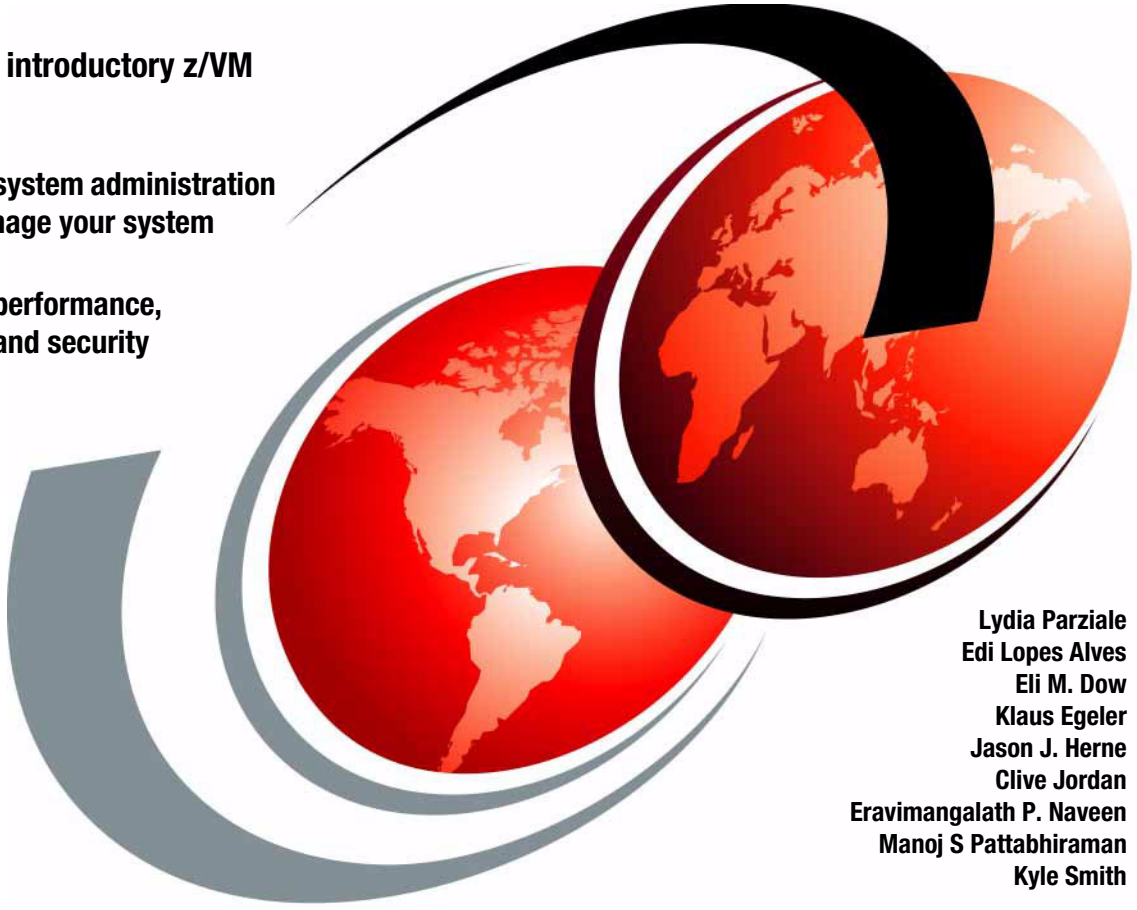
IBM

Introduction to the New Mainframe: z/VM Basics

- Understand introductory z/VM concepts
- Learn basic system administration tasks to manage your system
- Study z/VM performance, networking and security



Lydia Parziale
Edi Lopes Alves
Eli M. Dow
Klaus Egeler
Jason J. Herne
Clive Jordan
Eravimangalath P. Naveen
Manoj S Pattabhiraman
Kyle Smith

Redbooks

ibm.com/redbooks

International Technical Support Organization

Introduction to the New Mainframe: z/VM Basics

November 2007

SG24-7316-00

Note: Before using this information and the product it supports, read the information in “Notices” on page 433.

First Edition (November 2007)

This edition applies to Version 5, Release 3 of z/VM (product number 5741-A05).

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

© Copyright International Business Machines Corporation 2007. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface xiii
How each chapter is organized xiii
The team that wrote this book xiv
Acknowledgements xv
Become a published author xvi
Comments welcome xvii

Chapter 1. Introduction to the mainframe hardware systems 1
1.1 System z hardware architecture 2
1.1.1 Consolidation of mainframes 2
1.1.2 An overview of the early architectures 3
1.1.3 Early system design 5
1.1.4 Current architecture 8
1.2 Hardware Management Console 8
1.3 Frames and cages 9
1.4 Processing units 10
1.4.1 Multiprocessors 10
1.4.2 Processor types 11
1.5 Memory hierarchy 13
1.6 Networking the mainframe 15
1.7 Disk devices 16
1.7.1 Types of DASD 18
1.7.2 Basic shared DASD 19
1.8 I/O connectivity (channels) 20
1.9 System control and partitioning 23
1.9.1 Controlling the mainframe 24
1.9.2 Logically partitioning resources 24
1.10 Exercises 27

Chapter 2. Introduction to virtualization and z/VM 29
2.1 What is virtualization 30
2.2 Benefits of virtualization 31
2.3 How virtualization works 33
2.3.1 Resource sharing 34
2.3.2 Resource aggregation 34
2.3.3 Emulation of function 35
2.3.4 Insulation 36

© Copyright IBM Corp. 2007. All rights reserved.

2.4 Server virtualization 37
2.4.1 Hardware partitioning 37
2.4.2 Hypervisor-based partitioning 39
2.4.3 Hypervisor technologies 41
2.5 Virtualization on the mainframe 44
2.5.1 I/O definition and partition profiles 45
2.5.2 How LPARs are created 46
2.5.3 Additional mainframe virtualization facilities 48
2.6 Virtualization in action 49
2.6.1 Virtualization in a test environment 49
2.6.2 Virtualization to maintain outdated software 50
2.6.3 Improving availability and resilience 51
2.7 Introducing z/VM 52
2.7.1 The virtual machine capability of z/VM 53
2.7.2 Types of operating environments 54
2.7.3 First-level versus second-level guest system 55
2.7.4 z/VM strengths 56
2.8 Exercises 58

Chapter 3. History of z/VM 59
3.1 Life before VM 60
3.2 VM from the beginning 62
3.3 Exercises 66

Chapter 4. z/VM - job roles and basic concepts 67
4.1 Roles in the mainframe world 68
4.1.1 Introduction to roles 68
4.1.2 Role review 73
4.2 Components of z/VM 73
4.2.1 Control Program 74
4.2.2 Conversational Monitor System 75
4.2.3 TCP/IP 76
4.2.4 APPC/VM VTAM Support (AVS) 76
4.2.5 Dump Viewing Facility 77
4.2.6 Group Control System (GCS) 77
4.2.7 HCD and HCM for z/VM 77
4.2.8 Language Environment 78
4.2.9 OSA/SF 78
4.2.10 REXX/VM 79
4.2.11 TSAF 79
4.2.12 VMSES/E 79
4.2.13 DFSMS/VM 80
4.2.14 Directory Maintenance Facility for z/VM 81

4.2.15 Performance Toolkit for VM 82
4.2.16 RACF Security Server for z/VM 82
4.2.17 RSCS Networking for z/VM 84
4.3 VM Directory 85
4.4 How to log on to z/VM 87
4.4.1 Connecting with IBM Personal Communications 87
4.4.2 Connecting with x3270 90
4.4.3 Logging on 94
4.5 Working in a 3270 terminal 97
4.5.1 Keyboard mapping 98
4.6 Session management 99
4.6.1 Logging on 99
4.6.2 Disconnecting 100
4.6.3 Reconnecting 101
4.6.4 Stealing a virtual machine session 101
4.6.5 Logging out 102
4.7 Exercises 102

Chapter 5. Control Program for new users 103
5.1 Introduction to the Control Program (CP) 104
5.1.1 What CP is not 105
5.1.2 CP modes of execution 105
5.1.3 CP commands 106
5.2 Learning about the system 107
5.2.1 Getting to CP mode 107
5.2.2 Examining your virtual machine 108
5.2.3 Other users on the system 113
5.3 Working with a guest operating system 114
5.3.1 Starting a guest operating system 114
5.3.2 Issuing CP commands while running a guest operating system . . . 115
5.3.3 Pausing a guest operating system 116
5.3.4 Resuming a guest operating system 118
5.3.5 Halting a guest operating system 118
5.4 Your virtual machine's virtual devices 119
5.4.1 Querying your virtual devices 120
5.4.2 Processors (CPUs) 122
5.4.3 Storage (main memory) 124
5.4.4 DASD (disk devices) 126
5.4.5 Temporary DASD (TDISK) 131
5.4.6 Virtual DASD (VDISK) 133
5.4.7 Spool devices 135
5.4.8 Communication devices 140

5.5 Terminal management 141
5.5.1 Setting the clear screen timeout 141
5.5.2 Highlighting user input 142
5.5.3 Changing screen colors 143
5.6 z/VM services 144
5.7 Exercises 145

Chapter 6. Conversational Monitor System 147
6.1 CMS introduction 148
6.1.1 Overview 148
6.1.2 Characteristics of CMS 149
6.1.3 About your CMS environment 149
6.2 Getting help from CMS 151
6.2.1 Task menus 151
6.2.2 Component menus 152
6.2.3 Command menus 153
6.2.4 Formatting options 153
6.2.5 Other ways to get help 157
6.2.6 Dealing with error messages 157
6.2.7 Caution when using HELP 158
6.2.8 Exiting the HELP system 158
6.3 Using truncations and abbreviations 158
6.4 Full screen CMS 160
6.5 Examining disks 162
6.5.1 Your disks 162
6.5.2 Linking 163
6.5.3 CMS formatting disks 163
6.5.4 Accessing disks 165
6.5.5 Your A disk 166
6.5.6 Running out of space 167
6.6 Working with files 168
6.6.1 The CMS file system 168
6.6.2 Filename structure 168
6.6.3 Listing 171
6.6.4 CMS search order 173
6.6.5 Searching 174
6.6.6 File management commands 175
6.6.7 CMS Shared File System 178
6.6.8 Concluding file management 181
6.7 Editing files with XEDIT 183
6.7.1 The XEDIT window layout 184
6.7.2 XEDIT and full screen CMS 186
6.7.3 Data manipulation with prefix subcommands 186

6.7.4 Moving through a file 190
6.7.5 Searching within a file 192
6.7.6 Setting tabs 192
6.7.7 Inserting from external files 193
6.7.8 Ending an editing session 195
6.7.9 Customizing xedit 196
6.7.10 Getting help with XEDIT 197
6.8 The PROFILE EXEC 197
6.8.1 PROFILE EXEC capabilities 198
6.8.2 Creating a PROFILE EXEC 198
6.8.3 Synonyms, abbreviations and parsing 200
6.9 Distributing files 201
6.9.1 SEND and RECEIVE 201
6.9.2 LINK and GRANT 202
6.9.3 FTP 203
6.10 Exercises 204

