server network type (Ethernet, 802.3, TokenRing, TR,
FDDI):
Ethernet
HLEVMA0055R Enter Server network MTU size (576, 1492, 1500, 2000, 4096,
4352, 8902):
1500
HLEVMA0056R Enter Server IP address:
9.12.6.73
HLEVMA0057R Enter Server IP mask:
255.255.255.0
HLEVMA0058R Enter Server gateway IP address:
9.12.6.75
HLEVMA0079I
HLEVMA0079I Here is the configuration (please make a note of it):
HLEVMA0079I
HLEVMA0079I Server:
HLEVMA0079I         Network device address: 292C
HLEVMA0079I         Network port: 0
HLEVMA0079I         Network type: ETHERNET
HLEVMA0079I         Network MTU size: 1500
HLEVMA0079I         IP address: 9.12.6.73
HLEVMA0079I         IP mask: 255.255.255.0
HLEVMA0079I         Gateway IP address: 9.12.6.75
HLEVMA0079I         Client IP address:
HLEVMA0079I
HLEVMA0080R Is this correct (Yes(1),No(0)):
1
14:51:28 AUTO LOGON   ***          VMADMIN  USERS = 8          BY MAINT
HLEVMA0498I VMADMCTL complete - VMADMIN is now operational

```

Verify that all the settings are correct by using the **VMADMIN Q ALL** command:

```

VMADMIN Q ALL
HLE$QU0079I
HLE$QU0079I Here is the configuration (please make a note of it):
HLE$QU0079I
HLE$QU0079I Server:
HLE$QU0079I         Network device address: 292C
HLE$QU0079I         Network port: 0
HLE$QU0079I         Network type: ETHERNET
HLE$QU0079I         Network MTU size: 1500
HLE$QU0079I         IP address: 9.12.6.73
HLE$QU0079I         IP mask: 255.255.255.0
HLE$QU0079I         Gateway IP address: 9.12.6.75
HLE$QU0079I         Client IP address: NOT DEFINED
HLE$QU0079I
HLE$QU1300I 0 of 0 MB of server paging space in use
HLE$QU1301I 0 MB of 0 MB of Linux image partition space in use

```

```

HLE$QU1302I VMADMIN performance: CPU is Green, Paging is Green, I/O is
Green
HLE$QU1324I No paging volumes are defined
HLE$QU1324I No image volumes are defined
HLE$QU1303I Server uses IP address 9.12.6.73 with device 292C
HLE$QU1306I Server level: 18, Service 000
HLE$QU1307I Last boot on 2001-08-02 at 15:15:06

```

16.1.3 Using VMADMIN

VMADMIN functions can be run from a CMS session or from a Linux guest machine. Before you can run VMADMIN from Linux, however, you must first create a Linux image, using the VMADMIN CMS client. Once you have created the first Linux system, all other VMADMIN work can be done from Linux.

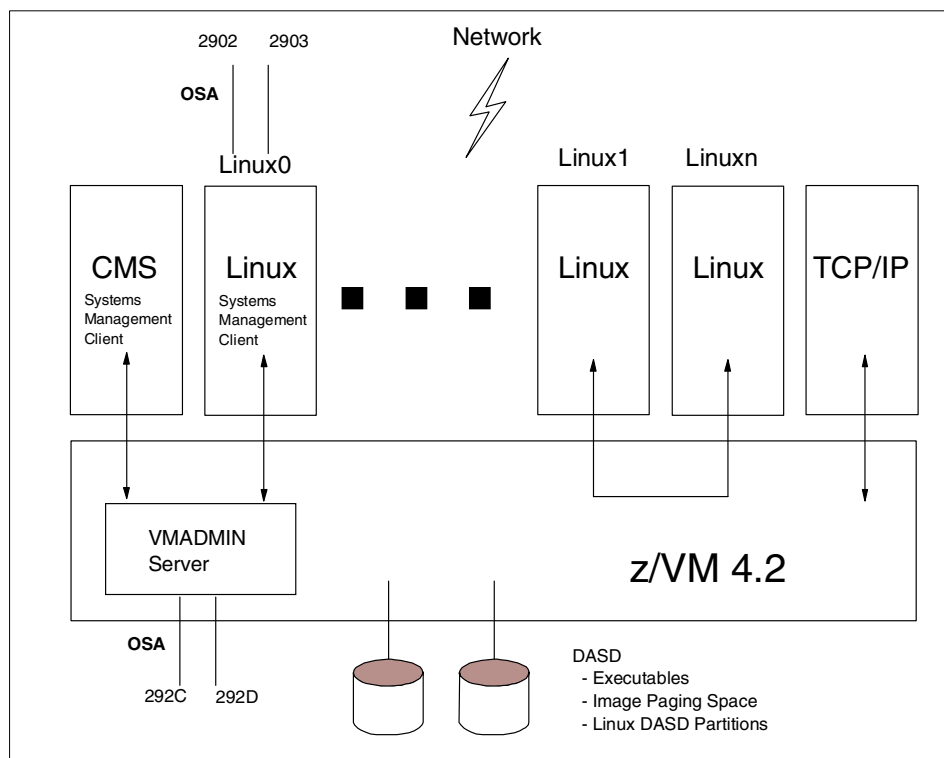


Figure 16-2 Example System Administration Facility environment

16.1.4 Creating the initial Linux guest

The following steps take you through the creation of the initial Linux virtual machine. These steps must be performed from a CMS session that has access to the VMADMIN facility. (Typically, in a new installation you should use the MAINT user ID for this task.)

1. Using the VMADMIN SERVER VOLUME command, create the paging and image (Linux filesystem) space that will be used by the Linux guest.

Note: The following examples show user input in bold.

Create a paging device, using device address 3E44, and give it a volume label of VM3E44.

```
vmadmin server volume add paging 3e44 vm3e44
16:19:38 DASD 3E44 ATTACHED TO VMADMIN 1000 BY VMADMIN WITH DEVCTL
HLE$V02004I Command may take up to 16 minutes; please wait
```

Once the format has completed, map the volume to ensure that everything worked.

```
vmadmin server volume map used vm3e44
HLE$V02216E VM3E44 (3E44) is a PAGING volume
Command Complete
```

Add a volume for "image" space (i.e., space that will be used by the Linux filesystems). Once this completes, also map this volume to ensure that the format completed successfully.

```
vmadmin server volume add image 3ca3 vm3ca3
HLE$V02200I IMAGE volume VM3CA3 added
Command Complete
```

2. Create the first Linux guest by using the command **VMADMIN IMAGE CREATE**. (In our case we named it LNXMSTR since it is the first Linux guest, but you can name it whatever you prefer.)

Note: This command merely defines the guest to VM and does not perform the actual Linux install.

```
vmadmin image create LNXMSTR
Enter password for Image:
PASSWORD
Re-enter password:
PASSWORD
HLE$IM1500I Image LNXMSTR created successfully
Command Complete
```

3. Use the **VMADMIN SERVER INSTALL** command to copy the kernel image, parmline, and initial ramdisk files from a nominated FTP server to the guest machine. Once this completes, you'll be ready to boot the Linux system starter system.

Select the FTP server that contains the kernel image, parmline, and initial RAMdisk that you want to install. In our case we used SuSE, so the location of these files is stored in the suse.ins file.

Note: We had a problem in getting this step to work, at first. We resolved the problem by editing the suse.ins file, entering fully qualified path names (i.e. paths from the root directory) for the image, parmline, and initrd files. This problem may now be resolved, however, as we were using an early build of z/VM 4.2.

```
VMADMIN SERVER INSTALL 9.12.6.134 ftpuser ftppwd suse.ins
HLE$IN2305I Transferring Linux from 9.12.6.134 suse.ins
HLE$IN2003I Command may take up to 20 seconds; please wait
HLE$IN2304I Linux installed from 9.12.6.134 suse.ins
Command Complete
```

4. Define the network device that'll be used by the first Linux guest, by using the **VMADMIN IMAGE NETWORK** command. In our example, the Linux guest will use an OSA card with device addresses 2902 and 2903 (see Figure 16-2 on page 398 for details).

```
VMADMIN IMAGE NETWORK lnxmstr add 2902
HLE$IM1506I NETWORK ADD completed successfully
Command Complete
```

5. Authorize the Linux master guest to be able to be able to run VMADMIN commands.

```
vmadmin server clientipaddress 9.12.6.65
HLE$CL2504I Ping to Client IP address 9.12.6.65 failed
HLE$CL2501I Client IP address is set to 9.12.6.65 successfully
Command Complete
```

6. Provide read-only access to VMADMIN's 203 minidisk, which holds the Linux vmadmin command.

```
vmadmin partition share vmadmin 203 with lnxmstr 203
HLE$PA1506I PARTITION SHARE completed successfully
Command Complete
```

You're now ready to boot the initial Linux guest machine.

7. Logon to a CMS session using (in our example) user ID LNXMSTR. The Linux system will automatically boot.

```
LOGON LNXMSTR PASSWORD
There is no logmsg data
FILES: NO RDR, NO PRT, NO PUN
LOGON AT 21:55:18 EDT THURSDAY 08/16/01
Linux version 2.2.19 (root@s39016) (gcc version 2.95.2
(SuSE+gcc-2.95.2.4-diffs+gcc-bugfixes)) #1 SMP Mon Jun 18 05:19:40 2001
Commandline is: ramdisk_size=32768root=/dev/ram0ro
We are running under VM
```

```
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 001F on subchannel 0000 - PIM = 80, PAM = 80, POM = FF
Detected device 2902 on subchannel 0001 - PIM = 80, PAM = 80, POM = FF
Detected device 2903 on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
Detected device 0203 on subchannel 0003 - PIM = F0, PAM = F0, POM = FF
...
```

8. Because this is the first time this Linux image has been booted, you'll be prompted for networking definitions. Configure the network, and then telnet to the image. You can now install the full system by using an appropriate installation script (we used YaST in our example).

To issue VMADMIN commands from the Linux guest, once you log onto the image, you must make the DASD at device number 203 known to Linux and it must be mounted read-only. (For more information about the procedure you'll need to follow to accomplish this step, refer to the documentation for the Linux distribution you are using.)

Note: For a complete description of the functions available with the System Administration Facility, refer to the z/VM 4.2 publication *System Administration Facility Guide*, SC24-6034.

16.2 VM LAN support

Prior to z/VM 4.2, virtual connectivity options for connecting one or more virtual machines were restricted to virtual channel-to-channel (CTC) links, and the Inter User Communications Vehicle (IUCV) facility. These are point-to-point connections, which means that in the case of CTC links, when you want two virtual machines to communicate with each other, you must define CTC device pairs in *each* machine and couple those devices between guest machines. You also have to define static routing statements in each guest that needs to communicate with another guest in the system.

Another problem with point-to-point links is that, if one side of the connection went down, it was often difficult to subsequently reconnect the two machines. Frequently, one of the Linux guest machines would have to reboot in order to pick up the connection.

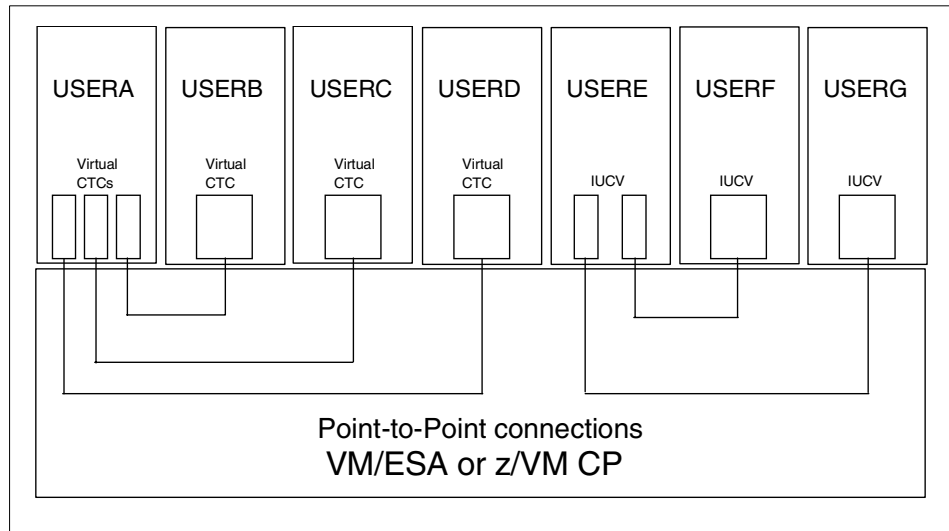


Figure 16-3 Guest virtual connectivity options prior to z/VM 4.2”

From z/VM 4.2, CP has been enhanced to provide a feature known as “VM LAN”; see Figure 16-4 on page 403. This feature allows you to create multiple virtual LAN segments within a z/VM environment, and there is no limit on the number of LAN segments that you can create. Individual guest machines can create a virtual Network Interface Card (NIC) to allow them to connect to the virtual LAN and communicate with other guests using standard TCP/IP protocols.

The virtual NIC emulates a HiperSockets device, as introduced by the zSeries z900 GA-2 machines in late 2001. As the VM LAN is a virtualization technique, it is not limited to the use of zSeries hardware; support for VM LAN goes back to 9672 Generation 5 machines and the Multiprise 3000.

Unlike with the complexity of point-to-point connections, when using the VM LAN facility, you only have to define the virtual network adapter in each guest and connect that adapter to the LAN.

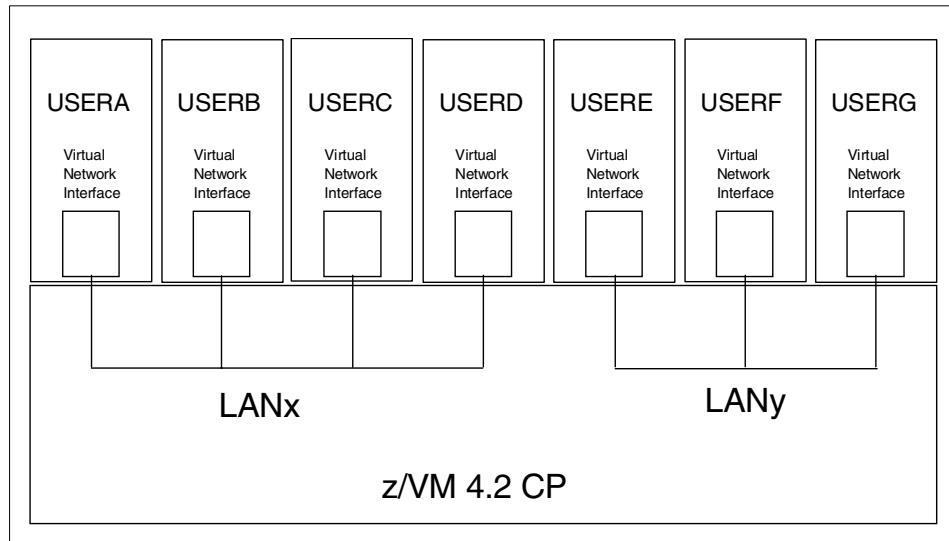


Figure 16-4 VM LAN support in z/VM 4.2

To use the VM LAN, the guest O/S (for example, Linux) must be able to support HiperSockets. In our case, we used an IBM internal test Linux system which had Hipersockets support built into the QETH driver.

VM LANs can be created or destroyed dynamically. It is possible to have both unrestricted and restricted LANs. Access to a restricted LAN is controlled via an access control list.

16.2.1 Creating a VM LAN

To manually create a virtual LAN for a group of VM guests, follow these steps:

1. Create a VM LAN segment in the VM host system.