Chapter 7. The REXX programming language 207
7.1 What is REXX 208
7.2 Features of REXX 208
7.3 REXX and VM 209
7.4 REXX overview 210
7.4.1 REXX components 211
7.4.2 General structures and syntax 213
7.5 Creating an EXEC 217
7.6 Executing an EXEC 219
7.7 Stopping an EXEC 219
7.8 Terminal I/O and control structures 219
7.8.1 The ARG statement 221
7.8.2 Parsing data 222
7.9 Conditional branching structures 224
7.9.1 The IF instruction 224
7.9.2 The SELECT instruction 226
7.10 Looping structures 228
7.10.1 Iterative looping 228
7.10.2 Infinite looping 229
7.10.3 Conditional looping 231
7.11 Functions and subroutines 235
7.11.1 Control instructions 235
7.11.2 Functions 238
7.11.3 Program stack 243
7.11.4 Compound variables and stems 248
7.11.5 Host environment commands 250

7.11.6 Detecting and correcting errors . . . 253
7.11.7 EXERCISE . . . 256

Chapter 8. CMS pipelines . . . 259
8.1 Pipeline concepts . . . 260
8.2 Developing pipelines . . . 261
8.2.1 Device driver stages . . . 263
8.2.2 Pipelines in REXX . . . 266
8.2.3 More device drivers . . . 267
8.2.4 Selective filters . . . 268
8.2.5 Multistream pipelines . . . 271
8.2.6 Reference . . . 273
8.3 Exercises . . . 273

Chapter 9. System administration tasks . . . 275
9.1 Overview of system administration tasks . . . 276
9.2 CP commands . . . 276
9.3 CP utilities . . . 279
9.4 CP messages and codes . . . 281
9.5 System configuration . . . 282
9.5.1 CP-owned DASD volumes . . . 283
9.6 PARM disks . . . 284
9.6.1 Accessing the PARM disk . . . 285
9.6.2 Displaying PARM disk content . . . 286
9.7 CPLOAD MODULE . . . 287
9.8 SYSTEM CONFIG file . . . 288
9.8.1 System Config file specifications . . . 288
9.9 LOGO CONFIG . . . 291
9.10 User administration tasks . . . 293
9.11 User directory . . . 293
9.11.1 DISKMAP . . . 293
9.11.2 USER DIRECT control statements . . . 294
9.11.3 Adding guest virtual machines . . . 295
9.11.4 DIRMAINT overview . . . 296
9.11.5 Adding guest virtual machines using DIRMAINT . . . 297
9.12 Managing storage . . . 299
9.12.1 NSS and DCSS . . . 299
9.12.2 Querying NSS . . . 300
9.13 Backing up and restoring data . . . 301
9.13.1 SPXTAPE . . . 303
9.14 Advanced DASD services under z/VM . . . 304
9.14.1 FlashCopy . . . 305
9.14.2 Peer-to-Peer Remote Copy (PPRC) . . . 306

9.14.3 Parallel Access Volumes (PAV) . . . 307
9.14.4 System disk maintenance . . . 308
9.15 Starting z/VM . . . 310
9.15.1 Shutting down z/VM . . . 314
9.16 Basic automation . . . 314
9.17 Advance messaging between users . . . 316
9.18 Installing and servicing the z/VM system . . . 318
9.18.1 Installing . . . 318
9.19 Exercises . . . 320

Chapter 10. Performance . . . 321
10.1 z/VM performance . . . 322
10.1.1 What is performance . . . 322
10.2 Recognizing a performance problem . . . 323
10.3 CP scheduling and dispatching . . . 323
10.4 Performance monitoring . . . 327
10.4.1 CP commands . . . 327
10.4.2 Monitor data collection . . . 334
10.4.3 NSS and DCSS . . . 336
10.5 Performance Toolkit . . . 337
10.5.1 Modes of operations . . . 338
10.6 Tivoli Omegamon for z/VM and Linux . . . 339
10.6.1 Performance monitoring . . . 340
10.6.2 Tivoli OMEGAMON workspaces . . . 340
10.7 Analyzing your data . . . 342
10.7.1 Reactive analysis . . . 344
10.7.2 Predictive analysis . . . 347
10.7.3 Tuning guidelines . . . 349
10.7.4 Other references . . . 350
10.8 Exercises . . . 351

Chapter 11. Networking and connectivity . . . 353
11.1 Introduction to networking in z/VM . . . 354
11.1.1 I/O channel requirements . . . 354
11.2 Supported network devices . . . 355
11.2.1 Open Systems Adapter . . . 355
11.2.2 HiperSockets . . . 356
11.2.3 Channel-to-channel connection . . . 356
11.3 Virtual network types supported by z/VM . . . 356
11.3.1 Inter-User Communications Vehicle (IUCV) . . . 357
11.3.2 Guest LAN . . . 357
11.3.3 Virtual switch . . . 358

11.4 Defining a VSWITCH 360
11.4.1 Enabling VSWITCH failover 361
11.5 Connecting guests to the network 362
11.5.1 Dedicating OSA devices 362
11.5.2 Coupling to a VSWITCH or guest LAN 364
11.6 TCP/IP commands provided by z/VM 366
11.6.1 NETSTAT 366
11.6.2 TRACERTE 372
11.6.3 PING 372
11.7 The z/VM network service model 373
11.8 Exercises 374

Chapter 12. z/VM security 375
12.1 Introduction to z/VM security 376
12.2 External security managers 376
12.3 Directory management 376
12.4 User authentication and authorization 377
12.4.1 Privilege classes 378
12.5 z/VM security features 380
12.5.1 Processor and memory protection 380
12.5.2 Disk protection 380
12.5.3 Tape security 382
12.6 Available cryptographic facilities 383
12.7 Best practices 384
12.8 Exercises 385

Chapter 13. Guest operating systems 387
13.1 Guest support 388
13.1.1 Guest simulation 388
13.2 Supported guest operating systems 389
13.2.1 Linux as a guest operating system 389
13.2.2 z/OS as a guest operating system 390
13.2.3 z/VSE as a guest operating system 391
13.2.4 z/VM as a guest operating system 392

13.3 Exercises 393

Appendix A. Enhancements in z/VM Version 5, Release 3 395
Enhanced scalability and constraint relief 396
Support for up to 256 GB of real memory 396
Up to 32 real processors in a single z/VM image 396
Enhanced memory management for Linux guests 396
Enhanced memory utilization using VMRM between z/VM and Linux guests 397
HyperPAV support for IBM System Storage DS8000 398
Enhanced FlashCopy support 398
Support for the IBM System Storage SAN Volume Controller 399
IBM System Storage SAN Volume Controller Storage Engine 2145 399
IBM System Storage SAN Volume Controller Software V4.1 400
z/VM support for the 2145 SAN Volume Controller 401
Virtualization technology and Linux enablement 401
Support for IBM System z specialty engines (processors) 401
Enhanced virtual switch and guest LAN usability 403
MIDAWs for guests 403
Guest ASCII console support 404
Enhanced SCSI support 404
Network virtualization 405
Improved virtual network management 405
Enhanced failover support for IPv4 and IPv6 devices 405
VIPA support for IPv6 406
Support for OSA-Express2 IEEE 802.3 and link aggregation 406
Security 406
Delivery of LDAP server and client 406
Enhanced system security with longer passwords 407
Conformance with industry standards 407
SSL server enhancements 408
Tape data protection with support for encryption 408
Systems management 409
Enhanced management functions for Linux and other virtual 409
New function level for DirMaint 410
Enhancements to the Performance Toolkit 411
Enhanced guest configuration 411
z/VM Integrated Systems Management 411
Installation, service, and packaging changes 412
Additional DVD installation options 412
Enhanced status information 413
RSCS repackaged as an optional feature 413
New RACF Security Server for z/VM 413

U.S. daylight saving time effect on z/VM 414
z/Architecture CMS shipped as a sample program 414
Withdrawal of ROUTED and BOOTP servers 415
Additional changes 415
Support for searches across PDF files in the z/VM Library 415

Appendix B. Answer key 417
Chapter 1 Introduction to the mainframe hardware systems 418
Chapter 2 Introduction to virtualization and z/VM 418
Chapter 3 z/VM history 419
Chapter 4 z/VM overview 420
Chapter 5 Control Program for new users 420
Chapter 6 Conversational Monitor System 422
Chapter 7 REXX basics 423
Chapter 8 Pipelines 425
Chapter 9 System administration tasks 427
Chapter 10 Performance 428
Chapter 11 Networking and connectivity 428
Chapter 12 Security 430
Chapter 13 Guest operating systems 430

Notices 433
Trademarks 434

Related publications 435
IBM Redbooks publications 435
Other publications 436
CMS 436
Installation and Service 436
Networking and connectivity 436
Performance 437
REXX/VM 437
Security 437
Online resources 437
How to get Redbooks 438
Help from IBM 438

Index 439

Preface

This textbook provides students with the background knowledge and skills necessary to begin using the basic functions and features of z/VM® Version 5, Release 3. It is part of a series of textbooks designed to introduce students to mainframe concepts and help prepare them for a career in large systems computing.

For optimal learning, students are assumed to be literate in personal computing and have some computer science or information systems background. Others who will benefit from this textbook include z/OS® professionals who would like to expand their knowledge of other aspects of the mainframe computing environment. This course can be used as a prerequisite to understanding Linux® on System z™.

After reading this textbook and working through the exercises, the student will have received a basic understanding of the following topics:

- The Series z Hardware concept and the history of the mainframe
- Virtualization technology in general and how it is exploited by z/VM
- Operating systems that can run as guest systems under z/VM
- z/VM components
- The z/VM control program and commands
- The interactive environment under z/VM, CMS and its commands
- z/VM planning and administration
- Implementing the networking capabilities of z/VM
- Tools to monitor the performance of z/VM systems and guest operating systems
- The REXX™ programming language and CMS pipelines
- Security issues when running z/VM

How each chapter is organized

Each chapter follows a common format:

- Objectives for the student
- Topics relevant to the basics of z/VM computing
- Questions or hands-on exercises to help students verify their understanding of the material

© Copyright IBM Corp. 2007. All rights reserved.

The team that wrote this book

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

Lydia Parziale is a Project Leader for the ITSO team in Poughkeepsie, New York with domestic and international experience in technology management including software development, project leadership and strategic planning. Her areas of expertise include e-business development and database management technologies. Lydia is a Certified IT Specialist with an MBA in Technology Management and has been employed by IBM® for 24 years in various technology areas.

Edi Lopes Alves is an IT Systems Management Specialist with IBM Global Services, Brazil. She has more than 20 years of experience as a VM systems programmer and with IBM DB2® Content Manager solutions in the Finance area. She is a Certified z/Series Specialist with a Masters degree in e-business from ESPM, Sao Paulo. Edi currently supports IBM z/VM internal systems and products.

Eli M Dow is a Software Engineer with the Test and Integration Center for Linux in Poughkeepsie, New York, where he performs Linux integration testing. He holds a Bachelors degree in Computer Science and Psychology, as well as a Master of Science degree in Computer Science from Clarkson University. His areas of expertise include numerous virtualization platforms like Xen and z/VM, Linux, as well as systems programming.

Klaus Egeler is an IT Systems Management Specialist with IBM Global Services, Germany. He has more than 15 years of experience as a VSE and VM systems programmer. He has worked with Linux for IBM eServer™ zSeries® and IBM S/390® for more than five years. Klaus has contributed to several z/VM- and Linux-related IBM Redbooks® publications, and has been a presenter at ITSO workshops and customer events.

Jason Herne is a z/VM developer in Endicott, NY. He has five years of experience with z/VM and System z. He has a Masters of Science degree in Computer Science from Clarkson University. His areas of expertise include z/VM and Linux operating systems.