```
CP define lan SEWLAN MAXCONN 100 ownerid system
23:00:55 LAN SYSTEM SEWLAN is created
```

We created a LAN called SEWLAN, and the maximum number of guests that can connect to this LAN is 100. The MAXCONN setting can be changed to any number between 1 and 1024. Alternatively, if you do not use a MAXCONN value, then there is no limit on the number of guests that can connect to this LAN.

The LAN definition given in our example is not permanent across IPLs of VM, so you should add a DEFINE LAN statement to VM's SYSTEM CONFIG file. Refer to the DEFINE LAN section in the *z/VM CP Command and Utility Reference* for additional details on this subject.

2. On each individual guest machine, you must create a virtual network interface card (NIC).

```
CP define nic 500 hiper devices 3
```

```
23:08:01 NIC 0500 is created; devices 0500-0502 defined
```

This creates a set of devices that will look like a HiperSockets interface to Linux. Again, this is not a permanent definition; it will only exist for the life of the guest session. To make this definition permanent, add a SPECIAL statement in the CP directory for that guest, either by editing the USER DIRECT file or by running DIRMAINT.

3. On each guest, connect the virtual NIC to the LAN.

```
CP couple 500 to system sewlan
```

```
23:10:12 NIC 0500 is connected to LAN SYSTEM SEWLAN
```

To ensure this happens whenever a Linux guest starts up, put this **COUPLE** command into each guest's PROFILE EXEC file.

16.2.2 Using the VM LAN with Linux guests

Note: The information in this section is based on tests using a pre-GA QDIO driver on an internal IBM system; your experiences may be different. Both the VM LAN facility and the supporting Linux device drivers will be available by late 2001 as GA code.

To use the VM LAN with Linux guests, we followed these steps:

1. We defined the VM LAN, created a virtual NIC for each our Linux guests, and then connected those NICs to the LAN as discussed in 16.2.1, "Creating a VM LAN" on page 403.
2. We booted Linux, using an initial RAMdisk installation.
3. When we got to the networking prompts, we entered the following details:

```
Welcome to Linux for S/390
Is your machine connected to a network (Yes/No) ? yes
```

```
Select the type of your network device
1) for lcs osa token ring
2) for lcs osa ethernet
3) for qdio osa ethernet
4) for channel to channel and escon channel connections
5) for IUCV
6) for CLAW
Enter your choice (1-6): 3
```

```
Please type in the channel device configuration options, e.g
qeth0,0xfd00,0xfd01,0xfd02,0,1
```


qeth parameter:
qeth0,0x0500,0x0501,0x0502,0,0,0

Please enter your IP address: **192.168.0.10**
Please enter the net mask: **255.255.255.0**
Please enter the net address: [192.168.0.0] **192.168.0.0**
Please enter the gateway address: [192.168.0.1] **192.168.0.1**
Please enter the IP address of the DNS server:
Please enter your host name: **zvm1nx3.itso.ibm.com**
Please enter the DNS search domain: **itso.ibm.com**

Configuration will be:
Channel device : qeth0,0x0500,0x0501,0x0502,0,0,0
Host name : zvm1nx3.itso.ibm.com
IP address : 192.168.0.10
Net mask : 255.255.255.0
Broadcast address: 192.168.0.255
Gateway address : 192.168.0.1
Net address : 192.168.0.0
DNS IP address :
DNS search domain: itso.ibm.com
Is this correct (Yes/No) ?
Yes

4. When we got to the Linux shell, we entered the command **ifconfig -a** in order to determine if we had a hipersockets device defined.

```
# ifconfig -a
hsi0      Link encap:Ethernet HWaddr 00:00:00:00:00:00
          NOARP MTU:8192 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:14

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

We also entered a **CP QUERY NIC** command to determine the status of our Linux guest machine's virtual Network Interface Card. As seen from the following example, we had an established session, but did not have an IP address bound to the NIC.

```
CP Q NIC 500 DETAILS
Adapter 0500 Type: HIPER      Name: UNASSIGNED Devices: 3
Port 0 MAC: 00-04-AC-00-00-05 LAN: SYSTEM ITSOLAN      MFS: 16384
```

```

Connection Name: HALLOLE   State: Session Established
Device: 0500 Unit: 000   Role: CTL-READ
Device: 0501 Unit: 001   Role: CTL-WRITE
Device: 0502 Unit: 002   Role: DATA

```

5. We displayed the contents of the /proc/chandev file in order to verify that the devices 0x500,0x501,and 0x0502 have been detected.

```
# cat /proc/chandev
```

```

channels detected
      chan  cu    cu    dev    dev
      devno type  type  model  type  model pim  chpids
=====
0x0000 0x2946 0x04  0x3088 0x60  0x0000 0x00 0x80 0x190000000000000000 no  no
0x0001 0x2947 0x04  0x3088 0x60  0x0000 0x00 0x80 0x190000000000000000 no  no
0x000e 0x0500 0x10  0x1731 0x05  0x1732 0x05 0x80 0x050000000000000000 yes yes
0x000f 0x0501 0x10  0x1731 0x05  0x1732 0x05 0x80 0x050000000000000000 yes yes
0x0010 0x0502 0x10  0x1731 0x05  0x1732 0x05 0x80 0x050000000000000000 yes yes

```

6. We were now ready to activate the hipersockets device via an **ifconfig** command, as follows:

```
# ifconfig hsi0 192.168.0.1 netmask 255.255.255.0 multicast up
```

```

hsi0  Link encap:Ethernet HWaddr 00:00:00:00:00:00
      inet addr:192.168.0.10 Mask:255.255.255.0
      UP RUNNING NOARP MULTICAST MTU:8192 Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      Interrupt:14

```

7. Finally, we performed another **QUERY NIC** command and noted that this time, we had an IP address bound to the virtual NIC.

```

#CP Q NIC 500 DETAILS
Adapter 0500 Type: HIPER   Name: UNASSIGNED Devices: 3
Port 0 MAC: 00-04-AC-00-00-05 LAN: SYSTEM ITSOLAN   MFS: 16384
Connection Name: HALLOLE   State: Session Established
Device: 0500 Unit: 000   Role: CTL-READ
Device: 0501 Unit: 001   Role: CTL-WRITE
Device: 0502 Unit: 002   Role: DATA
Unicast IP Addresses: 192.168.0.10

```

This Linux guest can now communicate with any other member of the virtual LAN without requiring static routes to individual machines or specific COUPLE statements to link it to other guests.



Roadmap

In this chapter we discuss some areas we would have liked to have investigated further. Some research was done to test “good ideas.” In some cases experiments were done and prototypes were coded to verify the ideas. We believe these areas could allow significant improvements to running Linux images on VM, but time did not allow us to complete the work.

17.1 Ability to reconfigure CTC and IUCV

In “Linux as a virtual router” on page 68 we drew attention to the lack of ability to reconfigure the CTC and IUCV drivers.

The way that these drivers work is that connections are defined when the device driver is loaded. The restrictions are different for each driver:

- CTC** The virtual CTCs must be defined at Linux IPL time because the current Linux kernel does not properly handle dynamically defined CTC devices. The devices do not need to be coupled to the peer user ID until the moment you want to activate the connection, so you can postpone that decision and use the **hcp** command to do the couple. Thus, the restriction is only in the number of devices you want to use.
- IUCV** The IUCV driver does not use S/390 devices, so it is not subject to restrictions in the device layer of the kernel. The current driver, however, requires all peer user IDs to be specified when the driver is loaded. You can unload the driver and specify additional peer user IDs, but that means you must bring down the other IUCV connections in this image.

We believe it would not be very difficult to change the IUCV driver such that you specify only the number of connections when it is loaded, and specify the peer user ID just before the **ifconfig** command by writing into a `/proc` entry. This would give the IUCV driver at least the flexibility of the CTC driver.

There are also some scaling issues with the CTC driver that prevent this from being effective. When a number of connections are down and the virtual router is trying to establish the connection, this appears to keep the CTC driver from sending packets on the other connections that are up.

Currently a maximum of 8 or 10 connections is defined for the CTC and IUCV drivers. People have suggested that this could be changed (based on the fact that it is defined as a constant), but that might be less trivial than it looks. The CTC driver allocates 2 times a 64 KB buffer for each connection, so 100 connections would require 12.5 MB of storage. This would make it hard to create a small compact Linux router that can be kept resident by z/VM. The IUCV driver requires the user IDs to be listed when the driver is loaded. That causes some practical problems as well if you want to specify hundreds of user IDs.

It would be very attractive if z/VM would provide a kind of virtual network based on IUCV or some other interface to CP (as opposed to virtual point-to-point connections). Such a virtual network should offer broadcast capability as well so that a DHCP server (in a Linux image on VM, connected to that same virtual LAN) would provide the information to let a DHCP client in each Linux image

configure the IP stack automatically (as suggested in 10.4.4, “Reducing the number of changes required” on page 220). Given the penetration of Ethernet in the Linux arena, it would be attractive if the Linux device driver would make it appear as a network interface on a virtual Ethernet LAN.

17.2 SOURCEVIPA equivalence for Linux

As discussed in 4.2.3, “Virtual IP addressing” on page 79, we can use a dummy interface in our Linux instance to provide a persistent traffic path for TCP/IP connections. This is used to provide a single addressing point for applications, even when multiple interfaces are used. It also allows for private IP addressing ranges to be used in the virtual routing setup, conserving the Internet address range.

While the dummy interface provides resilient connectivity support for incoming connections, it does not assist when the Linux instance establishes an outbound connection. This is because of the way the TCP/IP **connect ()** function works. Part of **connect ()** processing uses the TCP/IP routing table to determine which interface the connection request (TCP SYN packet) will be sent over. The IP address of this interface is used as the source address for the connection.

In z/OS and z/VM, this default behavior has been modified, adding extra processing to support SOURCEVIPA. z/OS and z/VM check the configuration for the interface chosen in **connect ()**, and if SOURCEVIPA has been specified for that interface, the address of the appropriate VIPA will be used as the source address.

Note: This assessment of the SOURCEVIPA function in z/OS and z/VM has been done simply by looking at how the function works, not by actual inspection of the code. None of the authors of this redbook have access to z/OS or z/VM code, so the way the function is implemented may be different from this.

It would be possible to make the same changes to the Linux INET code to support the same feature. There are many considerations, however, that would have to be considered:

- ▶ The change would have to be duplicated in TCP and UDP (and may operate differently for both).
- ▶ IPV4 and IPV6 would need to be looked at.
- ▶ Other parts of the kernel might be affected. For example, the code that sends the SYN packet may be dependent on the interface address determined in **connect ()**, and would have to be changed as well.

- ▶ Applications that initiate connections might not perform as expected.

A fairly large amount of work would be involved in making this kind of modification to the INET code. However, at the end of the task Linux would be able to benefit from the same high-availability connectivity as z/OS and z/VM for incoming and outgoing connections.

Even without a function like SOURCEVIPA in your penguin colony, though, you obtain high availability for incoming connections using the dummy interface `ad` previously described.

17.3 DCSS-mapped block devices

Reduction of the storage requirements (“footprint”) of the Linux images is one of the prerequisites for effectively running a large number of Linux images on a VM system. One of the options for reducing the footprint is to share storage among the images. Sharing the read-only portions of the kernel as suggested in 10.7.1, “Using an NSS with just the kernel” on page 228 will help somewhat, but the kernel is only a small portion of the Linux image. It would be much more attractive to share the application code (the shared libraries and binaries).