Clive Jordan is a Software Specialist in the UK. He has 34 years of experience in the IBM data processing environment. He apprenticed in electrical engineering at the Cranfield Institute of Technology before joining IBM as a hardware engineer. Clive currently supports customers in the UK and Ireland for z/VM, z/VSE™ and z/OS automation products, and teaches the z/VM curriculum.

Eravimangalath P Naveen is an IT Infrastructure Architect at the IBM India Software Lab in Bangalore, India, with seven years of experience. His areas of expertise include IBM AIX®, Linux, z/VM and System z hardware, IBM Tivoli® Storage Manager, VMWare, Storage Systems, Networking, and Virtualization across all IBM Server systems.

Manoj S Pattabhiraman is a Staff Software Engineer in Linux Technology Center (LTC), India. He holds a Masters degree in Computer Applications from the University of Madras. He has seven years of experience in z/VM Application Development and Systems Programming for Linux on zSeries. His areas of expertise include z/VM, Linux on System z, middleware performance and z/VM REXX programming.

Kyle Smith is a software engineer with IBM in Poughkeepsie, New York, where he tests IBM Director and enterprise Linux distributions on IBM System z. He holds a Bachelor of Science degree in Computer Science from Clarkson University. Kyle's areas of expertise include Linux and Java™ Application Development.

Figure 1 From left: Kyle Smith, Clive Jordan, Jason Herne, Manoj S. Pattabhiraman, Eli Dow, Eravimangalath P Naveen, Klaus Egeler, Edi Lopes Alves

Acknowledgements

The following people are gratefully acknowledged for their contributions to this project:

Roy P. Costa
International Technical Support Organization, Poughkeepsie Center

Frank Le Blanc
Boston University

Melinda Varian
Princeton University

Jeff Gribbin
EDS

Scott P. Drummond, Mac Holloway, Arunkumaar Ramachandran, Raymond J. Sun™
IBM Software Group

Samuel D. Cohen
Global Technology Services, IBM

Tracy Adams, Alan Altmark, Robert J. Brenneman, Daniel Fitzgerald, Igor Hernandez, Yanelis Hernandez, Michael MacIsaac, Reed A. Mullen, Dorin Pascar, Jim Rymarczyk, Donald J. Smith, Romney White, Patrick F. Wilbur, Steve G. Wilkins
Systems and Technology Group, IBM

Svend Erik Bach, Charlie Burger, Mark Cathcart, Jim Elliott, Harvey W. Emery Jr, Christian Matthys, Alan Naylor, Bill Seubert, Julie A. Schuneman
Sales and Distribution, IBM

Special thanks to the authors of the z/OS Basics textbook, *Introduction to the New Mainframe: z/OS Basics*, SG24-6366, published in July, 2006, for supplying some of the content for this book:

Mike Ebbers, Bill Ogden
International Technical Support Organization, Poughkeepsie Center

Wayne O'Brien
IBM Systems & Technology Group

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

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

1

Introduction to the mainframe hardware systems

As a new z/VM user, you will need to develop an understanding of the hardware platform that runs the z/VM operating system. z/VM is designed to make full use of mainframe hardware and its many sophisticated peripheral devices. Though much of the hardware used with z/VM has its roots in older mainframe system designs, this chapter will introduce you to the concepts and current systems in production today.

Objectives

After completing this chapter, you will be able to:

- Discuss System z hardware used with z/VM
- Explain processing units and disk hardware
- Explain how mainframe hardware differs from personal computer systems

© Copyright IBM Corp. 2007. All rights reserved.

1.1 System z hardware architecture

System z, as with all computing systems, is built on hardware components. Most of those components were introduced early in the mainframe era, and were developed over the years. As a user of z/VM, you will often need to interact with the hardware or speak in terms of the terminology prevalent in the mainframe world. This chapter discusses the reasons why things are the way they are now, and also supplies you with the necessary understanding of the hardware to enable you to efficiently handle your day-to-day use of z/VM.

1.1.1 Consolidation of mainframes

There are fewer mainframes in use today than there were 15 or 20 years ago. Why is this? In some cases, all the applications were moved to other types of systems. However, in most cases, the reduced number is due to consolidation; several lower-capacity mainframes have been replaced with fewer higher-capacity systems.

There is a compelling reason for consolidation. Software (from many vendors) can be expensive and typically costs more than hardware. It is usually less expensive (and sometimes *much* less expensive) to replace multiple software licenses (for smaller machines) with one or two licenses (for larger machines). Software license costs are often linked to the power of the system, but the pricing curves favor a small number of large machines.

Software license costs have become a dominant factor in the growth and direction of the mainframe industry. There are several nonlinear factors that make software pricing very difficult. One such factor, the exponential growth of mainframe processing power, has been problematic in recent years.

The power needed to run a traditional mainframe application (a batch job written in COBOL, for example) is unlike the needs of a modern mainframe running z/VM with many guest operating systems (which in turn might be executing complex code with graphical user interfaces, or written in Java). The consolidation effect has produced very powerful mainframes.

Regardless of the speed of your mainframe, you should take the time to learn some of the internal hardware used in mainframes. It is vital that z/VM users and operators have a basic understanding of the hardware they are running on. Some of the hardware is not even physically made anymore but still has relevance to z/VM. The remainder of this chapter will illustrate the basic hardware concepts and whenever possible, directly relate them to the use of z/VM.Evolution of mainframe hardware systems

This chapter provides a simplified overview of mainframe hardware systems, with emphasis on the processor. (For more extensive information about mainframe hardware, there are numerous other resources that you can consult.)

1.1.2 An overview of the early architectures

If you are wondering why an historical overview of early mainframe architecture is important, keep in mind that unlike personal computers—where technology is continuously made obsolete—mainframe computers maintain backward compatibility. This section examines the intricacies of z/VM design and of interacting with mainframes in general.

Related reading:
The publication *Principles of Operation* provides detailed descriptions about the major facilities of z/Architecture®. You can find this and other IBM publications at the z/VM Internet Library Web site:

`http://www.ibm.com/servers/eserver/zseries/zos/bkserv/zvmpdf/zvm53.html`

We start by explaining terminology that is associated with mainframe hardware. Being aware of various meanings of the terms systems, processors, CPs, and so forth is important for your understanding of z/VM and mainframes in general.

In early mainframe computing, starting with the IBM S/360™ (the ancestor of the current mainframes in production today), a system had a single *processor*, which was also known as the *central processing unit* (CPU). This is the same term used to describe the processor in your laptop or desktop personal computer.

In those days, the terms *system*, *processor*, and *CPU* were used interchangeably. Today, systems are available with more than one processor, and the terminology used to refer to such components has evolved. All current mainframes have more than one processor, and some of those processors are specialized for specific purposes (for example, I/O). Some of the terminology that is used today is illustrated in Figure 1-1 on page 4.

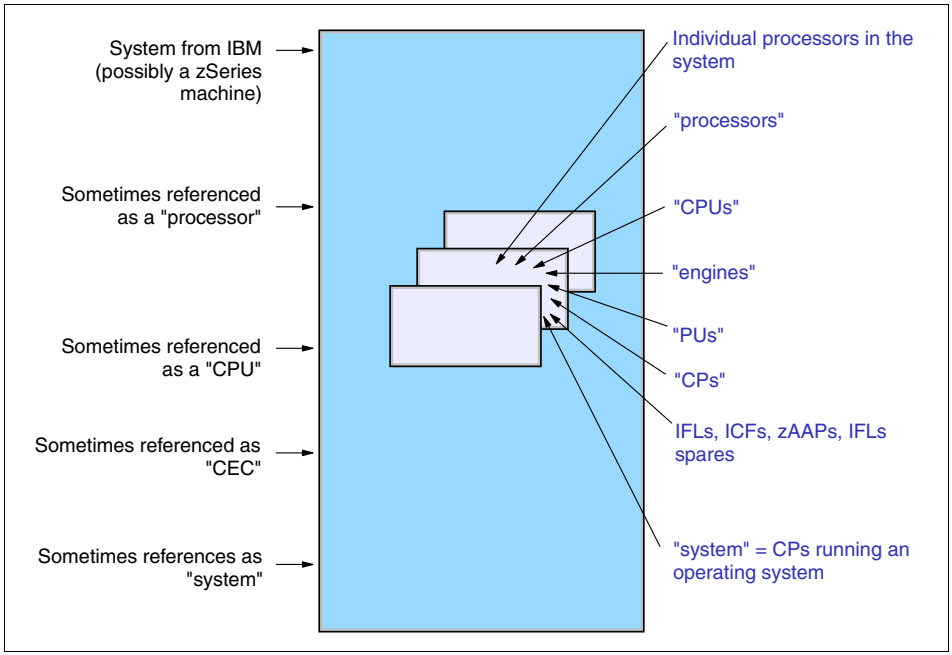


Figure 1-1 Terminology overlap

System programmers use the IBM terms *Central Processor Complex* (CPC) or *Central Electronics Complex* (CEC) to refer to the physical mainframe.

In this text, we use the term CPC and CEC interchangeably to refer to the physical collection of hardware that includes main storage (memory), one or more central processors, timers, channels, and the numerous other hardware components that could be contained inside a single mainframe chassis.

Although the terms processor and CPU can refer to either the complete system, or to one of the processors (CPUs) within the system, we recommend that you use the term CPU to refer to an actual processor unit inside the system, and the term CPC or CEC to discuss the entire physical machine.

Note: Although the meaning may be clear from the context of a discussion, even mainframe professionals must clarify which processor or CPU meaning they are using in a discussion. Adhering to the conventions suggested here will help to avoid confusion.

1.1.3 Early system design

The central processor contains the processors, memory, control circuits, and interfaces for channels. A *channel* provides a path between I/O devices and memory. Early systems had up to 16 channels. In contrast, modern mainframes can have many channels.

Channels connect to control units. A *control unit* contains the logic to work with a particular type of I/O device. A control unit for a printer, for example, has much different internal circuitry and logic than a control unit for a tape drive. Some control units can have multiple channel connections providing multiple *paths* to the control unit and its devices.

Control units connect to *devices* such as disk drives, tape drives, communication interfaces, and so forth. The division of circuitry and logic between a control unit and its devices is not defined, but it is usually more economical to place most of the circuitry in the control unit.

Figure 1-2 on page 6 presents a conceptual diagram of a mainframe system. Note that current systems are not connected as shown in Figure 1-2 on page 6. However, this diagram helps to explain the background terminology that permeates mainframe discussions.

Modern mainframes are “descended” from the architecture presented in this diagram, and retain some of the design shown. Because modern mainframes retain backward compatibility with their predecessors, this early design warrants further examination.