17.3.1 Sharing between processes

When a process needs access to an executable or shared library, an `mmap()` function call is used to map this file into the virtual storage of the process. The file is not completely read into memory before the process starts, but portions are brought into storage when needed (like demand paging). When the process accesses a part of the file that is not in Linux storage, a page fault exception passes control to the kernel to start the I/O to read in a portion of that file and resume the process when the I/O has completed. In the case of shared libraries, another process might need the same portion of the mapped file and it would find the portion already in storage and not encounter a page fault. This means that portion of the file will be loaded into Linux storage only once. Popular portions of files will continue to reside in storage and will happen to be available for processes when needed.

This is something Linux does on each platform. It is not unique to S/390 and it does not exploit S/390 facilities other than the Dynamic Address Translation hardware (referred to as Memory Management Unit (MMU) on other platforms).

17.3.2 Sharing between images

This sharing process, described in 17.3.1, “Sharing between processes” on page 410, happens within a single Linux image. When Linux images are sharing disks with code, many of the Linux images will have read in the same portions of the same files. From a VM point of view, this means that duplicates of the same data is in the virtual storage of each of these Linux images. If the pages are referenced frequently enough by the Linux images, they will be part of the resident storage of the virtual machine. It would be very attractive if VM could play similar tricks as Linux does and map these virtual machine pages on the same real page frames. Since Linux is using normal I/O operations to read the data, it is not trivial for VM to recognize the pattern and perform a form of mapping.

Some benefit may be gained from VM Minidisk Cache (MDC) that will cache portions of the data in real storage page frames.

Note: One could argue whether MDC is effective in this case. Because Linux images already buffer these popular pages themselves, MDC will not notice that the page is more popular than other pages read once by a Linux image. The Linux I/O tends to be rather “MDC unfriendly” (except when using the diagnose interface), so this may be a moot point. Rather than argue, we probably should measure.

When the Linux image issues the I/O to get the data because it page faults, it is probably too late for VM to intercept and try to do smart things. This is further complicated by the fact that each Linux image will read that block of disk into a different page of the virtual machine storage.

One option would be to use the z/VM facilities to do something similar to the Linux `mmap()` function. The z/VM facilities, however, build on the XC architecture, and Linux for S/390 currently cannot run in that mode.

17.3.3 Using shared segments

Another z/VM facility to share storage is a discontinuous saved segment (DCSS). A virtual machine can “attach” a DCSS, which means that a defined part of its address space is mapped on a DCSS. The pages of the DCSS reside in a special spool file and are brought in when necessary through the VM paging subsystem, and will remain in storage when referenced frequently enough. Multiple virtual machines share the real page frames, much like the `mmap()`

approach in Linux. The DCSS is normally located outside the virtual storage of the virtual machine (i.e., at a virtual address beyond the size of the virtual machine). At boot time Linux will set up tables to map what it sees as real page frames, not segments attached to it after the boot process.

The way we can talk Linux into using a DCSS could be to have a block device driver. The block device driver will issue a Diagnose 64 call to attach the segment when the device is opened. The way to access the segment would be through the `ioremap()` function in Linux (this is used on PC platforms to access memory on PCI cards). The kernel currently does not export the symbol for device driver modules to use it, but this is trivial to do in `arch/s390/kernel/s390_ksyms.c`. We made that change and the code appears to work as expected, in that a device driver module can access the data in the segment.

The first attempt to implement this DCSS block device was to build a device driver on top of the device file system (devfs) so that segment names would show up in the `/dev` tree when attached. While this might be an elegant approach for a production driver, it turned out to be “a lot of work.”

The second attempt involved taking the XPRAM device driver that is part of the Linux for S/390 source tree, and change it to use a DCSS. The module was changed to take the names of the DCSS as a parameter when loading.

```
Aug 9 01:05:34 tux60000 kernel: dcssinfo:trying to load module
Aug 9 01:05:34 tux60000 kernel: dcssinfo:initializing:
Aug 9 01:05:34 tux60000 kernel: dcssdebug:dcss: this is 0 TUXTEST
Aug 9 01:05:34 tux60000 kernel: dcssdebug: major 34
Aug 9 01:05:34 tux60000 kernel: dcssinfo: hardsector size: 4096B
Aug 9 01:05:34 tux60000 kernel: dcssdebug:diag64 TUXTEST is 0 20000000 200ffff
Aug 9 01:05:34 tux60000 kernel: dcssinfo: 1024 kB expanded memory found.
Aug 9 01:05:34 tux60000 kernel: dcssdebug: device(0) offset = 0 kB, size = 1024 kB
Aug 9 01:05:34 tux60000 kernel: dcssinfo:Module loaded successfully
```

While writing the redbook the device driver was already doing the `ioremap()` call and it kept the pointer to the mapped memory for the segment. The `request()` function was changed to copy from and to mapped memory. In fact, we do not want to copy the page into Linux storage. We need to convince the system to use the page sitting outside virtual storage. Some extra stuff is needed to load the segment in non-shared mode to have it writable and to issue the `SAVESYS` when the segment is detached.

Another interesting application for the DCSS driver would be to attach a segment that has been prepared with the `mkswap` command. This would be like a swap disk in virtual disk (VDISK), but without the expensive channel programs to drive it.

17.4 Shadowed disk support

In the case where many hundreds to thousands of Linux images are essentially similar installations, with the exception of a few configuration files, it would be advantageous to have some means of minimizing the number of disk devices needed. To this end we began investigating the feasibility of a “shadowed” disk driver. This driver would use a common master drive as read-only data, but writes would be directed to a guest-specific shadow disk. Subsequent reads would retrieve changed blocks from the shadowed disk and unchanged blocks from the master disk.

One significant possible benefit to using a “shadowed” approach is that it becomes a much simpler matter to upgrade software for all the systems simultaneously. Several issues still remain; for example, configuration file format changes will still require some careful consideration and thought. In the main, though, it will be highly desirable to be able to “insta-patch” all the Web servers at once. It also greatly simplifies the management of hundreds of images by ensuring that they are all running the same version of software.

Another use of a disk shadow is to allow extremely simple recovery to a “known good” state. If one of the guest images manages to damage its configuration to the point that it cannot be easily repaired, or even that it will no longer boot, all that must be done is to delete the shadow and replace it with a blank, new shadow. In a matter of seconds, the system is up and running again with a known baseline configuration. With some of the other configuration automation techniques we have discussed, the newly “rebuilt” machine could even automatically make the first changes (e.g. IP address) such that when it comes up it is already alive and well on the network. Carrying this idea even further, for well-defined servers all carrying out a similar function (i.e., a cluster of Web servers) an automated process could automatically “resurrect” a failed server by bringing online a new, freshly configured image while retaining the old shadow copy for the system administrator to look at to determine the cause of the failure.

Combined with the DCSS driver in “DCSS-mapped block devices” on page 410, this would not only save disk space, but would also allow portions of the common master driver to reside in storage that is shared among Linux images.

One aspect of the driver that is somewhat more complex is maintaining shadow consistency with the master disk. If the master disk changes at all, then the block map on the shadow is invalid. Some tools could be developed that will:

- ▶ Generate a new shadow based on the old shadow and master
- ▶ Rebuild shadow/master to reclaim unused space (this will depend on how the file system behaves)

- ▶ Add new shadows to an existing shadow (multiple shadow disks to expand shadow capacity)
- ▶ Factor common changes out of multiple shadows to generate a new, more efficient master
- ▶ Simplify shadow management for the system administrator

A prototype implementation using the Linux md driver was developed during this residency, but was not completed in time for significant results to be documented in this book. Development is continuing, and we hope to be able to publish results at some future date.

17.5 File system access tool for systems management

Linux currently does not properly handle the dynamic LINK and DETACH of minidisks as CMS users are used to having.

There is a need to allow one Linux image to access the file system of another (not running) Linux image. When authorized to do so, users can link to each other's minidisks. Unfortunately, the current implementation of Linux for S/390 does not support dynamic linking to a minidisk and use of this disk in the DASD driver. Even when Linux would correctly handle the machine check interrupts involved with configuration changes of the virtual machine, the DASD driver would still need to be unloaded and reloaded with new parameters (which is not possible when the root file system is on disk).

The possibility to do this builds on VM facilities and is therefore not present on other Linux platforms.

17.5.1 Possible use for the utility

In general, systems management can be simplified when a running Linux image can dynamically access the file system on a minidisk other than the minidisks linked to the virtual machine at IPL. The restrictions of the kernel and DASD driver currently do not offer this option.

Ad-hoc file system repair activities

For CMS-based applications, a VM systems programmer or application programmer would want to link and access minidisks of service machines to investigate issues and fix problems. A similar facility would be useful for systems management of Linux images on VM.

Alternative data sharing

Instead of sharing data via permanent network connections, applications can exploit VM facilities to link to the file system of another Linux image. This should be done while the other Linux image is not running (in which case a network connection would not be possible, anyway).

Automated configuration changes of cloned images

When a new Linux image is created by copying the disks of an existing Linux image, several files must be customized for the new image (IP address, host name, root password). This process can be simplified if normal Linux utilities can be used to apply these changes.

DirMaint-controlled file system copy

For minidisks in CMS format with CMS files, DirMaint can automatically copy the files when the size of a minidisk needs to be increased. If a Linux system can dynamically link to the old and new minidisk, the same function could be implemented for minidisks containing a Linux file system. This is very important for storage management.

17.5.2 Design outline

A program on the user's Linux image (the master) will take the arguments needed to access the minidisk (user ID, virtual address, optionally read password, mount point). Because of the limitations of the DASD driver to dynamically add devices, a new Linux image (the worker) must be started when a minidisk needs to be accessed. This new image links the proper minidisk and IPL Linux from an NSS that also contains a RAMdisk image. The DASD driver can now be loaded and the file system on the minidisk can be mounted in the root file system. The new Linux image connects to the master Linux image via an IUCV connection and exports the mounted file system via NFS. The program that initiated this can now mount that exported file system in its own file system.

With an enhancement to pick up the IPL parameters, as shown in 10.7.3, "Picking up IPL parameters" on page 230, we do not even need to boot with initrd. The additional disks can be specified in the IPL command: an extra option could be passed to the boot scripts to indicate what action is required from the worker.

17.5.3 Detailed design

The IPL of the worker should be reasonably fast to make this work. We have seen that booting a kernel with an uncompressed 30 MB RAMdisk image from NSS can be done within 3 seconds when the pages needed are already in storage. The RAMdisk image can probably be made smaller, which would further speed up the process. If necessary, the system can be IPLed from disk if that turns out to be faster.

To get IUCV connections between worker and master, the proper IUCV statements must be in the CP directory. The netiucv driver requires all peer user IDs to be defined when the driver is loaded. These restrictions suggest that we create a few of these workers for each virtual machine that must use this facility. These workers should be dedicated to this master. This simplifies security issues, because the workers can have the same authorization as the master. The workers do not need disk space, so the cost of half a dozen workers for each system administrator is not much.

The program to initiate this function can find the first free IUCV connection and XAUTOLOG the corresponding worker. The XAUTOLOG command can be issued using the `hcp` command in the `cpint` package. An `ifconfig` command can be issued to define the IUCV connection. Root authorization in Linux is normally needed for the `hcp` command, but that applies to the final `mount` command as well.

17.5.4 Additional points for the implementation

Booting from RAMdisk may be too restrictive for a full implementation. The alternative is to keep a separate root device for each worker. Because of the restrictions of the DASD driver, the kernel must be IPLed from NSS and a small modification must be made to the bootstrap such that it takes parameters from the IPL command to insert them in the command line for the kernel. This is not rocket science and has been done before. It would be very useful for other purposes as well and greatly simplify the IPL from NSS.

To make the facility more generic, it must be possible to pass the command to be executed on the worker. By default, this would be the script to set up the network connection with the master, but it will also be possible to have a command executed without connecting the network. This results in a kind of background processing outside the virtual machine.

To copy a file system to a new disk, two minidisks must be linked and the `tar` and `untar` for the copy must be issued.

An elegant solution would be if the worker could mount part of the master's file system so that local scripts and commands in the master could be executed. Different levels of commands and libraries could make this very complicated.

Some complications with respect to UID and GID may occur when files in the target file system are modified or created.

The private IUCV network between each worker and the master is a secure solution and there are no security risks for the NFS export in the worker.

The DASD driver currently has a delay of 10 seconds in the startup. The reason for this is unclear. It should be possible to remove this delay or improve the process.

Most parameters will be fixed for each worker and could be taken from a file on the 191 disk using the cmsfs driver (or computed from the user ID of the virtual machine). The actual parameters for the command could be passed to the Linux boot scripts via CP Global variables (for example, the TAG of a virtual device).

There are no specific requirements for the level of Linux running in the worker. The only requirement is that it should be reasonably current, but it probably can be an off-the-shelf kernel with RAMdisk. With recent levels of Linux, the incompatibilities between different versions of the DASD driver seem to be fixed.

17.6 Synergy with CMS Pipelines

CMS Pipelines is the z/VM built-in productivity tool for CMS. Many CMS applications are written to use *CMS Pipelines* facilities. In the hands of an experienced plumber, *CMS Pipelines* is also very useful for ad-hoc analysis of the output of experiments, as we did while writing the redbook.

While *CMS Pipelines* design was originally mildly inspired by the concept of pipes in UNIX, it has been enhanced significantly beyond that. It features multistream pipes as well as pipes that dynamically modify the topology of the pipeline by adding segments to the running pipeline. *CMS Pipelines* has drivers to interface with various z/VM facilities.