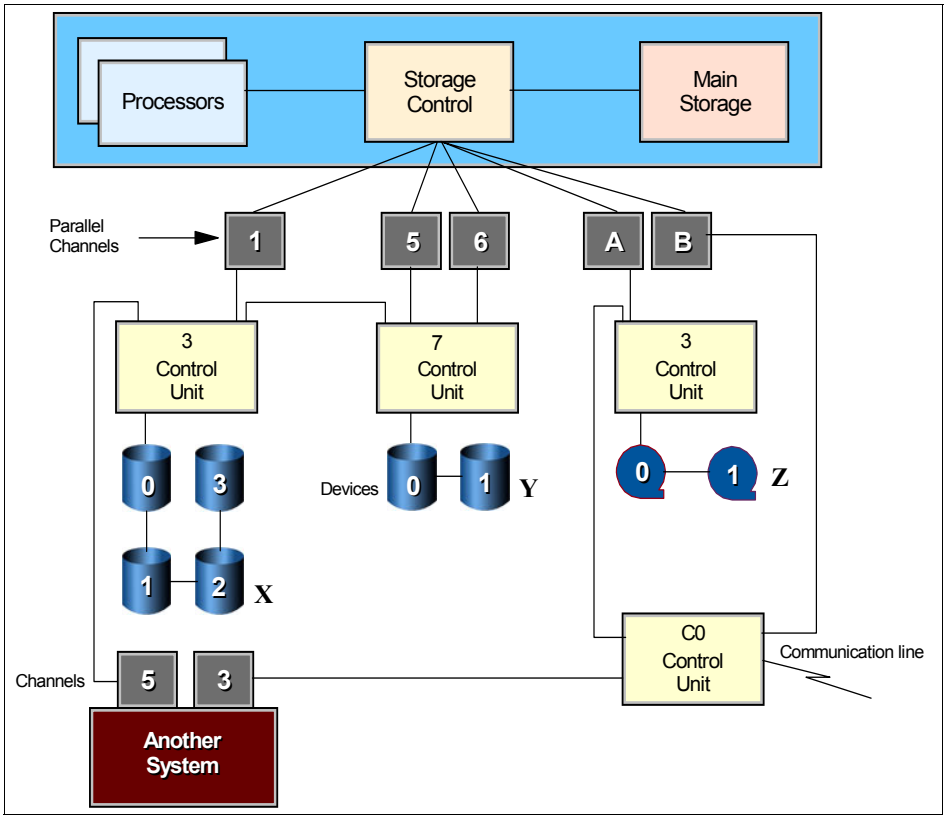


Figure 1-2 Conceptual mainframe

The Storage Control block shown in Figure 1-2 indicates the logic controlling the hard disk and tape-related operations. Notice that control units can be connected to multiple devices. The maximum number of devices depends on the particular control unit.

Each channel, control unit, and device has an address, expressed as a hexadecimal number. The disk drive marked with an X in Figure 1-2 has address 132, which is derived as shown in Figure 1-3.

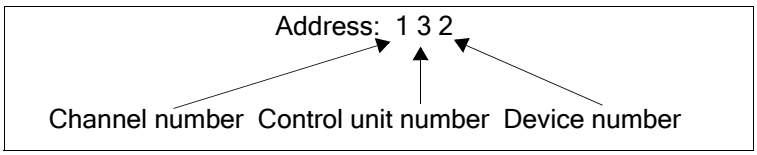


Figure 1-3 Device address

The disk drive marked with a Y in Figure 1-2 on page 6 can be addressed as 171, 571, or 671 because it is connected through three channels. By convention, the device is known by its lowest address (171), but all three addresses could be used by the operating system to access the disk drive.

Multiple paths to a device are useful for performance and for availability. In the conceptual system represented in Figure 1-2 on page 6, when an application wants to access disk 171, the system will first try channel 1. If it is busy (or not available), it will try channel 5, and so forth.

Figure 1-2 on page 6 also contains another S/360 system with two channels connected to control units used by the first system. This sharing of I/O devices is common in all mainframe installations. Tape drive Z is address A31 for the first system, but is address 331 for the second system.

Sharing devices, especially disk drives, is not a simple topic and there are hardware and software techniques used by the operating system to control exposures such as updating the same disk data at the same time from two independent systems.

Note: For more information about the development of IBM mainframes since 1964, refer to the following Web site:

http://www-03.ibm.com/history/exhibits/mainframe/mainframe_basinfo.html

As mentioned, current mainframes are not used exactly as shown in Figure 1-2 on page 6. Differences include the following areas:

- The parallel channels represented in Figure 1-2 on page 6 are no longer available on the newest mainframes and are slowly being displaced on older systems. They have been replaced with newer, more efficient types of channels called Enterprise Systems CONnection (ESCON®) and FIber CONnection (FICON®) channels. We examine each of these technologies in later sections of this chapter.
- Current mainframes have more than 16 channels and use two hexadecimal digits as the channel portion of an address.
  - Channels are generally known as *channel path identifiers (*CHPIDs) or *physical channel identifiers (*PCHIDs) on later systems, although the term *channel* is also correct. The channels are all integrated in the main processor box. Note the following points:
  - A CHPID is a value assigned to each channel path of the system that uniquely identifies that path.
  - A PCHID reflects the physical location of a channel-type interface. PCHID number is based on the device location and the port number.

The device address seen by software is more correctly known as a *device number* (although the term *address* is still widely used) and is indirectly related to the control unit and device addresses.

1.1.4 Current architecture

Current CPC designs are considerably more complex than the early S/360 design. This complexity includes many areas, such as:

- I/O connectivity and configuration

  Note: For more information about mainframe I/O connectivity and configuration, refer to *System z Connectivity Handbook*, SG24-5444.

- I/O operation
- Partitioning of the system

1.2 Hardware Management Console

As a user of z/VM, you may be called upon to alter the hardware configuration of your system, or to answer questions from support personnel regarding the system configuration.

Regardless of what hardware is in your mainframe, the hardware can be managed by using either the Support Elements (SE) directly attached to the server, or the Hardware Management Console (HMC). The HMC is a desktop application providing the end-user interface to control and monitor the status of the system.

Working from a Hardware Management Console, an operator, system programmer, or IBM technical personnel can perform basic operations on System z servers. Some of the common capabilities of the HMC are:

- Load the System z hardware configuration.
- Load or reset a system image.
- Add and change the hardware configuration (most of them dynamically).
- Access the hardware logs.

All of these functions can be executed by using a Web interface in a secure environment. If you are reading this book, probably you have not yet used a Hardware Management Console, but as a concept it is important to understand because the HMC is the centralized location from which hardware management

for the entire mainframe can be performed. Changes made on the HMC may alter only your z/VM system, or the entire mainframe.

You may never need to use the HMC if you rely on system operators or administrators, but if you spend enough time using z/VM, you may need it.

1.3 Frames and cages

The current System z hardware layout is the result of the continuous evolution of the mainframe hardware architecture. Over the years new components have been added to the design but the basic requirement had not been changed. The current design is highly available and has redundancy for most of the parts.

The z9™ is built on *frames*, to which the various components are fixed. Depending on the model number that is ordered, the frames may not be fully populated. Each frame can contain several *cages*. There are two types of cages:

- CEC cage

  The CEC cage is the basic cage. It contains the processor units (PU), the physical memory, and connectors to the other cages. On each System z machine there is only one CEC cage.

- I/O cage

  The I/O cage contains the hardware necessary for interacting with System z external I/O devices. Each System z configuration has at least one I/O cage, with a maximum of three I/O cages.

Memory and processors reside in what are termed *books*. The book concept was first introduced on the z990 machine. A book contains processors, memory, and connection to the I/O cages. Books are located in the CEC cage. Each System z configuration has one to four books.

Note: The current maximum hardware configuration allows 54 processor units to be available to the operating systems. Physically there are up to 64 PUs installed on the machine. The 10 additional PUs are used for I/O workloads and as hot spares in case another PU malfunctions.

Figure 1-4 on page 10 shows a z9 processor with the covers removed.

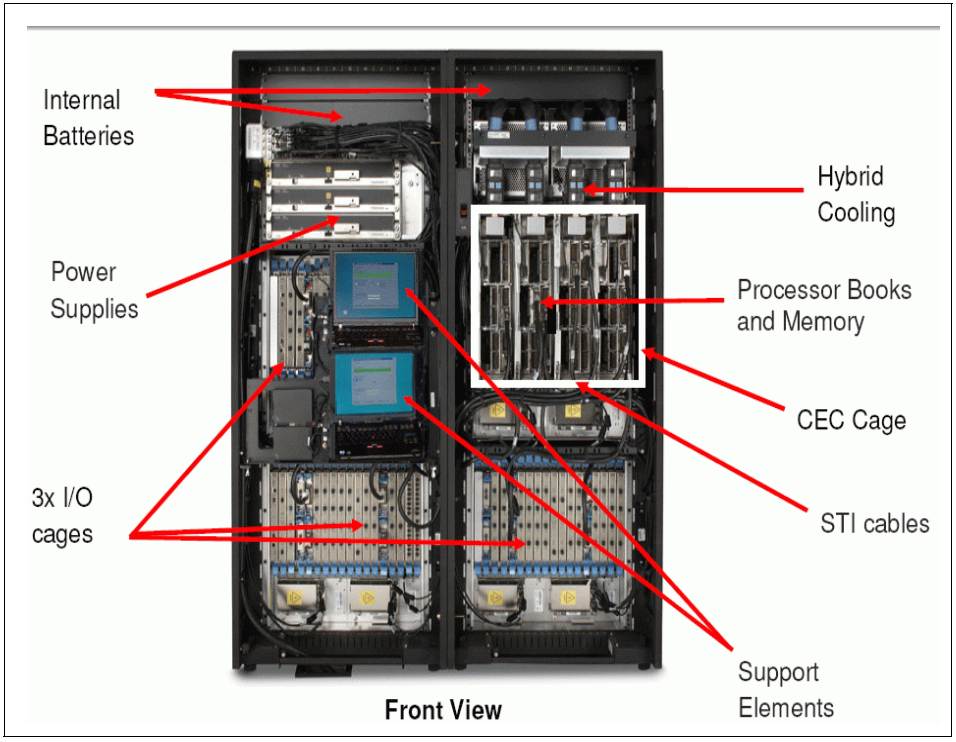


Figure 1-4 IBM System z9™ mainframe

1.4 Processing units

This section gives an overview of the different types of processors used on the mainframe.

1.4.1 Multiprocessors

Though it is possible to purchase a current mainframe with a single processor (CP), it would not be a typical system.[1] The term *multiprocessor* means several processors (CP processors), and it implies that several processors are used by an instance of z/VM.

[1] All current IBM mainframes also require at least one System Assistance Processor (SAP®), so the minimum system has two processors: one CP and one SAP. However, the use of the term "processor" in the text usually means a CP processor usable for applications. Whenever we discuss a processor other than a CP, we always make this clear.

The earliest operating systems were used to sequentially control single-user computer systems. In contrast, current computer systems are capable of *multiprogramming* (which means executing many programs concurrently). When a job cannot use the processor, perhaps because it needs to wait for an asynchronous I/O operation to complete, multiprogramming enables the system to suspend the job, thus freeing the processor to work on another job. When the I/O operation completes, the currently executing piece of work is interrupted and the suspended job is scheduled to run.

Most modern operating systems today, from mainframes to personal computers, can function in a multiprocessor environment. However, the degree of integration of the multiple processors varies considerably. With the mainframe, for example, pending interrupts in a system can be accepted by any processor in the system. Any processor can initiate and manage I/O operations to any channel or device available to the system. Channels, I/O devices, interrupts, and memory are owned by the system and not by any specific processor.

This multiprocessor integration appears simple on the surface, but its implementation is complex. It is also important for maximum performance; the ability of any processor to accept any interrupt sent to the system is especially important.

Operating systems make multiprogramming possible by capturing and saving status information about the interrupted program before allowing another program to execute. When the interrupted program is ready to begin executing again, it can resume execution just where it left off. Multiprogramming enables the operating system to run thousands of programs simultaneously.