The *CMS Pipelines* home page is hosted by Princeton University at

<http://pucc.princeton.edu/~pipeline>

The home page has pointers to several papers on *CMS Pipelines*. It also offers the latest version of the *CMS Pipelines* Runtime Library free for download to allow customers with slightly older levels of CMS to run the latest version of *CMS Pipelines* on their system. The z/VM documentation comes with two publications for *CMS Pipelines* users:

- ▶ *CMS Pipelines Reference*, SC24-5971
- ▶ *CMS Pipelines User's Guide*, SC24-5970

Experienced plumbers tend to prefer the documentation in the “*CMS Pipelines* Author's Edition,” which is available on the VM Collection CDs as well as on the home page:

<http://pucc.princeton.edu/~pipeline/pipeline.book>

In an earlier residency, John Hartmann, the author of *CMS Pipelines*, created the “Plumber's Workbench” (PWB). It is a workstation application with CMS in its back room. It allows access to all of *CMS Pipelines* from the workstation. PWB is available for OS/2 and MS Windows workstations. John Hartmann also worked on a PWB client for Linux (including Linux for S/390). This gives Linux applications access both to *CMS Pipelines* and to CMS applications.

We did some experiments with the code and it certainly works, though the syntax of the `vmpipe` command is error prone due to the overloading of the “|” character. To manipulate Linux data with *CMS Pipelines*, the PWB client sends the data over a TCP/IP connection to the PWB agent running in a CMS user ID. Since Linux for S/390 can have a fast connection to the CMS user ID, this is less likely to be a showstopper.

For people with the appropriate skills, it could be very attractive to use those skills for their Linux work. The ideal would be to have a Linux implementation of *CMS Pipelines*. This is currently not available.

17.7 Make DirMaint the registration vehicle

Several things need to be arranged in VM to create a new Linux image, as shown in 9.2, “Things to do for new Linux images” on page 190. It could be attractive to enhance DirMaint so that it takes the role of central registration vehicle in VM.

To do this, there would be a need for new options in the prototype files that invoke exit routines to do the “things” that are needed for the new Linux images. For example, a new statement in the prototype file could look like this:

```
ADDIP subnet-17
```

This could invoke an exit that allocates a new IP address for this image in a specific subnet, and creates the definitions in the TCP/IP configuration files, DNS, DHCP, etc. If real network interfaces are used, the exit should probably pass the correct DEVICE statements to DirMaint to have these included in the directory entry for the new image.

Because we do not know yet what is needed, a flexible implementation should make the statements, as well as the exit routines, user-defined.

The same processes obviously would take care of removing the definitions when the Linux image is deleted with DirMaint, or when the creation process is rolled back for some reason.

Discussion with people associated with DirMaint development showed they are aware of the need to make DirMaint assist in cloning Linux images. One of the possible enhancements could be to have the DATAMOVE virtual machine create new minidisks as a copy of an original disk rather than format them.



A

Linux Community Development System

The Linux Community Development System (LCDS) was created by a team of IBMers in the spring of 2001. Its purpose was to provide the open source community with free access to Linux on a mainframe. In this chapter, we describe the experiences and lessons learned.

Components of the system

The following sections discuss the system components.

Linux on a mainframe for free

The first component of the Linux Community Development System is the Linux part—in other words, making Linux systems on S/390 available to the open source community. Here is the invitation as it appears on the LCDS home page:

<http://www-1.ibm.com/servers/eserver/zseries/os/linux/lcds/index.html>

Welcome to the Linux Community Development System (the 'Service'), a Service provided by IBM. The Service provides you with access to a Linux on S/390 environment for the purpose of providing the Open Source community with a platform to develop, port and/or test drive your products or applications on this platform. We anticipate the majority of users to include entrepreneur developers/vendors that otherwise might not have the opportunity to test/port their code to the S/390 platform. However, we invite all interested parties that meet the established terms and conditions to register and experience 'Linux for S/390'.

Community: the global response

The LCDS home page opened for business on May 22, 2001. In three days the page had received 27,000 hits. Not all of those hits led to a request for a Linux system, but as of late July 2001, there were a little over 600 images running on the system. The users are a truly global community, representing these countries:

- ▶ Angola
- ▶ Argentina
- ▶ Australia
- ▶ Austria
- ▶ Belgium
- ▶ Brazil
- ▶ Bulgaria
- ▶ Canada
- ▶ Chile
- ▶ China
- ▶ Croatia
- ▶ Czech Republic
- ▶ Denmark
- ▶ Dominican Republic
- ▶ Egypt

- ▶ Estonia
- ▶ Finland
- ▶ France
- ▶ Germany
- ▶ Great Britain
- ▶ Greece
- ▶ Hungary
- ▶ Iceland
- ▶ India
- ▶ Indonesia
- ▶ Ireland
- ▶ Israel
- ▶ Italy
- ▶ Japan
- ▶ Malaysia
- ▶ Mexico
- ▶ Netherlands
- ▶ New Zealand
- ▶ Norway
- ▶ Pakistan
- ▶ Peru
- ▶ Poland
- ▶ Romania
- ▶ Russia
- ▶ Singapore
- ▶ South Korea
- ▶ Spain
- ▶ Sri Lanka
- ▶ Sweden
- ▶ Switzerland
- ▶ Taiwan
- ▶ Thailand
- ▶ Turkey
- ▶ Ukraine
- ▶ United Arab Emirates
- ▶ United Kingdom
- ▶ United States
- ▶ Venezuela
- ▶ Vietnam
- ▶ Yugoslavia

Development: what is being tried

The users represent a global range of applications as well as geographies. Film production, aerospace, pharmaceutical, insurance and banking companies are participating, as well as many universities from around the world, and gnu.org. The following list shows the variety of reasons users gave for wanting a Linux system on S/390:

- ▶ Rotund prime sequencing
- ▶ Samba, Apache, Sendmail
- ▶ Digital document system
- ▶ C++ compilers, C code front ends, general tests
- ▶ Cryptography, security, intrusion detection
- ▶ Java, XML, XMK
- ▶ Wireless, voice, embedded devices
- ▶ Working on, experimenting with, testing...
- ▶ Want to see:
 - If this works
 - How easy it is
 - If I can port
- ▶ Pong
- ▶ Oriental herbology

System: what it is being run on

The operating system software that runs the LCDS is z/VM. This allows hundreds of unique Linux images to exist on one physical machine. The hardware is S/390 technology, not zSeries. The techniques learned and refined on this system will deliver even better results as they are deployed on 64-bit, zSeries hardware. The full details of the system are described in the following section.

Technical implementation of the LCDS

Hardware specifications

CPU

The LCDS is hosted on a 9672 G6 Model ZX7 machine. This hardware has IEEE floating point and is 31-bit technology. It is a 10-way processor, with 32 GB of memory. It is part of the S/390 family.

DASD/disk capacity

The Linux images and z/VM operating system have access to disk storage on a Shark (Enterprise Storage Server) Model 2105-F20 configured with 2.1 terabytes of capacity.

Network

The network design of the LCDS had to accommodate an interesting mix of legal and physical characteristics. The z/VM operating system had to be accessible to IBM employees who were setting it up and administering it on the internal IBM network. The Linux guests had to be on a direct connection to the Internet, and there could be *no* connection between the two (Internet and internal IBM). The physical constraints included the use of an existing T1 connection to the Internet over a 3172 LAN Channel Station. Although there was only one physical connection to the Internet, the design had to accommodate hundreds of unique IP addresses—one for each Linux guest.

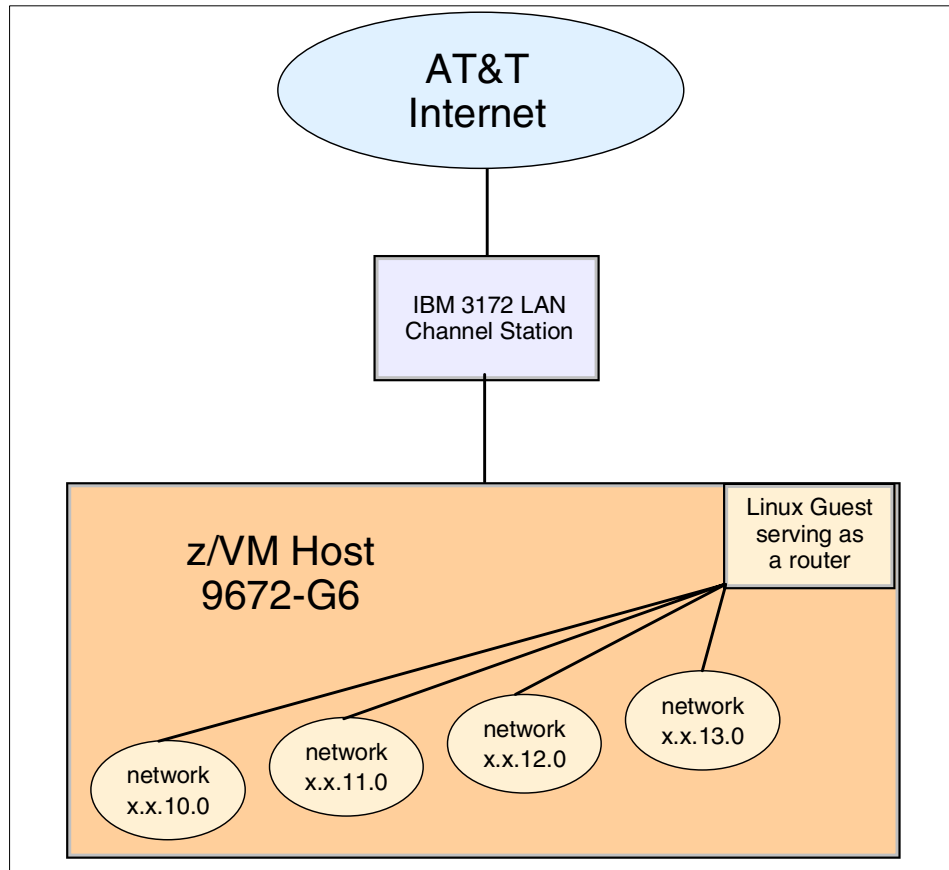


Figure 17-1 LCDS connection to the Internet

A total of 2000 IP addresses were obtained. These addresses are spread across 20 routers in groups of 100. All the routers to the Linux guests are themselves Linux guests. This allows the exploitation of z/VM architecture by networking all the Linux guests through virtual, or software-defined, connections. The LCDS uses IUCV-type connections, as opposed to Virtual Channel To Channel (VCTC). The IUCV connections were found to initialize quicker and recover more automatically, for example after a reboot.

One task of the Linux routers is to do Network Address Translation (NAT). Within the LCDS virtual network, the Internet addresses are translated to class A 10.x.x.x addresses. This means the 2000 purchased Internet addresses do not have to be consumed by internal routers, name servers, and gateways.

The other advantage of the 10.x.x.x addresses is that they cannot be routed. The network traffic isn't going to "leak". Within the LCDS network, ICMP is turned off. This means ping cannot be used to discover the network topology. Once users of the LCDS are logged on to their Linux image, they can access any address on the Internet using any protocol that is authorized on the target server.

Within the LCDS network, the Linux guests are architecturally isolated from each other. This isolation is achieved under the control of z/VM. Each Linux guest can only have access to a resource that is defined to it. There are no pathways available for a Linux guest to communicate with, access, or modify any resource that is not defined to it.

The domain name server is within the scope of the LCDS network. Since this is a very dynamic setup, with Linux images being defined by the dozen, it seemed better to control the configuration of the name server within the LCDS staff.

Back-routers are used to contain and shorten the network traffic among the Linux guests. They allow an asset to be shared without exposing the traffic (and asset) to the Internet.

As mentioned before, ping (ICMP) is turned off. The only access to log in as root is through SSH access. Telnet and ftp are enabled, and can be used once you have successfully logged in.

Staff and processes

Team

The Linux Community Development System was designed, implemented, and is administered by a small team based mostly in Endicott, NY. Everyone involved made their contributions while still keeping their day job, in keeping with the open source community tradition. They are officially part of the Advanced Technical Support (ATS) organization. The team members are:

John Sutera	Manager
Bill Bitner	Performance, VM Development
Pamela Bryant	PROP
Steve Gracin	Networking, RedHat, WebSphere Application Server
Stanley Jones Jr	Registration, Lotus Notes work flow
Bob Leicht	Enrollment, SSH, System
Richard Lewis	VM, Networking, System
John Schnitzler Jr	Hardware, IOCP, SSH
Jon vonWolfersdorf	Networking, LCDS Home Page

Pam Bryant and Richard Lewis are based in Gaithersburg. The rest of the team is in Endicott, as is the hardware.

Register

Access to the LCDS is open to anyone (except internal IBMers), anywhere in the world. A form is provided on the Internet, asking for a minimal amount of information, including the user's purpose in testing Linux on a mainframe. Once the form is filled out, a Lotus Note is sent to an administrative ID. This ID is monitored by two team members. When a request for a Linux system is received, it is converted to an entry in a Lotus database. The request is reviewed, then accepted or rejected. An accepted request triggers a Lotus Notes agent to assign an ID and password, which are sent to the requestor.

Generate a system

One requirement for access to the LCDS is SSH (Secure SHell) encryption. The requestor is responsible for getting a terminal emulator that is SSH capable, and for generating public and private keys.

Note: The freeware program PuTTY does both, with the PuTTYgen.exe and putty.exe programs. They can be found on the Web at:

<http://www.chiark.greenend.org.uk/~sgtatham/putty>

When requestors of a Linux system receive the note with the ID and password, they use these to sign on to a secure Web page. This Web page is on the LCDS z/VM system where the Linux guests are defined and run. The ID and password are validated, then a new Linux guest is generated. The automated generation process creates a VM guest Linux user, with associated disk space, virtual memory, and a network address.

The CLONEM exec takes advantage of CMS techniques such as PIPES. It also uses drivers from the open source community. Rick Troth of BMC Software has written a driver that allows CMS files to be read from Linux, and Neale Ferguson of Software AG has written a device driver that allows CP commands to be issued from Linux (see Appendix B, "Using the hcp command" on page 437 for more details.) This permits an architecture where customization information is managed from z/VM, which allows one person to administer hundreds of systems from one central focal point.

There are two other key pieces of interface technology. The first is the Web page mentioned earlier. It is CA's VM:Webgateway. It allows the requestor information to be collected in the z/VM environment and propagated to each new Linux guest. The second interface is specialized customization to the boot process of Linux. Richard Lewis of the IBM Washington System Center created a shell script that runs very early in the boot process, before the network connection is started.

The script reads the customization information on the Linux guest's A-disk (gathered from the Web page). It assigns the correct network address and essentially answers the questions a user answers when installing a Linux distribution.

The first access that requestors of the Linux guest have to their system is when they SSH into it as root. They do not have to go through SuSE or Turbo panels to configure the distribution. This automated process provides a high level of security, as it shields the user from the underlying z/VM system, prevents network configuration errors (accidental or deliberate), and reserves control of the configuration process to the system owner.

Help, Support

When someone downloads Linux and installs it on their home PC, they understand they are on their own as far as technical support is concerned. It is their responsibility to find (or contribute!) answers through the use of news groups and mailing lists. The same is true on the Linux Community Development System. Free access to a somewhat hard-to-acquire and expensive hardware platform has been provided by IBM. The goal is to prove that Linux on the mainframe is the same as any other Linux. "The same" includes the same style of support. There is a forum on the LCDS Web site, where community members share their experiences. They can describe problems they have encountered, and may receive technical help. However, no one is restricted to using only that forum, and there is no guarantee they will get an answer there. Technical support comes from the open source community at large.

It often happens that a Linux system crashes after some user tests or modifications. Since the requestor of an LCDS Linux guest machine does not have access to the "big red switch" (the power switch) to do a reset of the hardware, it was necessary to provide a way to reboot a seriously incapacitated Linux guest. The REBOOT service machine is accessed using SSH and accepts the name of your Linux machine as the login ID. There are four options to choose from. You can:

- ▶ Exit without doing anything.
- ▶ IPL your Linux with a rescue system. This reboots a Linux rescue system from a RAM disk.
- ▶ IPL your Linux from a specified device. This performs a normal reboot.
- ▶ Force your Linux offline; do not restart it. This forces a shutdown -h now, which will then require a reboot with either option 2 or 3.

As of the time of writing, there were over 600 Linux guests running on the LCDS. At no time has a reboot of one Linux guest impacted any of the others. The architecture of z/VM allows complete freedom for individual users to try any high-risk change they like, while completely isolating the other Linux machines from any impact of that change.

Monitoring

The LCDS usage has been monitored both interactively and using accumulated statistics. The historical data has turned up interesting facts, such as z/VM setting a new record for paging of 259,000 pages per second. The previous record was 45,000 pages per second. This is a testament to the robustness of the z/VM architecture.

Network monitoring showed no particular bottlenecks. Once network traffic is within the virtual network of z/VM and the Linux routers, communication is at very high speed and bandwidth. The physical limitation is the capacity of the T1 line. A sample graph of the daily usage is shown in Figure A-1.

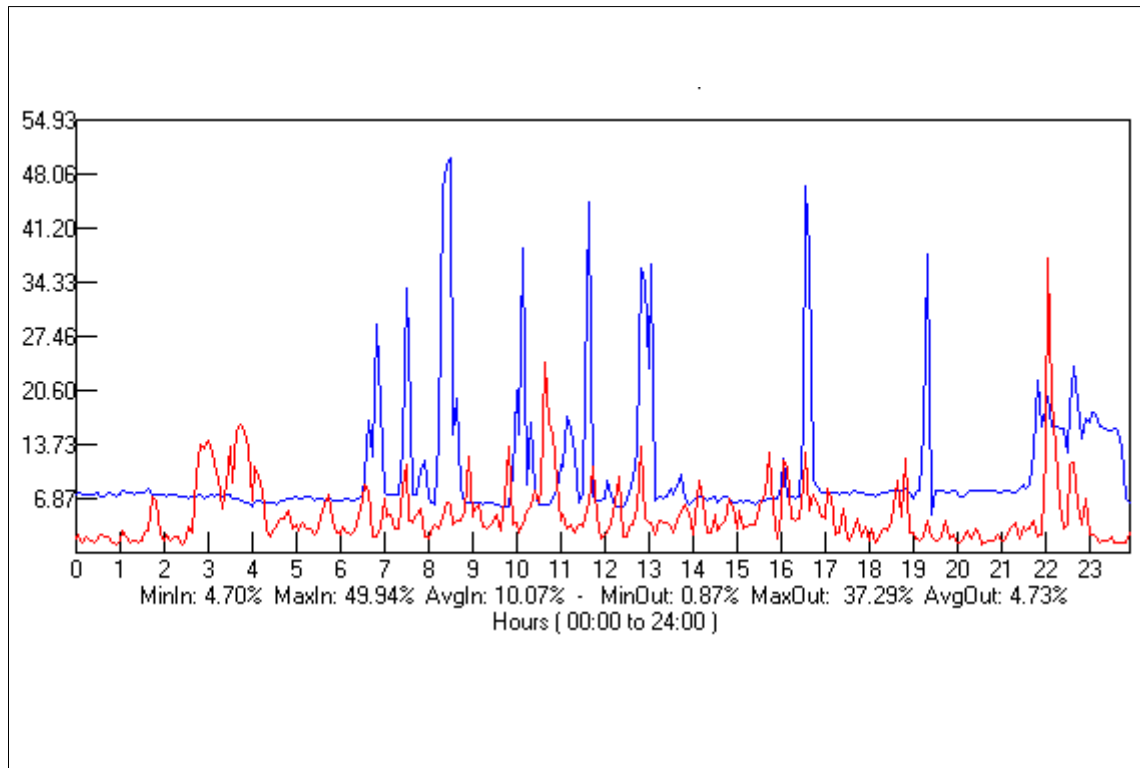


Figure A-1 percent utilization of the T1 line

The higher line (blue) is inbound traffic, and the lower line (red) is outbound traffic. The usage pattern is a pretty typical workday series of peaks and valleys as people come in to work, come back from lunch, and hurry to finish something at the end of the day. The graph of weekly activity showed that Saturday was one of the busier days—an interesting finding.

Interactive observation of the Linux guests was done by the LCDS staff. On occasion, a spike in CPU activity would be noted and investigated. In a production ISP/ASP environment, this process could be automated. The CP ACCOUNT facility also collected the CPU activity for each guest, and could have been used to track high CPU usage.

One area that several people would like to explore is the possibility of using the Monitor facility. This system interface is used in CMS to report more detailed usage information from within the virtual machine. CMS reports statistics on its Shared File System, as an example. Neale Ferguson has started work on a Linux driver to talk to the Monitor Application Data interface. Some members of the LCDS team hope to cooperate in refining this driver.

Termination

Users are given access to a Linux guest for 30, 60, or 90 days. At the end of the time they requested, the image is deleted from the system. It is the users' responsibility to retrieve any data they wish to keep. The automated process to return resources to the system is fairly basic, since there is no requirement for any information to be preserved.

Evolution and lessons learned

The LCDS has been a very dynamic experiment, and a fast-changing environment. A lot has been learned, both about z/VM running Linux guests, and about the nature of the Linux kernel. Several refinements to z/VM tuning were made, and there was at least one contribution to the Linux kernel.

z/VM

The IBM labs have been doing validation of the early Linux code drops, even before they go to the various distributors. The LCDS staff has been actively engaged in that validation. Two areas of interest have been the DASD drivers and how Linux behaves with mini-disk caching. There was an iteration of the kernel that did not respond well to mini-disk caching, but that was quickly resolved. The code drops that are being tested as of this writing will be out in the fall of 2001.

z/VM Release 4.10 includes an enhancement to CCW translation. Code was written for VSE guests that improved I/O to DASD devices. (VSE is another operating system.) This fast path code was only available for DASD, since VSE systems typically do a lot of data processing, and very little network activity. IBM developers working with Linux under z/VM realized that although VSE did very little network activity, Linux does a lot of network activity. They thought of making the I/O commands to network devices eligible for the fast path code. This was done for LAN Channel Station (LCS) and CTC connections and a 40% improvement in processor efficiency for network I/O was achieved for Linux guests.

Linux

The Linux kernel has a bit of logic that wakes up to check for work. It is referred to as *jiffies* or the *jiffies timer pop*. This results in wasteful overhead on a mainframe processor that is optimized to respond to interrupts. David Boyes of Sine Nomine Associates has experimented with altering the Hz value in the Linux kernel. The default value is 100, but it has been set to a value such as 16. This means more useful work is done, and there is less dispatching of a Linux machine that has no work to do, simply to check for work. Setting the Hz value too low can be a problem. Responsiveness goes down, and some things stop working. At this time the LCDS timer is set to its normal default value of 100, in order to maintain the consistency of Linux on other platforms.

A patch has been submitted to the Linux organization that implements a much different scheduling technique. It is not in the platform-dependent code. It would affect all platforms, and is designed to help all platforms that have multiple processors, but it has not been accepted into the kernel. Users or distributors do have the option of including the patch, which applies to Linux 2.4. There is a great deal of interest in this area of the code, so a lot of innovation can be expected. The Linux news groups and mailing lists will have the most current information.

Tuning both

One problem that was encountered very quickly was default settings for **cron**. There is a security package that scans for trivial passwords, which by default was started at midnight. On a single Linux system, this is a very conscientious thing to do. When hundreds of Linux guests on the same hardware all do the same thing, at the same time, it is a very bad thing to do! The system spiked to 1000% busy (which means all 10 CPUs were at 100% utilization) and paging went to 259,000/s. The system did not crash. Eventually all the Linux guests completed the scan for trivial passwords, and CPU busy returned to normal. It took a bit of investigation to discover the cause of the activity, but it was simple to fix. As upgrades are made and Linux is reinstalled, the defaults in **cron** are checked for tasks that should not be scheduled to run.

A characteristic of Linux is that the more memory it has allocated, the more it will use that memory for file caching. In some architectures, this is very desirable. However, when running under z/VM, it is more efficient to use the mini-disk caching capability. In fact, when a change was made to the LCDS system to reduce the amount of mini-disk caching storage and give it to paging, performance actually got worse. Paging and file I/O are covered extensively in Chapter 3, “Virtual server architecture” on page 45.

Structuring the file system to save space

One concern the LCDS staff had was to not use up the available disk space any faster than could be helped. After some experimentation, the structure shown in Figure A-2 on page 434 was chosen. The greatest amount of file space is used in the part of the directory tree under /usr. Therefore, much of that part of the file system is mounted read only (r/o).

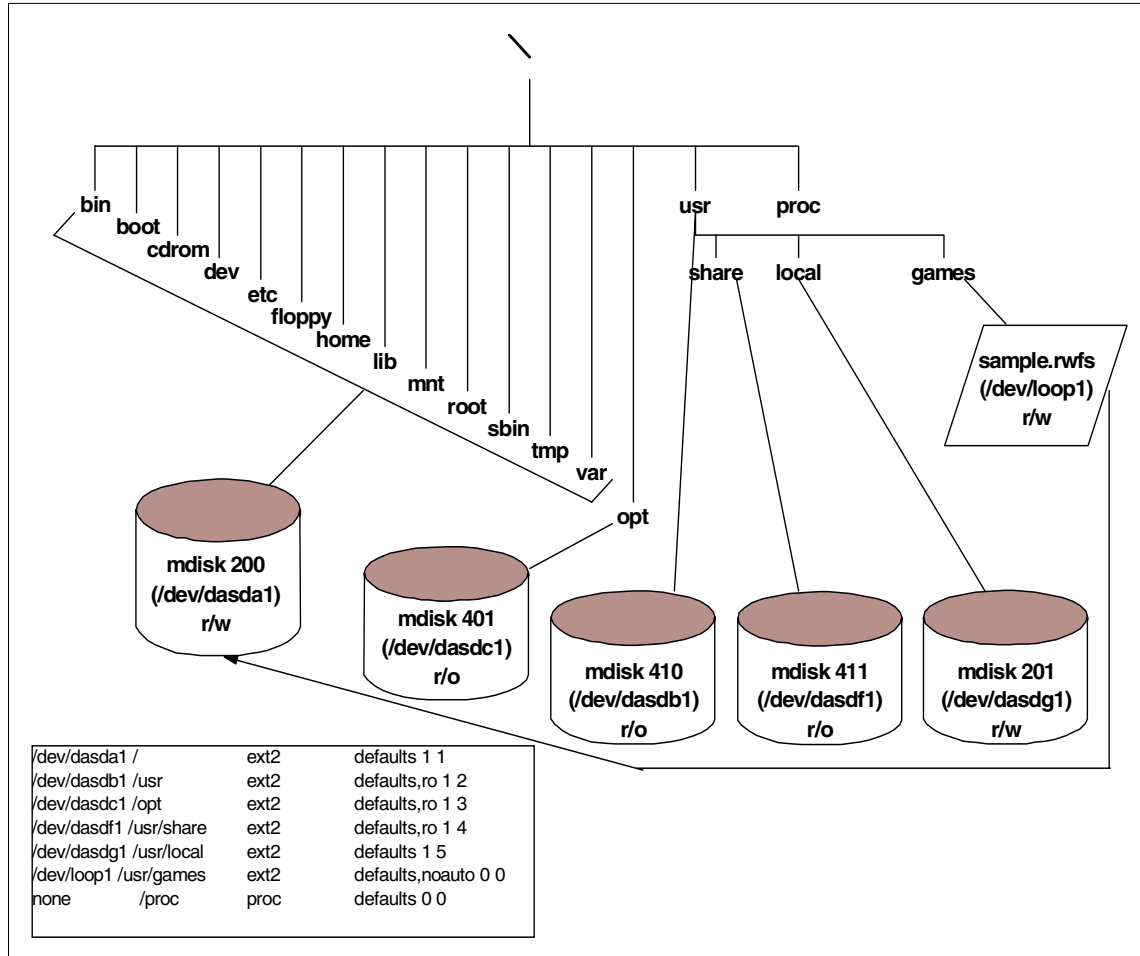


Figure A-2 LCDS file system structure - r/o and r/w files

The attribute of being read/only is enforced at the z/VM level. Editing the /etc/fstab file and changing /usr to be read/write does not affect access to the underlying physical device. To provide users with their own, private read/write files, a separate device is mounted at /usr/local. This allows about 80% of the file system to exist in one copy, shared read/only, by all 600 Linux guests.