When a computer system has multiple processors, the operating system supports multiprocessing, where each of the multiple processors can simultaneously process separate instruction streams from the various programs. The multiple processors share the various hardware resources, such as memory and external disk storage devices. Multiprocessing enables multiple programs to be executed simultaneously, and allows a single program to use multiple processes in the same program to do parallel work.

The System z environment has supported multiprogramming and multiprocessing for its users for many years.

1.4.2 Processor types

In any given mainframe there could be multiple processors, with each one processing different kinds of software. Figure 1-1 on page 4 lists several different classifications of processors tailored to various kinds of work. Although each processor is of the same System z architecture, they are to be used for slightly

different purposes.[2] Several of these purposes are related to software cost control, while others are more fundamental.

All these processors start as equivalent processor units[3] (PUs) or engines. A PU is a processor that has not been characterized for use. *Characterized* in this context means restricting the type of code that can be executed on a given processor.

Each of the processors begins as a general purpose processor and is characterized by the manufacturer during installation or at some later time. The potential characterizations are:

- Central Processor (CP)

  This is a processor available to normal operating system and application software.

- System Assistance Processor (SAP)

  Every modern mainframe has at least one SAP; larger systems may have several. The SAPs execute internal code[4] to provide the I/O subsystem.

  An SAP, for example, translates device numbers and real addresses of CHPIDs, control unit addresses, and device numbers. It manages multiple paths to control units and performs error recovery for temporary errors. Operating systems and applications cannot detect SAPs, and SAPs do not use any “normal” memory. See 1.8, “I/O connectivity (channels)” on page 20 for more information about SAPs.

- Integrated Facility for Linux (IFL)

  An IFL is almost exactly the same as a normal central processor. The only difference is that the IFL lacks two instructions that the CP has, and which are used only by z/OS. Linux and z/VM do not use these instructions.

  The difference in using an IFL with Linux and z/VM from z/OS is that an IFL is not counted when specifying the model number[5] of the system and thus does not contribute to the “performance” of the machine when it comes time to license certain software packages. This can make a substantial difference in software costs; thus, many users opt for IFLs to run z/VM and Linux.

---

[2] Do not confuse these with the controller microprocessors. The processors discussed in this section are full, standard mainframe processors.

[3] This discussion applies to the current System z and zSeries machines at the time of writing. Earlier systems had fewer processor characterizations, and even earlier systems did not use these techniques.

[4] IBM refers to this as Licensed Internal Code (LIC). It is often known as microcode (which is not technically correct) or as firmware. It is definitely not user code.

[5] Some systems do not have different models; in this case a *capacity model number* is used.

- Spare

  An uncharacterized PU functions as a *spare*. If the system controllers detect a failing CP or SAP, it can be replaced with a spare PU. In most cases this can be done without any system interruption, even for the application running on the failing processor.

- Various forms of *Capacity on Demand* (CuOD) and similar arrangements exist whereby a customer can enable additional CPs at certain times (for unexpected peak loads, for example).

- Integrated Coupling Facility (ICF)

  An ICF runs special code that is used to "couple" together multiple z/OS systems into a cooperative environment.

Note: Other important mainframe processors exist, but they are of less relevance to z/VM users. For example, the System z9 Integrated Information Processor (zIIP) is a specialized engine for processing eligible database workloads. The System z Application Assist Processor (zAAP) is a processor with a number of functions disabled (interrupt handling, some instructions) such that no full operating system can be executed on the processor. However, z/OS can detect the presence of zAAP processors and will use them to execute Java code (and possibly other similar code in the future). These processor types exist only to control software costs.There are also processors specialized for cryptography.

In addition to these characterizations of processors, some mainframes have models or versions that are configured to operate slower than the potential speed of their CPs in order to decrease purchase and operations cost. This is widely known as *throttling* or *capacity setting*. IFLs, SAPs, zAAPs, and ICFs always function at the full speed of the processor, because these processors do not count in software pricing calculations.[6]

1.5 Memory hierarchy

Getting data to the processor quickly and efficiently is a major task for all systems, and all major modern computing architectures have some level of hierarchy used for getting and storing data for execution on a processor. The mainframe is no different. The memory in a mainframe is typically referred to as *storage* (or more technically, *real storage*). The term *memory* is used as the equivalent in personal computer computing. Some people in the mainframe

[6] This is true for IBM software but may not be true for all software vendors.

community refer to hard disk units as “storage” as well, but we recommend that you use the term storage only for the equivalent of personal computer memory.

The hierarchy on the mainframe is illustrated in Figure 1-5.

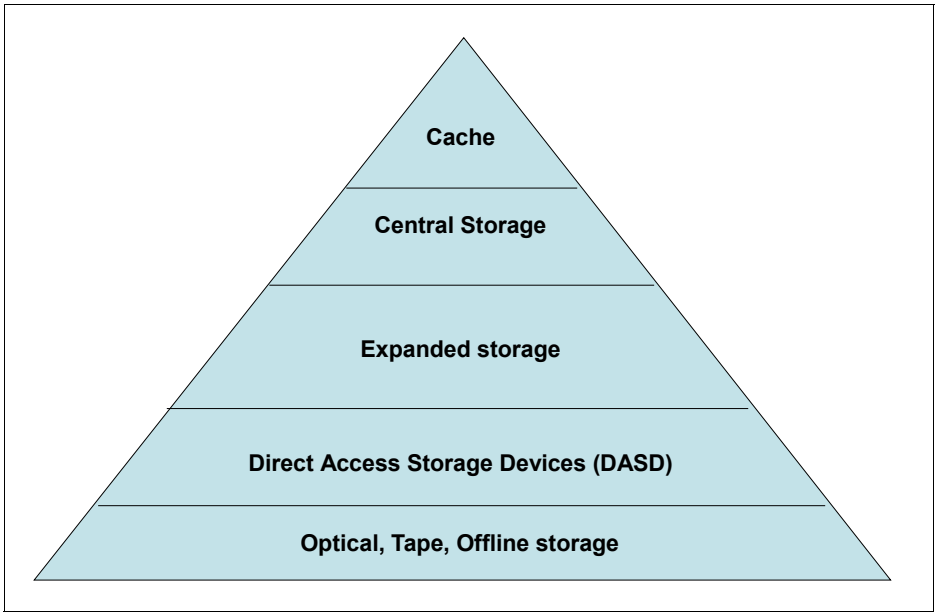


Figure 1-5 Mainframe memory/storage hierarchy

The terms shown are explained here:

- L1/L2 Cache

  The cache stores frequently accessed instructions for faster execution by the processor.

- Central Storage

  Central storage contains the current running operating system and any processes or programs and data being used by the operating system. The amount of central storage supported depends on the addressability of the processor architecture. z9 supports 512 GB today, but the architecture supports up to 16 Exabytes (EB).

  The amount of central storage which can be addressed is constrained by the operating system implementation, as well. For example, S/390 processors represented memory addresses as a 32-bit number but the later generation of zSeries, and the most current iteration known as System z, can address memory as a 64-bit number.

- Expanded storage

  Expanded storage is needed to exploit certain special software facilities and also used as a faster paging device.

  Note: In the past, when memory was expensive, another type of memory was built on top of physical memory, called Expanded storage. This memory was built by using physical memory that was slower and less costly than the physical memory used for the central memory. Expanded storage was addressable only at the page level, so instructions could not be executed from Expanded storage.

  Unlike z/OS, which no longer uses Expanded storage, the z/VM operating system uses expanded memory as a high-speed paging device. Expanded storage is now implemented using the same physical memory as the central memory.

- DASD storage

  With cached control units, DASD are the fast devices external to the processor hardware.

- Peripheral storage

  Peripheral storage is mainly used for long-term persistent storage and is less expensive in comparison to the previous types. Peripheral storage includes Tape, optical storage devices, storage area networks (SAN), and SCSI I/O devices for Linux on System z.

1.6 Networking the mainframe

The various components required to perform networking with the mainframe are explained Chapter 11, “Networking and connectivity” on page 353. Here, we only mention that the interface on the mainframes for networking is known as the Open System Adapter (OSA). This is the LAN adapter for the mainframe. The Integrated Console Controller (ICC) is also configured through the OSA card, which eliminates the need for a separate control unit for the system consoles.

For detailed information about networking a mainframe, refer to *Introduction to the New Mainframe: Networking*, SG24-6772.

1.7 Disk devices

In the mainframe environment, disks are usually referred to as DASD, which stands for Direct Access Storage Device. Although similar in concept to a hard disk in a personal computer or laptop computer, DASD typically comprises many drives in a far more sophisticated arrangement.

Another key difference between mainframe DASD and a personal computer type of hard disk is that the physical disks are external to the mainframe. The device housing the physical disks can be as large as, or larger than, the mainframe.

IBM 3390 disk architecture is commonly used on current mainframes. Originally the 3390 was a hardware model sold with earlier mainframes. Conceptually, the 3390 system is a simple arrangement, as shown in Figure 1-6.

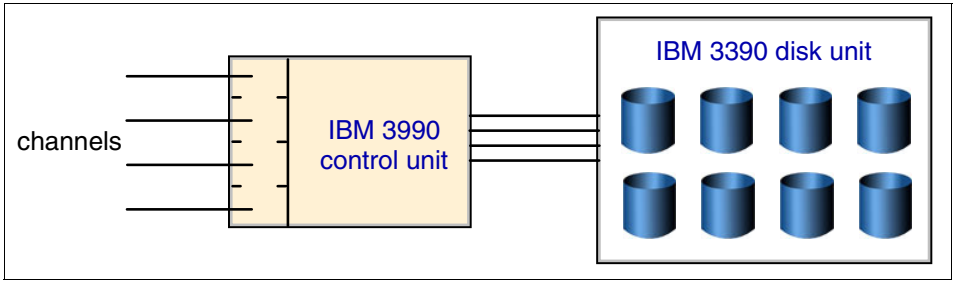


Figure 1-6 Early IBM 3390 disk implementation

This illustration shows 3990 and 3390 units, and it also represents the concept or architecture of current devices. Historically, each 3390 control unit (3990) may have up to eight channels connected to one or more processors, and the 3390 disk unit typically had eight or more disk drives. The concept of the control unit is similar to thinking about the printed circuit board found on the bottom of laptop and personal computer hard disks, although the current control units are far more complex. The control unit in those cases is the circuit board. The purpose of a control unit is to control access to the device, while managing data reads and writes.

Modern mainframe DASD units are very sophisticated devices. They emulate a large number of control units and associated 3390 disk drives. The Host Adapters appear as control unit interfaces and can connect up to 32 channels (ESCON or FICON).

The physical disk drives are a serial interface, known as Serial Storage Architecture (SSA), which is used to provide faster and redundant access to the disks[7]. A number of internal arrangements are possible, but the most common involves many RAID 5 arrays with hot spares.

Note: For readers who are unfamiliar with the terminology, RAID stands for Redundant Array of Independent Disks. It refers to a family of methods for ensuring higher performance or reliability by using commodity hard drive technology. The benefits of RAID are handled for you on modern DASD units, so users can enjoy the benefits without the effort of configuration.

The current equivalent devices are the IBM 2105 Enterprise Storage Server® (ESS) and the IBM System StorageDS8000 family of disk subsystems. Practically everything in the unit has a spare or fallback unit. The internal processing (to emulate 3990 control units and 3390 disks) is provided by four high-end RISC processors in two processor complexes[8]. Each complex can operate the total system. Internal batteries preserve transient data during brief power failures. A separate console is used to configure and manage the unit.