Perhaps the most outstanding result of running the LCDS has been to witness the integrity of the architecture. The inadvertent stress test that drove the system to 1000% busy did not result in an outage. We have seen how z/VM ensures the isolation of each server. This allows for the most effective economy of scale in consolidating servers in one hardware footprint, while at the same time permitting owners of the Linux guest the same freedom and autonomy they could have on a workstation-based server.

Summary

There are many options to the S/390 Linux architecture and system design that may be feasible for accommodating multiple Linux guests. The team utilized existing hardware, software, Linux distributions and network topology that was available. The LCDS demonstrates S/390 and z/VM versatility, strength and security within an existing I/T environment that is worldwide in scope and responsiveness.



B

Using the hcp command

Neale Ferguson of Software AG wrote the package `cpint` for Linux for S/390 to provide an interface to some CP functions. One of the components is the `hcp` command, which uses the Diagnose 8 interface to issue CP commands on behalf of the guest virtual machine that runs the Linux image. The `cpint` package for the 2.2 series kernel can be downloaded from the following Web site:

<http://penguinvm.princeton.edu/programs/cpint.tar.gz>

To use the package you need to compile the programs and install them on each of the systems where you want to use the package. This can be very impractical if you have many different systems. It is possible to package the compiled binaries and installation script in an rpm package that can be installed along with the other packages on each Linux system.

Creating an rpm package for cpint

An rpm package consists of the source files, patches, and the spec file that describes how to apply the patches, how to build the binaries and how to install the binaries on the system.

For a Red Hat installation, the rpm files reside in the `/usr/src/redhat/` directory. In a SuSE installation, these files are in the `/usr/src/packages/` directory.

To build an rpm package from the source files, we need to create a spec file:

Example: B-1 The cpint.spec file

```
Vendor:      Rob van der Heij
Distribution: SuSE Linux 7.0 (s390)
Name:        cpint
Packager:    rob@rvdheij.com

Copyright:   GPL
Group:       Applications/System
Provides:    cpint
Autoreqprov: on
Version:     2.2.16
Release:     3
Summary:     Device driver for CP diagnose functions
Source:      http://penguinvm.princeton.edu/programs/cpint.tar.gz1
Buildroot:   /var/tmp/cpint-root/
```

```
%define cpintmajor 254
```

```
%description
```

```
Author: Neale Ferguson <Neale.Ferguson@SoftwareAG-USA.com>
```

The cpint package provides the cpint device driver that communicates with VM/ESA through the CP Diagnose interface. Most useful right now is the diagnose 8 which allows the virtual machine to issue CP commands to query or modify the status of the virtual machine.

```
%prep
```

```
%setup -n cpint
```

```
%build
```

```
make
```

```
%install
```

```
linuxvers=`uname -r`
```

```
rm -rf $RPM_BUILD_ROOT
```

```
mkdir -p $RPM_BUILD_ROOT/dev
```

```
mknod $RPM_BUILD_ROOT/dev/cpcmd c %{cpintmajor} 8
```

```
chown root:wheel $RPM_BUILD_ROOT/dev/cpcmd
```

```
install -p -D hcp $RPM_BUILD_ROOT/sbin/
```

```
install -p -D cpint.o $RPM_BUILD_ROOT/lib/modules/$linuxvers/misc/
```

```
%post
```

```
if grep " cpint" /etc/modules.conf > /dev/null
```

```
then
```

```
    true
```

```
else
```

```
    echo "alias char-major-%{cpintmajor} cpint" >> /etc/modules.conf
```

¹ Unfortunately Neale packaged also binaries in his tar file so we untarred it, did a `make clean` and then tarred it again.

```

        /sbin/depmod -a
    fi

%postun
if grep -w "cpint" /etc/modules.conf > /dev/null
then
    grep -v -w "cpint" /etc/modules.conf > /etc/modules.conf.new \
    && install -b -p /etc/modules.conf.new /etc/modules.conf
    rmmmod cpint 2> /dev/null
    /sbin/depmod -a
fi

%files
%attr(600,root,root)/dev/cpcmd
/sbin/hcp
/lib/modules/%{version}/misc/cpint.o

%changelog
* Mon Jun 11 2001 rob@rvdheij.com
- Fixed incorrect mode for /dev/cpcmd node
* Fri Jun 08 2001 rob@rvdheij.com
- Use modules.conf rather than conf.modules

```

The next step is to copy the cpint.tar.gz (and optionally any patch files for the package) into the SOURCES/ directory, and then build the binaries rpm.

```
# rpm -bb cpint.spec
```

This creates the binaries rpm in the SPECS/s390 directory. To package the source and patches with the spec file, you create the source rpm package. The **rpm -ba** command creates both binaries and source rpm.

Installing the cpint rpm package

When the binaries rpm is created, the install is very simple:

```
# rpm -Uvh cpint-2.2.16-3.s390.rpm
```

If you do this on another system, you don't even have to copy the rpm file over by hand. A single **rpm** command can take care of the FTP and install (provided you put the rpm package in a place that can be reached via FTP):

```
# rpm -Uvh ftp://hostname/path/cpint-2.2.16-3.s390.rpm
```

Using the hcp command

Because the install scripts in the rpm package also register the major number for cpint in /etc/modules.conf, the kernel module will be loaded automatically when you issue the **hcp** command:

```
# hcp q t
TIME IS 20:00:55 EDT FRIDAY 07/13/01
CONNECT= 99:59:59 VIRTCPU= 077:40.06 TOTCPU= 098:48.55
```

Seeing this will make the average VM user probably feel at home immediately, but there are a few gotchas to watch out for. Make sure you specify the command in double quotes when necessary:

```
# hcp m * test
HCPMSG020E Userid missing or invalid
```

What happened in this case is that the shell substituted the "*" with the list of files in the current directory, as it does with each shell command. Depending on the number of files in the directory, you may also get another indication of the problem:

```
# hcp m * test
Write: Cannot allocate memory
```

Another one to watch out for is typing the **hcp** command without any parameters. This causes the virtual machine to drop in CP READ and get logged off after some time.



Using the Linux 2.4 kernel

In this appendix we describe how we upgraded to the Linux 2.4.5 kernel. There are different ways to get a Linux-2.4.5 kernel running on your system. We experienced, however, that not all obvious routes to that goal were easy to travel for someone with minimal Linux skills.

On June 29, just before we started our work on this Redbook, another “code drop” was done by the IBM team in Boeblingen. Part of the patches published on the IBM DeveloperWorks Web site were the S/390 patches for the Linux-2.4.5 kernel. This latest patch is called linux-2.4.5-s390, just like the previous one published on June 13. The size of the patch, however, has grown from 25 KB to 1.5 MB. The “readme” with the patch explains the new function introduced with this patch. One of the biggest chunks in this patch appears to be a rework of the DASD driver.

Note: We were unable to get a system running with this new DASD driver on existing disks. Because of this we decided to stick with the June 13 version of the patch (which is not available on the Web site any more). Later experiments suggest that this may have been caused by devfs unintentionally being enabled. Unfortunately, we did not have time to retrace our steps and repeat the process.

Can of worms: Somewhere along the line, the ELF architecture number for s390 has changed from 0xa390 to the now official 0x0016. The binaries contain this number to prevent binaries from another architecture to be executed on your system (which is good). This new number is defined in binutils and will cause the loader to produce binaries marked with the new number. At the same time the kernel was changed such that it can support both types of binaries, so this is upwards compatible.

This same architecture code is also present in the kernel modules, including the object code only (OCO) ones for the OSA interface. You need to get the correct version of the modules to make sure they can be loaded or you will be without your network connection. The **depmod** command will warn you about the architecture difference.

Silo and the DASD driver

Changes to the `ioctl()` functions in the DASD driver make the silo from your 2.2.16 kernel fail with the 2.4.5 kernel. From the `/usr/src/linux` directory you need to **make silo** again. Unfortunately, the changes in silo and the DASD driver are incompatible, so you should keep both versions of silo around. We renamed the old silo to `/sbin/silo-2.2.16` and copied the new one as `/sbin/silo-2.4.5`. This way you are constantly reminded to use the correct version.

There is a bug in **silo** in that it fails to clean up the temporary device node. This happens when the version of **silo** does not match the DASD driver in the kernel. The error message to recognize is shown in Example C-1. You need to **rm /tmp/silodev** to run **silo** again.

Example: C-1 Error message indicating the left-over device node

```
silo.c (line:428) 'mknod ("/tmp/silodev", S_IFBLK | S_IRUSR | S_IWUSR,
fst.st_dev)' returned 17='File exists'
```

If you want you can even write your own shell script `/sbin/silo` to pick the correct version based on the level of the kernel. The shell script is shown in Example C-2.

Later versions of **silo** support a configuration file. We did not use this because it mainly caused us problems and did not play well with alternate boot volumes.

Example: C-2 Shell script to pick the correct version of silo

```
#!/bin/sh
$0-`uname -r` $*
```

Another bug in **si1o** is that it leaves temporary files like `parm.6Djosg` in the `/boot` directory. You may want to enhance your **si1o** script such that it removes these temporary files before running **si1o** again.

Attention: When you copy a new kernel to your `/boot` directory, you should avoid overwriting the active kernel (for example, by avoiding the default name “image”). The way **si1o** works is that it records the block numbers of the kernel in a special bootmap file. When Linux is booted, these blocks are read into memory from disk without checking the directories on the disk. If you overwrite your active kernel, these blocks become unused and may be overwritten later by other files. If you do not run **si1o** at this point (for example, because it does not work on your new kernel), you are close to being unable to boot your old kernel again. For peace of mind you should implement the process outlined in 9.5, “Using an alternate boot volume” on page 206 when you play with kernel upgrades.

In the latest 2.4.5 patches the **si1o** command has disappeared. We have not yet worked with **zip1**, which should perform the same function.

Steps to upgrade SuSE 7.0 to Linux-2.4.5 kernel

Using the intermediate May 7 version of SuSE 7.0, you should be able to get 2.4.5 running following this recipe.

Untar the linux-2.4.5 sources

Get the `linux-2.4.5.tar.gz` file from the Web at:

<ftp://ftp.kernel.org>

or your other favorite FTP site and put that in `/usr/src` for now. You probably should save the current 2.2.16 sources as well, as follows:

```
# cd /usr/src
# rm linux
# tar xzf linux-2.4.5.tar.gz
# mv linux linux-2.4.5
# ln -s linux-2.4.5 linux
```

Apply the IBM patches

You need to download `linux-2.4.5-s390.tar.gz` from the DeveloperWorks Web site and **untar** that. The following assumes you do this in `/root/patch245/`:

```
cd /usr/src/linux
patch -p1 -i /root/patch245/linux-2.4.5-s390.diff
```

The CTC driver at this level of code has a small problem that, at the very least, floods the console with error messages. There were also reports about sudden hangs of the driver, which may be caused by the same bug. The patch for this bug is in Example C-3. It must be applied in the same way as the linux-2.4.5-s390.diff patch.

Example: C-3 Patch for the ctc driver in 2.4.5

```
--- boelinux-2.4.5/drivers/s390/net/ctcmain.c   Wed Apr 18 23:40:07 2001
+++ linux-2.4.5/drivers/s390/net/ctcmain.c     Wed Jun 20 23:48:45 2001
@@ -988,7 +988,7 @@
                                first = 0;
                                }
                                atomic_dec(&skb->users);
-                               dev_kfree_skb(skb);
+                               dev_kfree_skb_any(skb);
                                }
                                spin_lock(&ch->collect_lock);
                                if (ch->dccw) {
```

Restriction: The IUCV driver in this 2.4.5 level appears to be broken as well in that it caused a kernel oops with the first IP packet transmitted. A new version is being tested, but for the time being you will need to use CTC instead (and apply this patch).

Copy the config file from your old source tree

You probably want the new kernel configured similar to what you had on the previous one. If you have been building the 2.2.16 kernel yourself, the config file for the kernel will be the .config file in /usr/src/linux-2.2.16SuSE. If you did not do this already, then you find a copy of it as image.config in your /boot directory. Copy the config file to /usr/src/linux/.oldconfig and run **make oldconfig** now. You will be prompted for the new configuration options that were not in your old kernel.