A simplified illustration of modern DASD units is shown in Figure 1-7.

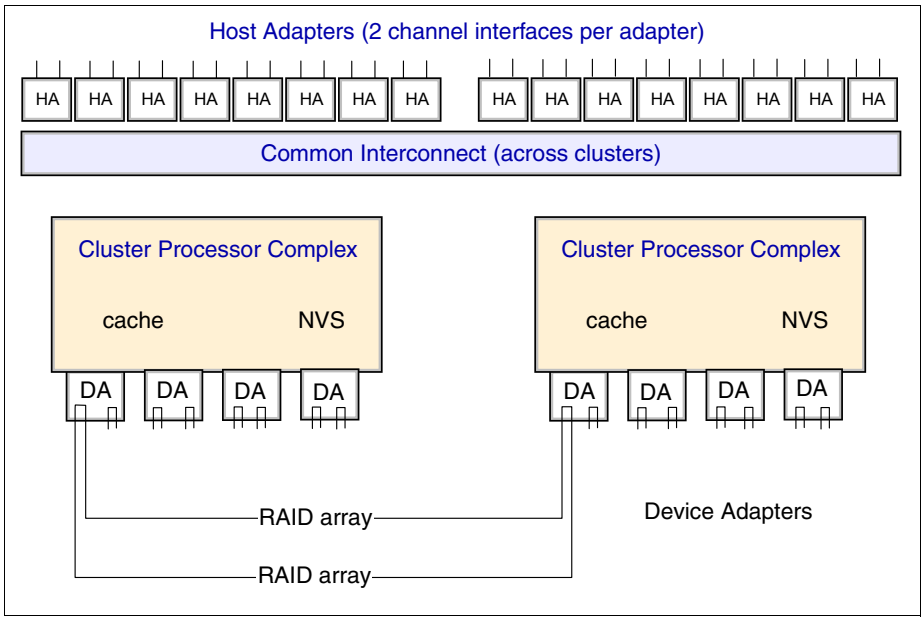


Figure 1-7 Current 3390 implementation

The 2105 and DS8000 offers many functions not available in real 3390 units, including FlashCopy®, Extended Remote Copy, Concurrent Copy, Parallel

[7] The DS8000™ family uses Fiber Channel disks in a fiber channel arbitrated loop (FC-AL) configuration.
[8] The DS8000 family uses a cluster of p5 systems with many advanced features such as storage partitions and so forth.

Access Volumes (PAV), Multiple Allegiance, a larger cache, and so forth. We discuss the z/VM support for some of these advanced features in Chapter 9, “System administration tasks” on page 275.

Clearly the simplistic 3390 disk drive and associated single control unit has much different underlying technology from the modern DASD units just discussed. However, in keeping with the backward compatibility that permeates the mainframe world, the basic architectural appearance to software is the same. This allows applications and system software written for very old 3390 disk drives to use the newer technology with no revisions.[9]

There have been several stages of new technology implementing 3390 disk drives; the DS8000 is the most recent of these. The process of implementing an architectural standard (in this case the 3390 disk drive and associated control unit) with newer and different technology while maintaining software compatibility is characteristic of mainframe development.

1.7.1 Types of DASD

In this section we briefly describe the types of DASDs used on the mainframe systems.

Extended Count Key Data (ECKD)

Extended Count Key Data (ECKD™), developed from the earlier Count Key Data (CKD) DASD, organizes its data in *tracks*. These equate to the path that a single read/write head on a disk drive would make in one revolution. The disks had multiple read/write heads which read data from multiple disk surfaces, therefore you had multiple tracks available whenever the read/write heads were in a particular position. This is known as a *cylinder*, which is a term that you will come across frequently when working with DASD and z/VM. A Model-3 3390 DASD will have 3339 cylinders available.

Fixed Block Architecture (FBA)

These address the DASD data differently in that the DASD is formatted into 512-byte blocks and these are addressed sequentially starting at the first block and numbered up to the end of the DASD.

SCSI

Small Computer System Interface (SCSI) drives are a recent addition to z/VM. They are supported by z/VM itself like the fixed block architecture (FBA) 9336-020 drives with up to a maximum of 2147483640 blocks. If not used by

[9] Some software enhancements are needed to use some of the new functions, but these are compatible extensions at the operating system level and do not affect application programs.

z/VM, they can be attached to operating systems that have the relevant support enabled.

1.7.2 Basic shared DASD

Within a mainframe environment it is usual to share data across systems and applications within those systems. Within a single z/VM system it is the z/VM Control Program that controls access to devices. However, if the DASDs are shared among different systems, then the operating systems must have a mechanism to not allow concurrent updates to the same data. This is done using channel commands RESERVE and RELEASE.

A basic shared DASD environment is illustrated in Figure 1-8. The figure shows z/VM images, but these could be any earlier version of the operating system. This could be two operating systems running on the same system or two separate systems; there is absolutely no difference in the concept or operation.

The capabilities of a basic shared DASD system are limited. The RESERVE command limits access to the entire DASD to the system issuing the command, and this lasts until a RELEASE command is issued. These commands work best when used to reserve DASD for a limited period or time.

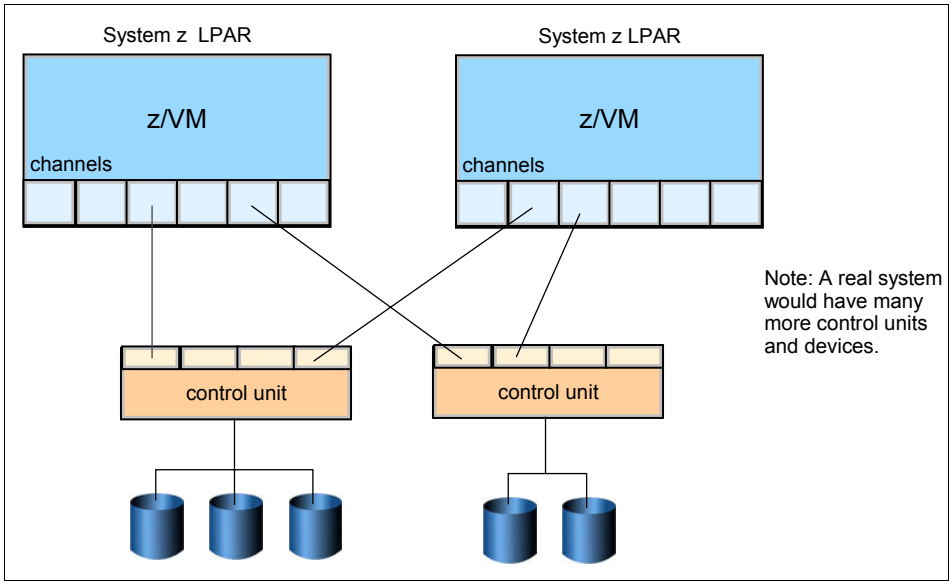


Figure 1-8 Basic shared DASD

Other types of devices or control units can be attached to both systems. For example, a tape control unit with multiple tape drives can be attached to both

systems. In this configuration, the operators can then allocate individual tape drives to the systems as needed.

1.8 I/O connectivity (channels)

As mentioned earlier, devices are connected to the mainframe CEC via channels. Unlike the old parallel channels used on the original mainframe architecture diagram (see Figure 1-2 on page 6), modern mainframe channels connect to only one control unit or, more likely, are connected to a *director* which handles the multiple paths to devices. Another difference is that modern channels are connected via optical fibers instead of the traditional copper wires. In the following sections, we discuss channel operations in more detail.

Channel subsystem

One of the main strengths of the mainframe computers is the ability to deal with a large number of simultaneous I/O operations. The channel subsystem (CSS) has a major role in providing this strength. The CSS manages the flow of information between I/O devices and central memory. By doing so, it relieves CPUs of the task of communicating directly with I/O devices and permits data processing to proceed concurrently with I/O processing.

The channel subsystem is built on the concept of a System Assist Processor (SAP) and the concept of channels. The SAP is one of the System z processor types. The SAP uses the I/O configuration loaded in the Hardware Storage Area (HSA) and knows which device is connected to each channel, as well as that device's protocol. The SAP manages the queue of I/O operations passed to the channel subsystem by the operating system.

Physical channel types

The channel subsystem may contain more than one type of channel path:

- Enterprise Systems Connection (ESCON)

  Since 1990, ESCON has replaced the S/370™ parallel channel as the main channel protocol for I/O connectivity, using fiber optic cables and a new switched technology for data traffic.

- FICON

  Based on the Fibre Channel Protocol (FCP), FICON was introduced in 1998. Because it is much faster than ESCON, it is becoming the most popular connection type.

- OSA-2 Open Systems Adapter-2

  A networking type of connector, OSA-2 Open Systems Adapter-2 can be configured to support various network protocols, such as Ethernet, Fast Ethernet, token-ring, and Fiber Distributed Data Interface (FDDI).

- OSA Express

  A faster networking connector, OSA Express supports fast Ethernet, Gigabit Ethernet, and 10 Gigabit Ethernet.

- OSA Express2

  OSA Express2 adds support for IBM Communication Controller for Linux, which provides a migration path for systems with dependency on SNA networking.

IOCDS

The I/O control layer uses a control file known as an I/O Configuration Data Set (IOCDS) that translates physical I/O addresses (composed of CHPID numbers, switch port numbers, control unit addresses, and unit addresses) into *device numbers* that are used by the operating system software to access devices. This is loaded into the HSA at power on (power-on Reset, or POR) and can be modified dynamically.

A device number looks like the addresses we described for early S/360 machines except that it can contain three or four hexadecimal digits. Without an IOCDS, no mainframe operating system can access I/O devices.

A subchannel provides the logical appearance of a device to the program and contains the information required for performing a single I/O operation. One subchannel is provided for each I/O device addressable by the channel subsystem when the system is activated.

ESCON and FICON

Recall our earlier discussion of the generic channel concepts born from the System 360 architecture. Current channels, such as ESCON and FICON, are logically similar to parallel channels but they use fiber connections and operate much faster. A modern system might have 100 to 200 channels or CHPIDs.[10] Key concepts include the following:

- ESCON and FICON channels connect to only one device or one port on a switch. When connected to a switch (port), many devices can be accessed simultaneously.

[10] The more recent mainframe machines can have more than 256 channels, but an additional setup is needed for this. The channels are assigned in a way that only two hexadecimal digits are needed for CHPID addresses.

- Most modern mainframes use switches between the channels and the control units. The switches may be connected to several systems, sharing the control units and some or all of its I/O devices across all the systems. The main advantage of using switches is that we can share a single I/O channel to connect to many I/O devices.
- CHPID addresses are two hexadecimal digits.
- Multiple partitions can sometimes share CHPIDs. Whether this is possible depends on the nature of the control units used through the CHPIDs. In general, CHPIDs used for disks can be shared.
- An I/O subsystem layer exists between the operating systems in partitions (or in the basic machine, if partitions are not used) and the CHPIDs.

Switches and directors

An ESCON director, FICON director, or switch is a sophisticated device that can sustain high data rates through many connections. (A large director might have 200 connections, for example, and all of these can be passing data at the same time.)

The difference between a *director* and a *switch* is that a director is somewhat more sophisticated and contains extra functionality, such as built-in redundancy for maximum fault tolerance. The director or switch must keep track of which CHPID (and partition) initiated which I/O operation, so that data and status information is returned to the right place. Multiple I/O requests, from multiple CHPIDs attached to multiple partitions on multiple systems, can be in progress through a single control unit.