Build the kernel and modules

You can now build the kernel and modules, starting in the /usr/src/linux/. This process may take a while, depending on the processing resources you have available:

```
# cd /usr/src/linux
# make image modules modules_install
```

Because of the incompatibility with **sil0** and the new DASD driver as explained in “Silo and the DASD driver” on page 442, you should also build **sil0** and make yourself the shell script for it:

```
# make sil0
```



```
# cp arch/s390/tools/silo/silo /sbin/silo-2.4.5
```

Prepare the boot disk

Unless you are very certain of what you are doing, you should prepare yourself an alternate boot disk as explained in 9.5, “Using an alternate boot volume” on page 206 and mount this at /boot. Copy the new kernel to the /boot directory and run **silo** with your normal parameter file:

```
# cp System.map /boot/System.map-2.4.5
# cp arch/s390/boot/image /boot/image-2.4.5
# cd /boot
# silo -d /dev/dasd? -f image-2.4.5 -p parmfile
```

Install your network driver

Depending on the type of network interface used in the system, you may need to get the version of one of the Object Code Only network drivers from the DeveloperWorks Web site. You can create a /lib/modules/2.4.5/oco directory for these (if you put them in the kernel directory they may disappear when you run a **make modules_install** again). If you add the network drivers after running the **make modules_install**, then you need to run **depmod -a** again.

Attention: Do not pick the 2.4.5-2 version of the network drivers. These versions use the new 0x0016 ELF architecture code which does not work with the modutils and kernel we have here.

If you use a CTC or IUCV connection to the system, the driver will have been built already by the **make modules** step.

Shut down and reboot

You can now shut down your Linux system and boot from the new boot disk. Though the messages during boot will be slightly different, things should look fairly normal with this boot. If it fails to get your network driver going you should log on from the virtual console as root and run **depmod -a** again.

About the timer patch

During the weeks we were writing this redbook, there was a lively discussion on the Linux-390 mailing list about the possible benefit of the so-called “timer patch.” A few months before that Martin Schwidewsky from IBM Boeblingen posted a possible way in which Linux for S/390 could do without the 10 mS timer tick. The description of the proposed patch can be found on the Web at:

<http://lwn.net/2001/0412/kernel.php3>

The patch removes the global variable “jiffies” that is used by the kernel and device drivers for time measurement. Instead, it provides a macro called “jiffies” to compute the current value using the STCK instruction. Though Martin posted portions of the patch to the kernel mailing list, there still is work to be done if you want to implement this yourself.

For the work on the redbook we had access to a preliminary version of the patch that Martin proposed. This patch is not part of the mainstream kernel sources. It is unclear whether this is going to happen at all.

The patch must be applied just like the linux-2.4.5-s390.diff, on top of what you have there now:

```
# cd /usr/src/linux
# patch -p1 -i /root/patches245/timer-2.4.5-s390.diff
```

Unfortunately, the lcs.o driver for 2.4.5 has a dependency on the “jiffies” symbol that is removed by the timer patch. The qdio driver does not have this dependency, so we expect this should continue to work (but we were not able to try it). Since the source for the lcs driver is not part of the kernel sources, we cannot rebuild it ourselves. Apart from the qdio driver the only alternative left now is the CTC driver.

The version of the timer patch that we used failed to export a symbol, thus causing unresolved references if you want to build the CTC driver as a module. The patch in Example C-4 fixes this problem. Apply it just like the timer patch before rebuilding the kernel.

Example: C-4 Exporting the “init_timer_cc” symbol

```
--- boeblinux-2.4.5/arch/s390/kernel/s390_ksyms.c Wed Apr 11 21:02:28 2001
+++ linux-2.4.5/arch/s390/kernel/s390_ksyms.c Sun Jul 29 17:07:02 2001
@@ -8,6 +8,7 @@
 #include <asm/checksum.h>
 #include <asm/delay.h>
 #include <asm/setup.h>
+#include <asm/io.h>
 #if CONFIG_IP_MULTICAST
 #include <net/arp.h>
 #endif
@@ -20,7 +21,7 @@
 EXPORT_SYMBOL(_zb_findmap);
 EXPORT_SYMBOL(__copy_from_user_fixup);
 EXPORT_SYMBOL(__copy_to_user_fixup);
-
+EXPORT_SYMBOL(__ioremap);
 /*
```

```
* semaphore ops
*/
```

After applying the timer patch you should run **make dep** again, followed by a **make image modules modules_install** as before. Then write the new kernel and map to your `/boot` directory and run **si1o**.

Using the device file system

The `/dev` directory as we know it contains the entries that describe the hardware devices for your Linux image. To be more precise, `/dev` contains an entry for each hardware device you could possibly have in your Linux image. You see lots of `/dev/dasd*` entries because some Linux image could have 20 disks and need an inode to access the device. To have all `/dev` inodes in the private disk space of each Linux image is difficult to maintain. It also is a waste of disk storage, even though the entries themselves are rather small.

It is not practical to put `/dev` on an R/O shared disk because some applications need to change the owner of the device entry. This is not possible on an R/O shared disk. For specific cases (like tty devices) there is a virtual file system, but a more attractive solution is the device file system (Devfs). The patches for Devfs were done by Richard Gooch and have been included in the 2.3.46 kernel.

Devfs is a virtual file system, similar to the `proc` file system mounted in Linux as `/proc`. The entries shown in such a file system with the `ls` command do not really exist. Specific entry points of the device driver for the file system (like `procf` or `devfs`) are called when a directory listing is required. Other entry points are called when you read or write a “file” in the virtual file system. If you type the command **cat /proc/uptime**, that value is computed because you want to display it. Just as `/proc` provides the peepholes to expose kernel variables to the applications, `devfs` shows the device configuration of the Linux image to the applications. The device file system is mounted on the `/dev` mount point, so with `devfs` running `/dev` contains only the devices present in your configuration at that moment.

The device drivers must be aware of `devfs`. They must call the correct `devfs` functions to register and they must provide the proper data structures to hold the device information. Whenever `devfs` needs information about a directory in the device file system, it invokes the appropriate functions of the device driver. For compatibility, a `devfsd` package is available that provides access to the old inodes through the new `devfs` interfaces.

The DASD driver with Linux for S/390 has been enhanced to support `devfs` as well. The difference is obvious when you use the `df` command to show the mounted file systems.

Example: C-5 Device names shown with devfs

```
tux60002:/proc # df
Filesystem          1k-blocks      Used Available Use% Mounted on
/dev/dasd/01A0/part1 174116         63256   101872   38% /
/dev/dasd/01A1/part1 1415848       1169352  174576   87% /usr
```

Because the DASD driver keeps the active devices in a specific subtree of the /dev file system, it is easy to see what disks are accessible to the Linux image.

Example: C-6 Full list of disk devices in the system

```
tux60002:/proc # find /dev/dasd
/dev/dasd
/dev/dasd/0200
/dev/dasd/0200/disc
/dev/dasd/0200/part1
/dev/dasd/01A0
/dev/dasd/01A0/disc
/dev/dasd/01A0/part1
/dev/dasd/01A1
/dev/dasd/01A1/disc
/dev/dasd/01A1/part1
/dev/dasd/01C0
/dev/dasd/01C0/disc
/dev/dasd/01C0/part1
```

The listing in Example C-6 shows the entries in /dev/dasd for this particular system. The DASD driver creates a subdirectory named after the device address. In this subdirectory is one entry for the raw device (the disc entry) and one for each partition on the disk. In addition to the subdirectories named after the device address, there is also a /dev/discs directory, shown in Example C-7, that lists the disks in sequential order. These entries are links to the corresponding entries in the /dev/dasd directory.

The /dev/discs directory is the standard Linux devfs structure. This is not specific to Linux for S/390. On another platform these entries would be links to SCSI or IDE devices.

Example: C-7 Available disk devices in /dev/discs

```
tux60002:/dev/discs # ls -l
total 0
drwxr-xr-x  1 root  root          0 Dec 31  1969 .
drwxr-xr-x  1 root  root          0 Dec 31  1969 ..
lr-xr-xr-x  1 root  root        12 Dec 31  1969 disc0 -> ../dasd/0200
lr-xr-xr-x  1 root  root        12 Dec 31  1969 disc1 -> ../dasd/01A0
lr-xr-xr-x  1 root  root        12 Dec 31  1969 disc2 -> ../dasd/01A1
```

The FAQ at the following Web site is recommended reading material if you want to get started with the device file system:

<http://www.atnf.csiro.au/~rgoach/linux/docs/devfs.html>

Installing the device file system on SuSE

We installed the device file system on a SuSE 7.0 system upgraded with a Linux-2.4.5 kernel as described in Appendix C, “Using the Linux 2.4 kernel” on page 441. The first step is to configure the kernel to support a device file system (but not enable the option to mount/dev automatically).

With this kernel active you can install devfsd (from the SuSE distribution). Because the device file system mounts on top of the /dev directory, anything in there becomes unaccessible. The devfsd package contains the devfsd daemon that provides the compatibility interface. This way the old programs will continue to work, even when you mount devfs over/dev.

Attention: The startup script /sbin/init.d/boot.devfs use by SuSE refers to a file /sbin/init.d/mygrep (because grep lives in /usr/bin, which is not yet available at that point during startup). The installation of devfsd did not put the file there, so we copied it over from /usr/share/doc/packages/devfsd/ to get it working

When you have verified that devfsd is working properly at the next reboot, you can enable the option devfs=mount in the kernel parameter file (and run `si lo` again).

Attention: We found that `mingetty` did not work with devfs for us, despite the comments about devfs in the code (it looks like support was added for some devices needed as virtual console, but not the /dev/console we have with Linux for S/390). This resulted in error messages about /dev/console permissions. The easiest way out for the moment was to remove `mingetty` and use `su login` instead in /etc/initab.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 453.

- ▶ *Linux for S/390*, SG24-4987
<http://www.ibm.com/redbooks/abstracts/sg244987.html>
- ▶ *Linux for zSeries and S/390: Distributions*, SG24-6264
<http://www.ibm.com/redbooks/abstracts/sg246264.html>
- ▶ *IBM @server zSeries 900 Technical Guide*, SG24-5975
<http://www.ibm.com/redbooks/abstracts/sg245975.html>
- ▶ *OSA-Express Implementation Guide*, SG24-5948
<http://www.ibm.com/redbooks/abstracts/sg245948.html>

Other resources

These publications are also relevant as further information sources:

- ▶ *z/VM V4R1.0 General Information*, GC24-5991
- ▶ *z/VM TCP/IP Planning and Customization*, SC24-5981
- ▶ *z/VM TCP/IP Programmer's Reference*, SC24-5983 (available softcopy only)
- ▶ *z/VM TCP/IP User's Guide*, SC24-5982 (available softcopy only)

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ The Linux for S/390 developerWorks page:
<http://www10.software.ibm.com/developerworks/opensource/linux390/index.shtml>
- ▶ The Coda Web site:
<http://www.coda.cs.cmu.edu>

- ▶ The paper “Linux IP Networking - A Guide to the Implementation and Modification of the Linux Protocol Stack” by Glenn Herrin, May 2000:
<http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html>
- ▶ The paper “Addressing Security Issues in Linux” by Mark Chapman, December 2000, and the paper “Linux Security State of the Union” by Robb Romans and Emily Ratliff, May 2001, both available at:
<http://oss.software.ibm.com/developer/opensource/linux/papers.php>
- ▶ The Linux kernel FTP site:
<ftp://ftp.kernel.org>
- ▶ The OpenLDAP organization:
<http://www.openldap.org>
- ▶ The NET-SNMP home page:
<http://net-snmp.sourceforge.net>
- ▶ Linux Devfs (Device File System) FAQ:
<http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html>
- ▶ Linux Documentation Project home page:
<http://www.linuxdoc.org>
- ▶ Advanced Maryland Automatic Network Disk Archiver (Amanda) home page:
<http://www.amanda.org>
- ▶ Gnuplot Central home page:
<http://www.gnuplot.org>
- ▶ Multi Router Traffic Grapher (MRTG) home page:
<http://people.ee.ethz.ch/~oetiker/webtools/mrtg/>
- ▶ NetSaint home page:
<http://www.netsaint.org>
- ▶ OpenAFS home page:
<http://www.openafs.org>
- ▶ Samba home page:
<http://www.samba.org>
- ▶ Bynari home page:
<http://www.bynari.net>
- ▶ A resource for learning how the kernel works with regard to TCP/IP:
<http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html>

- ▶ CMS Pipelines home page:
<http://pucc.princeton.edu/~pipeline>
- ▶ Dante home page:
<http://www.inet.no/dante>

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Special notices

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

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

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

Index

Symbols

/dev/discs 448
/etc/crontab 271
/etc/init.d/network 251
/etc/pam.d 118
/etc/pam.d/login 365
/etc/passwd 117
/etc/rc.config 251, 304
/etc/route.conf 247, 251
/etc/shadow 117
/etc/smb.conf 352
/usr/afs/bin/klog 378
/usr/afs/etc/ThisCell 373
/usr/afs/local/BosConfig 366
/usr/vice/etc/CellServDB 373

Numerics

2064 15, 120
3746 76

A

Access Control List (ACL) 378
ACCOUNT 303
accounting
 Linux process 303
acct package 304
Active Directory 349
 joining 352
Address Resolution Protocol (ARP) 99
AFS 49
 Cache Manager 374
 client functionality 372
 completing installation 377
 configuring the cache 374
 configuring top levels 378
 defining cell name 366
 defining encryption key 369
 loading into kernel 383
afsd 373, 374
alternate boot volume 206
amadmin 284, 289
Amanda 138, 269

 backup and recovery scenarios 292
 backup via SMB 277
 disklist 273, 276
 using a tape changer 277
 using with cron 284
amanda.conf 273, 278
amandad 271
amcheck 280
amdump 271, 280, 281
amoverview 285
amplot 286
 sample output 288
amrecover 285
amstatus 280
Andrew File System (AFS) 53
architecture
 disk 14
 I/O 11
 memory 10
 network 13
 processor 8
 virtual server 45
ASP 3
ATM 76
automatic tape loader (ATL) 278
automount 51
availability monitoring 322