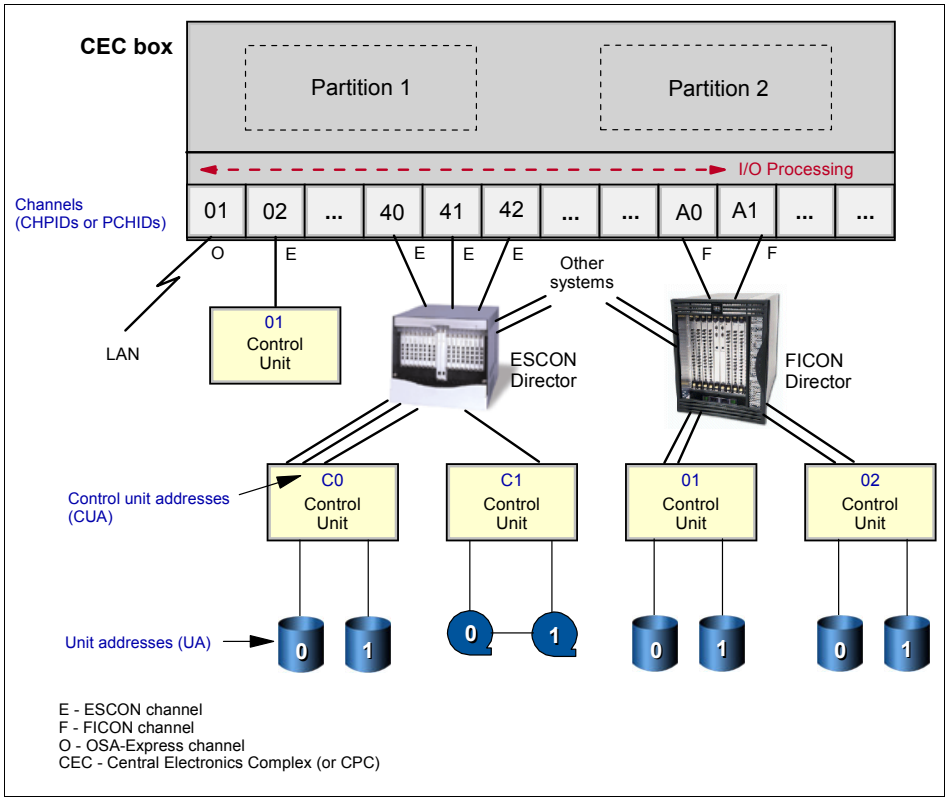


Figure 1-9 Recent system configuration

Modern control units, especially for disks, often have multiple channel (or switch) connections and multiple connections to their devices. They can handle multiple data transfers at the same time on the multiple channels.

Logical Channel Subsystem

The Logical Channel Subsystem (LCSS) concept was introduced with the more recent mainframes. The LCSS was created to enable the System z environment to handle more channel paths and devices available to the server. Each LCSS can have from 1 to 256 channels.

1.9 System control and partitioning

At this point, we have presented almost all the types of hardware needed for a basic understanding of the modern mainframe architecture. Having said that, we

still have not described how to control the mainframe, or how to partition the vast resources they contain.

1.9.1 Controlling the mainframe

There are many ways to illustrate a mainframe's internal structure, depending on the point of emphasis. Figure 1-10, while highly conceptual, shows several of the functions of the internal system controls on current mainframes. The internal controllers are microprocessors, but they use a much simpler organization and instruction set than System z processors. They are usually known as *controllers* in order to avoid confusion with System z processors.

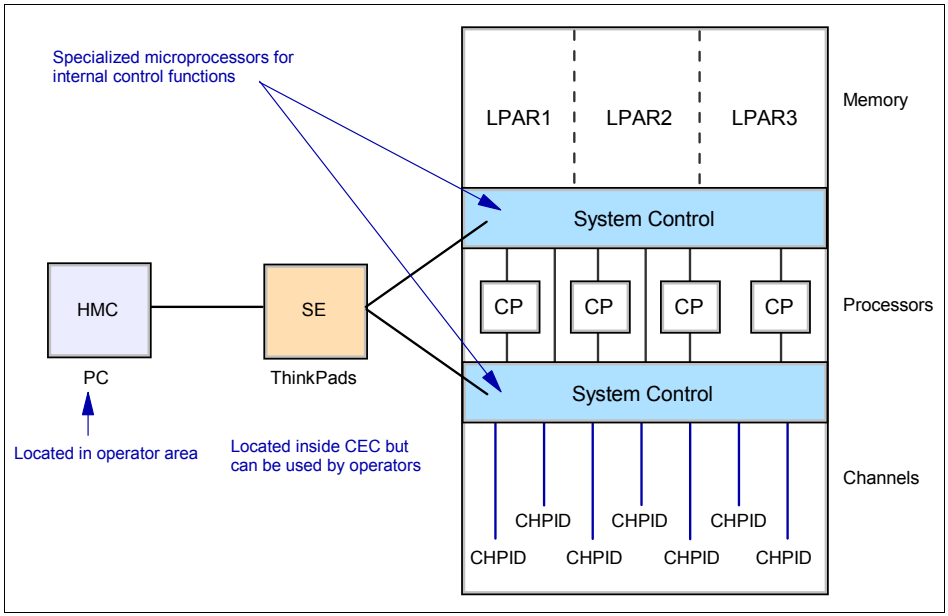


Figure 1-10 System control and partitioning

Tasks such as partitioning, allocating resources to partitions, and so forth on a mainframe are performed by using the HMC and SE discussed in 1.2, "Hardware Management Console" on page 8.

1.9.2 Logically partitioning resources

It is probable that, as a new z/VM user, you will not be given an entire mainframe to experiment with. In fact, as you begin learning about z/VM, you may not have to consider what resources your z/VM instance consumes. You may not even have access to the entirety of the z/VM installation you are executing on. It is

very likely that you will have a user account or ID for an already installed z/VM instance. This is similar to receiving an account on a Linux server without the root account.

But does this mean that the instance of z/VM that hosts your sessions is occupying the entirety of the underlying mainframe hardware? Probably not. Although a single large instance of z/VM could span the entirety of the hardware in your mainframe, it is likely you are running on a slice of the hardware only.

The Processor Resource/Systems Manager™ (PR/SM™) is a feature of IBM mainframes that enables logical partitioning of the CEC. A logical partition (LPAR) is a virtual machine at the hardware level. The IBM zSeries mainframes allow you to divide your physical machine into one or more LPARS (virtual machines), each of which contains a subset of your real machines processors, memory, and input/output devices.

This enables users to consolidate workloads and operating environments currently running on separate processors into one physical system, while using resource sharing to improve resource utilization and maintain performance.

Each LPAR operates as an independent server running its own operating environment. On the latest System z models, you can define up to 60 LPARs running z/VM, z/OS, Linux on IBM System z, and others. PR/SM enables each LPAR to have dedicated or shared processors and I/O channels, and dedicated memory (most of the resources can be dynamically reconfigured as needed). In other words, PR/SM transforms physical resources into virtual resources so that several logical partitions can share the same physical resources.

LPARs are, in practice, equivalent to separate mainframes. Each LPAR runs its own operating system. This can be any mainframe operating system; there is no need to run z/VM, for example, in each LPAR. The installation planners may elect to share I/O devices across several LPARs, but this is a local decision.

The system administrator can assign one or more system processors for the exclusive use of an LPAR. Alternatively, the administrator can allow all processors to be used on some or all LPARs. Here, the system control functions (often known as *microcode* or *firmware*) provide a dispatcher to share the processors among the selected LPARs. The administrator can specify a maximum number of concurrent processors executing in each LPAR. The administrator can also provide weightage for different LPARs (specifying, for example, that LPAR1 should receive twice as much processor time as LPAR2).

Note: The hardware and firmware that provide partitioning are known as Processor Resource/System Manager (PR/SM). PR/SM functions are used to create and run LPARs.

This difference between PR/SM (a built-in facility) and LPARs (the result of using PR/SM) is often ignored and the term LPAR is used collectively for the facility and its results.

System administrators assign portions of memory to each LPAR; memory cannot be shared among LPARs. The administrators can assign processors (noted as CPs in Figure 1-10 on page 24) to specific LPARs, or they can allow the system controllers to dispatch any or all the processors to all the LPARs using an internal load-balancing algorithm. Channels (CHPIDs) can be assigned to specific LPARs or can be shared by multiple LPARs, depending on the nature of the devices on each channel and how the system administrator wants to control the resource allocation between the LPARs on a system.

A system with a single processor (CP processor) can have multiple LPARs. PR/SM has an internal dispatcher that can allocate a portion of the processor to each LPAR, much as an operating system dispatcher allocates a portion of its processor time to each process, thread, or task.

Partitioning control specifications are partly contained in the IOCDS and are partly contained in a system *profile*. The IOCDS and profile both reside in the Support Element (SE), which is simply a notebook computer inside the system. The SE can be connected to one or more Hardware Management Consoles (HMCs), which are desktop personal computers used to monitor and control hardware such as the mainframe microprocessors. An HMC is more convenient to use than an SE, and it can control several different mainframes.

Working from an HMC (or from an SE, in unusual circumstances), an operator prepares a mainframe for use by selecting and loading a profile and an IOCDS. These create LPARs and configure the channels with device numbers, LPAR assignments, multiple path information, and so forth. This is known as a *power-on Reset* (POR). By loading a different profile and IOCDS, the operator can completely change the number and nature of LPARs and the appearance of the I/O configuration. However, doing this is usually disruptive to any running operating systems and applications and is therefore seldom done without advance planning.

Each LPAR can be started and stopped separately. Each LPAR runs a separate copy of an operating system, and each LPAR has its own operator console. If the system in one LPAR crashes, there is no effect on the other LPARs because each LPAR is isolated from all other LPARS on the system.

In Figure 1-10 on page 24, for example, we might have a production z/VM in LPAR1, a test version of z/VM in LPAR2, and Linux for S/390 in LPAR3. If our total system has 8 GB of memory, we might have assigned 4 GB to LPAR1, 1 GB to LPAR2, 1 GB to LPAR3, and have kept 2 GB in reserve for some reason. The operating system consoles for the two z/VM LPARs might be in completely different locations.[11]

For most practical purposes there is no difference between, for example, three separate mainframes running z/VM (and sharing most of their I/O configuration) and three LPARs on the same mainframe doing the same thing. With minor exceptions z/VM, the operators, and the applications cannot detect the difference.

The main advantage of using LPARs is to run z/VM, and then use z/VM to run many *virtual* servers.

Now that you have a firm grasp of the underlying hardware needed for a z/VM installation, Chapter 2, “Introduction to virtualization and z/VM” on page 29 further explains the concepts of virtualization.

1.10 Exercises

To help test your understanding of the material in this chapter, answer the following questions:

1. *Multiprocessor* means several processors (and that these processors are used by the operating system and applications). What does *multiprogramming* mean?
2. Usually, each read or write operation on a non-shared DASD device is one I/O operation. How many I/O operations will be involved when the DASD is in a shared mode?
3. What is the role of PR/SM?
4. What changes are needed in order for z/VM applications to work in an LPAR?

[11] Linux does not have an operator console in the sense of the z/VM consoles.