B

backup
 in service 129
 loss of an entire server 144
 of complex applications 129
 software 135
 types 128
backup and restore 127
 using Amanda 269
benchmarks 22
billing 298
Border Gateway Protocol (BGP) 91
BOS 365
bussserver 366
Busy Count 27

bzip 270

C

cache

 L1 and L2 10

Cache Manager 375

Capacity BackUp (CBU) 140

capacity planning 20

Capacity Upgrade on Demand (CUoD) 10

CCWs 167

CellServDB 373, 384

Central Processor (CP) 9

Central Processor Complex (CPC) 11

cfgmaker 317

channel 12

Channel Data Link Control (CDLC) 76

Channel-to-Channel (CTC) 74

charge back 298

check-router-alive 330

chroot 216, 217, 226

Cisco CIP 68, 76

cloning images

 copying disks for 214

 creating a patch file 223

 quick start disk 212

 sharing code 221

cloning Linux images 209

CMS pipeline 214

CMS Pipelines 417

CMS RESERVE format 47

cmsfs 417

Coda file system 53

Common Internet File System (CIFS) 49

Common Link Access to Workstation (CLAW) 64,
76

Compatibility I/O Cage 13

connection balancing 79

control unit, DASD

 loss of 142

COPYDISK 225

Count-Key-Data (CKD) 47

cpint 437

cpint.spec 438

cron 284

Cryptographic Element (CE) 11

cryptography 120

CTC

 device restrictions 102

 possible enhancement 408

Cycle Count 27

D

DASD

 formatting options for performance 161

DASD Dump and Restore (DDR) 138, 213

dasdfmt 161

DCSS 53

DCSS-mapped block devices 410

DEFSYS 233

DeMilitarized Zone (DMZ) 84

devfs 448

device file system 447

 installing on SuSE 449

DHCP 221

Diagnose I/O 163

 benefits 164

diff 216

DIRECTXA command 202

DIRM ADD command 204

DirMaint 162, 202, 415, 418

 DVHDXP exit 206

 EXTENT CONTROL 205

disaster recovery 128, 140

discontiguous saved segment (DCSS) 411

disk architecture 14

DISKACNT 300

disklist 273, 276, 278

Distributed Converter Assemblies (DCAs) 11

DMZ (DeMilitarized Zone) 267

DNAT 88, 89, 263

DNS 191

DNS manipulation 77

double-paging 57

DSPBUF 157

dummy interface 79, 409

DVM PROFILE 302

Dynamic Address Translation 410

Dynamic DNS 78

Dynamic Domain Name Service (DDNS) 221

dynamic OSA Address Table 74

dynamic routing 91

E

EAL5 108

Enhanced Interior Gateway Routing Protocol (EIR-
GP) 91

Enterprise Storage Server (ESS) 14, 15, 130, 142
ESALPS 150, 151
ESCON 13, 14
ESCON-16 channel card 12
expanded storage 59
ext2 138
exterior gateway protocols 92
External Time Reference/Oscillator (ETR/OSC) 11

F

FCON/ESA 150
fdasd 47
FDDI 76
Ferguson, Neale 437
FICON 13, 14, 142
FICON channel card 12
find 215
firewall 247, 249
Fixed Block Address (FBA) 47
FlashCopy 130
floor space 3
FORMAT, CMS command 161
forward chain 88
fragmentation 32
Freeside 298
fsck 213

G

gawk 218
GD library 324
Geographical Dispersed Parallel Sysplex (GDPS)
140
GFS 49, 51, 138
 status on zSeries 52
gnuplot 287
GPL 323
groupadd 324

H

hcp 201, 227, 416
 using 437
headroom 29, 31
high availability 139
HiperSockets 64
htpasswd 328
HYPERchannel 76

I

I/O architecture 11
IEEE floating-point 5
IMAP4 359
inetd 117
input chain 87
Integrated Coupling Facility (ICF) processor 9
Integrated Facility for Linux (IFL) 9
interior gateway protocols 92
Inter-User Communications Vehicle (IUCV) 64, 75,
116
IOCDs 106, 109
IOCP 106
ioremap() 412
IP Assist 74
IP forwarding 251
IP Tables 87
ip_nat_ftp 265
IPL command
 PARM option 231
IPTables 244
iptables 267, 268
ISO images
 mounting 211
ISP 3
ITSEC E4 107
IUCV
 device restrictions 102
 possible enhancement 408

J

J2EE 339
Java 339
Java Virtual Machine (JVM) 340
jiffies 55, 61, 185, 446
Just-In-Time compilation (JIT) 340

K

kas 368
kaserver 365, 366
Kerberos 367
klog 373
klogd 208

L

LDAP 118
lilo 206

- Linux
 - as a virtual router 68
 - cloning 209
 - modes it can be run in 5
 - RAMdisk for memory 59
 - running as a VM guest 7
 - running in LPAR mode 6
 - running in native mode 5
 - security 116
 - sharing memory 63
 - swap space 58
 - swap space in memory 59
 - using the 2.4 kernel 441
 - using the OSA with 95
 - Linux Community Development System 126, 421
 - components 422
 - lessons learned 431
 - monitoring 430
 - summary 435
 - team 427
 - technical implementation 424
 - Linux Documentation Project 268
 - Linux Standard Base (LSB) 222
 - locking
 - in shared file systems 55
- M**
- maintenance 18
 - major number 134
 - Maryland
 - University of 270
 - masquerading 89, 264
 - MDC 166
 - mdisk driver 163
 - measurement tools 150
 - memory
 - in Linux images 56
 - sharing 63
 - sizing 36
 - memory architecture 10
 - memory queues
 - with OSA-Express 14
 - Microsoft servers
 - replacing 345
 - migration planning 123
 - Minidisk Cache (MDC) 411
 - minidisk cache (MDC) 53, 214
 - mirroring 137, 144
 - mke2fs 214
 - mkkickstart 210
 - mkswap 412
 - mmap() 410
 - modes
 - as a VM guest 7
 - in an LPAR 6
 - native 5
 - modload 364
 - MPCOSA 76
 - MRTG 316
 - bandwidth reporting 318
 - mrtg_total 319
 - Multi Path Channel (MPC) 76
 - Multi-Chip Module (MCM) 8
- N**
- Named Saved System (NSS) 228
 - Named Shared Storage (NSS) 63
 - Net Saint
 - host groups 330
 - NetSaint 322
 - 3D status map 336
 - building 325
 - command configuration 331
 - configuring 325, 329
 - download 323
 - screen shot 329
 - service configuration 331
 - starting and stopping 333
 - summary screen 334
 - user authorization 327
 - using 333
 - netsaint.cfg 329
 - NETSNMP 153, 179
 - Network Address Translation (NAT) 88, 263, 426
 - network architecture 13
 - network block device 137
 - network design
 - resilient 66
 - network performance 172
 - Network Time Protocol Daemon (NTPD) 371
 - networking
 - devices 74
 - NFS 50
 - NOMDCFS 183
 - nscgi.cfg 329
 - nslookup 227

NSS 415

O

OBEY command 68
OBEYFILE 68, 192, 193, 199
Open Shortest Path First (OSPF) 91
Open Systems Adapter (OSA) 64, 74
OpenAFS
 installing 363
OpenLDAP 118
OS/390 101
OSA
 sharing 102
 using multiple 100
OSA Address Table (OAT) 74, 96
OSA/SF 82
OSA-2 13
OSA-Express 14
 features 74
 QDIO mode 101
OSA-Express (OSA-E) 12
output chain 88

P

P/390 6
packet filtering 81, 243
 planning 82
 what it is 81
Parallel Sysplex 140
parts of the book
 practical 187
 theoretical 1
passwords
 synchronizing NT and Linux 349
penguins
 breeding 223
performance analysis 147
 alerts 175
 DASD subsystem 161
 DASD write analysis 170
 data sources 153
 global controls 157
 global vs. local tuning 154
 measurement tools 150
 measuring Linux applications 180
 network performance 172
 server resources 174
 storage subsystem 160

 tuning guidelines 182
 why measure 148
PGT004 abend 190
pipeline 214
Plato 3
Pluggable Authentication Module (PAM) 118, 349
Point-to-Point Remote Copy (PPRC) 131
POP3 359
port forwarding 267
PPRC 137
practical discussions 187
private IP addresses 88
processor
 bandwidth 9
 cache 10
processor architecture 8
Processor Resource/System Manager (PR/SM) 6
processors
 configuration rules 9
PROFILE EXEC 220
PROFILE TCPIP 194, 198
pts 369
ptserver 366
putty 428

Q

qdio driver 446
QDIO mode
 OSA-Express
 QDIO mode 101
Queued Direct I/O (QDIO) 14
Queued Direct IO (QDIO) 74
QUICKDSP 158

R

R/390 6
RAID 135
RAMAC Virtual Array (RVA) 163
rcapache 326
RECORDING 300
Redbooks Web site 453
 Contact us xviii
relative capacity 21
RESERVE, CMS command 161
resilient IP addressing 77
round-robin DNS 77
routing
 using VM TCP/IP 192

- routing domain 92
- Routing Information Protocol (RIP) 91
- routing protocols
 - choosing 94
- RPC 50
- rpm 439
- RTM 150
- runntp 371

S

- S/390
 - architecture overview 4
- Samba
 - building 350
 - setting up as a PDC 346
 - using as a domain controller 345
- samedit 352
- SAVESYS 233, 412
- scanlog 120
- scp 117
- secure shell 117
- security 105
 - cryptography 120
 - in LPAR mode 106
 - in native mode 106
 - Linux 116
 - PR/SM certification 107
 - System Integrity Statement for z/VM 111
 - under VM 110
 - viruses 119
 - with channels 109
 - with memory 109
- sed 216
- Self-Timed Interconnect (STI) 74
- Self-Timed Interfaces (STIs) 12
- Serial Line Interface Protocol (SLIP) 74
- server farm 3
- Service Level Agreement 298
- service level agreements 177
- setacl 379
- setcellname 366
- setdate 294
- shadow passwords 118
- shadowed disk 413
- Shark 14
- SID 348
- silo 212, 229, 442
- sizing 19

- example 33
- memory 36
- skew 33
- skills required 17
- slapd 118
- slogin 117
- slurpd 118
- SMB 49
- SNALINK 76
- SNAT 88, 263
- SNMP 153, 178, 316
- snmpwalk 317
- SOURCEVIPA 81, 409
- SPECcpu 23
- spikiness 30
- SSH 428
- SSL transactions 121
- Standard Performance Evaluation Corporation (SPEC) 22
- subpools 51
- SuSEconfig 215, 251
- swap space 184
- sync 130, 213
- System Assist Processor (SAP) 9, 12
- SYSTEM DTCPARMS 201
- system maintenance 18
- system monitoring 297

T

- tape changer 277
- tape driver
 - usage 133
- tape support 133
 - major number 134
- tape390 133
- tcpd 117
- team
 - photo xvi
- team that wrote this book xiv
- Techline 26
- TechXpress form on Web 26
- thanks to xvi
- theoretical discussions 1
- Time-To-Live (TTL) 78
- Tivoli Storage Manager (TSM) 139
- tokens 378
- Tomcat 339
- Total Cost of Ownership (TCO) 39, 125

- components 40
- trade-offs 42
- TRACE 229
- Transaction Processing Performance Council (TPC) 22
- trust account 348
- tuning guidelines 182

U

- UCD-SNMP 317
- UID 348
- umount 213
- University of Maryland 270
- upclientbin 371
- upclientetc 370
- updclone.sh 227
- upserver 370
- USER DIRECT file 202
- utilization 26
- UTS Global 76

V

- VDISK 58, 61, 63, 115, 212
- Velocity Software 150
- Virtual Channel-To-Channel (VCTC) 64, 116
- Virtual IP Addressing (VIPA) 79
- virtual networking 90
- Virtual Private Networks (VPN) 119
- virtual processors
 - under VM 113
- virtual router 68
 - using VM 68
- virtual server architecture 45
 - disk topology 46
 - memory topology 55
 - network topology 64
- viruses 119
- vlserver 366
- VM
 - ACCOUNT command 303
 - accounting 300
 - accounting records 302
 - Diagnose I/O 163
 - TCP/IP restriction 104
- VM configuration 189
- VM Diagnose I/O 47
- VMPAF 150
- VMPRF 151

W

- WebSphere Application Server 341
 - planning 342
- wget 129
- whitespace - unused capacity 27
- workload tuning 70
- written manuals
 - philosophy 3

X

- XAUTOLOG 416
- xautolog 302
- xinetd 117

Y

- YaST 215, 222, 223

Z

- z/OS 342
- z/VM
 - advantages with Linux 16
- zipl 443
- zSeries
 - architecture overview 4
 - models 15



Redbooks

Linux for IBM [®]serverSeries and S/390: ISP/ASP Solutions

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Linux for IBM [®]@server zSeries and S/390: ISP/ASP Solutions



**Create and maintain
hundreds of virtual
Linux images**

**Running on the
mainframe**

Managed by z/VM

This IBM Redbook describes how Linux can be combined with z/VM on zSeries and S/390 hardware - the mainframe. This combination of hardware and operating systems enables Internet Service Providers (ISP) and Application Service providers (ASP) to more efficiently provide services. We assume a broad definition of ASP to include production enterprise solutions as simple as file serving.

In a world of discrete servers, when a new resource is required, workload can either be added to an existing server or a new one can be purchased. Often a new server is installed and the *server farm* grows in the enterprise.

S/390 and zSeries hardware, microcode and software allow physical resources to be made virtual among Linux systems. This allows many hundreds of Linux systems to exist on a single server. Running multiple Linux *images* as guests of VM/ESA or z/VM is a smart choice.

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**