2

Introduction to virtualization and z/VM

Before discussing z/VM, we need to introduce the topic of virtualization. z/VM is the most mature software virtualization product on the market, and z/VM is the main virtualization enabler for Linux on the mainframe.

As a z/VM system programmer or system administrator, you need to understand both the concept of virtualization and how z/VM implements virtualization.

Objectives

After completing this chapter, you will be able to:

- Understand the virtualization concept
- Explain different ways of virtualizing a server
- Explain the role of virtualization in the world of mainframes
- List the benefits of virtualization
- Explain z/VM in the context of virtualization
- Describe the differences between guest and host systems
- Describe the difference between a first level and second level guest system

© Copyright IBM Corp. 2007. All rights reserved.

2.1 What is virtualization

Virtualization is the ability for a computer system to share resources so that one physical server can act as many virtual servers. z/VM allows the sharing of the mainframe's physical resources such as disk, memory, network adapters and CPUs. These resources are managed by a hypervisor.

z/VM's hypervisor is called Control Program (CP). When the user logs onto z/VM, the hypervisor creates a virtual machine which can run one of many different mainframe operating systems, like z/OS, z/TPF, Linux, z/VSE, CMS or z/VM.

Creating many virtual machines consisting of virtualized processors, communications, storage, and I/O devices can reduce administration costs and the overhead of planning, purchasing, and installing new hardware to support new workloads. Through the “magic” of virtualization, software running within the virtual machine is unaware that the “hardware” layer has been virtualized. It believes it is running on its own hardware separate from any other operating system. This can reduce the number of processors and hardware devices needed.

When it comes to Information Technology, the virtualization concept is defined best by the following quote from Jonathan Eunice, Inc.

> Virtualization is the process of presenting computing resources in ways that users and applications can easily get value out of them, rather than presenting them in a way dictated by their implementation, geographic location, or physical packaging. In other words, it provides a logical rather than physical view of data, computing power, storage capacity, and other resources.

Virtualization creates an external interface that hides the underlying implementation. The concept of LPARs, as discussed in Chapter 1, “Introduction to the mainframe hardware systems” on page 1, is a classic example of virtualization; that is, one physical mainframe is divided into multiple *virtual* mainframes.

It is important to note that splitting a single physical entity into multiple virtual entities is not the only method of virtualization. For example, combining multiple physical entities to act as a single, larger entity is also a form of virtualization, and *grid computing* is an example of this kind of virtualization. The grid virtualizes heterogeneous and geographically dispersed resources, thus presenting a simpler view. But although this type of virtualization is presented here for completeness, it is not what z/VM is.

Virtualization is most commonly applied to servers, storage, and networks. It can also be applied to nonphysical resources, including applications, middleware, distributed systems, and even virtual resources themselves (for example, virtualizing a cluster of virtual servers).

2.2 Benefits of virtualization

The cost of administering IT systems is growing faster than the cost of new hardware for those systems, because the complexity of those systems requires growing numbers of people to manage them. And the primary concern of management is to contain cost, while increasing revenue levels.

Introducing virtualization can be a critical first step in managing computing infrastructures in the following ways:

- By lowering the cost of existing infrastructure
- By reducing the complexity of adding resources to that infrastructure
- By building heterogeneous infrastructure across multiple data centers, making those centers more responsive to business needs

The benefits of virtualization vary, depending on the objectives and the specific virtualization technologies selected, as well as on the existing IT infrastructure. Not all users obtain the same benefits from implementing a particular virtualization solution. However, users realize many of the following benefits to some degree, even when using virtualization for simple server consolidation.

Higher resource utilization

Virtualization enables the dynamic sharing of physical resources and resource pools, resulting in higher resource utilization, especially for variable workloads where the average needs are much less than an entire dedicated resource.

Lower management costs

Virtualization can improve staff productivity by reducing the number of physical resources that must be managed; hiding some of the resource complexity; simplifying common management tasks through automation, better information and centralization; and enabling workload management automation. Virtualization also enables common tools to be used across multiple platforms.

Usage flexibility

Virtualization enables resources to be deployed and reconfigured dynamically to meet changing business needs.

Improved security and guest isolation

Virtualization enables separation and compartmentalization that is not available with simpler sharing mechanisms, and that provides controlled, secure access to data and devices. Each virtual machine can be completely isolated from the host machine and other virtual machines. If one virtual machine crashes, none of the other is affected.

Virtualization prevents data from leaking across virtual machines, and ensures that applications communicate only over configured network connections.

Higher availability

Virtualization enables physical resources to be removed, upgraded, or changed without affecting users.

Increased scalability

Resource partitioning and aggregation enable a virtual resource, depending on the product, to be much smaller or much larger than an individual physical resource, meaning that you can make scale adjustments without changes to the physical resource configuration.

Interoperability and investment protection

Virtual resources can provide compatibility with interfaces and protocols that are unavailable in the underlying physical resources. This is increasingly important for supporting existing systems and ensuring backward compatibility as done by z/VM.

Improved provisioning

Virtualization can enable resource allocation to a finer degree of granularity than individual physical units. Virtualized resources, because of their abstraction from hardware and operating system issues, are often capable of recovering much more rapidly after a crash than a physical resource.

Consolidation

Virtualization enables multiple applications and operating systems to be supported in one physical system, as well as consolidating servers into virtual machines on either a scale-up or scale-out architecture, It also enables systems to treat computing resources as a uniform pool that can be allocated to virtual machines in a controlled manner.

2.3 How virtualization works

Virtualization deals with enabling basic systems management of multiple, often heterogeneous systems right “out of the box”.

As Figure 2-1 shows, partitioning and virtualization involve a shift in thinking from physical to logical by treating IT resources as logical pools rather than as separate physical entities. This involves consolidating and pooling IT resources, and providing a “single system illusion” for both homogeneous and hetrogeneous servers, storage, distributed systems, and networks.

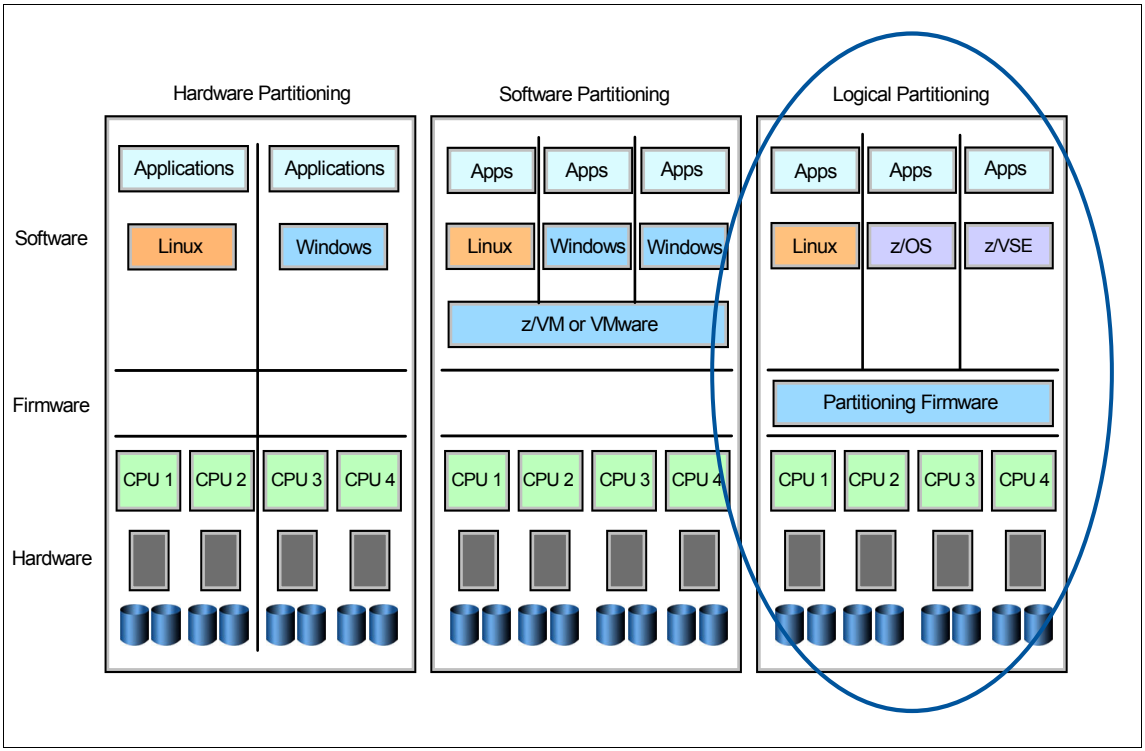


Figure 2-1 Logical partitioning[1]

Partitioning of hardware involves separate CPUs for separate operating systems, each of which runs its specific applications. Software partitioning employs a software-based “hypervisor” to enable individual operating systems to run on any or all of the CPUs.

[1] z/VM does not run Windows® as a “guest” operating system because Windows is not a mainframe operating system. Figure 2-1 simply shows the operating systems that run on z/VM or VMWare; it does not show the operating systems which run on both.

Hypervisors allow multiple operating systems to run on a host computer at the same time. Hypervisor technology originated in the IBM VM/370, the predecessor of the z/VM we have today. Logical partitioning (LPAR) involves partitioning firmware (a hardware-based hypervisor) to isolate the operating system from the CPUs.

Virtualization enables or exploits four fundamental capabilities: resource sharing, resource aggregation, emulation of function, and insulation. We explore these topics in more detail in the following sections.

2.3.1 Resource sharing

Multiple virtual resources can be defined to share the same single physical resource, either by allocating portions of the physical resource to each virtual resource, or by time-sharing the physical resource.

Figure 2-2 illustrates virtualization by resource sharing. With this type of virtualization, virtual resource users can share the physical resource. Virtual resources can also provide users with isolation that they would not get from sharing a physical resource directly, thus improving security and availability.

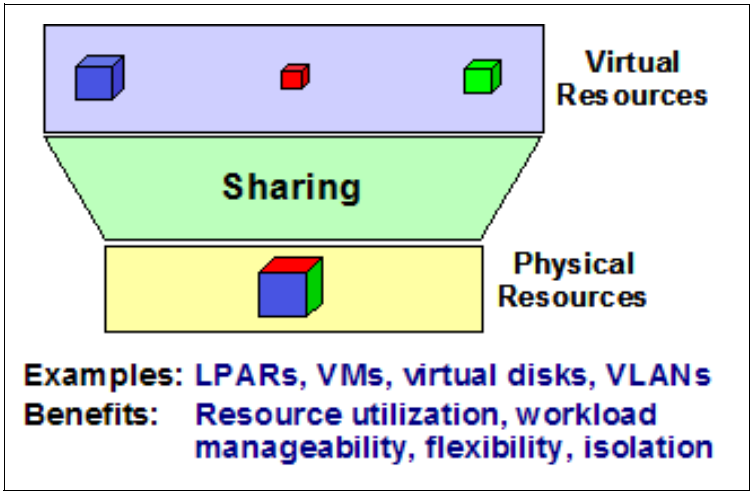


Figure 2-2 Virtualization by resource sharing

2.3.2 Resource aggregation

Virtual resources can span multiple physical resources, thus increasing their apparent capacity and simplifying their use and management. Figure 2-3 on page 35 illustrates virtualization by resource aggregation. For example, storage

virtualization can be used to create a virtual disk from free space on multiple physical disks. The virtual disk can be larger than any available physical disks.

As we will see later, aggregating items like discrete disks into larger logical disk pools is a task that z/VM handles very effectively.

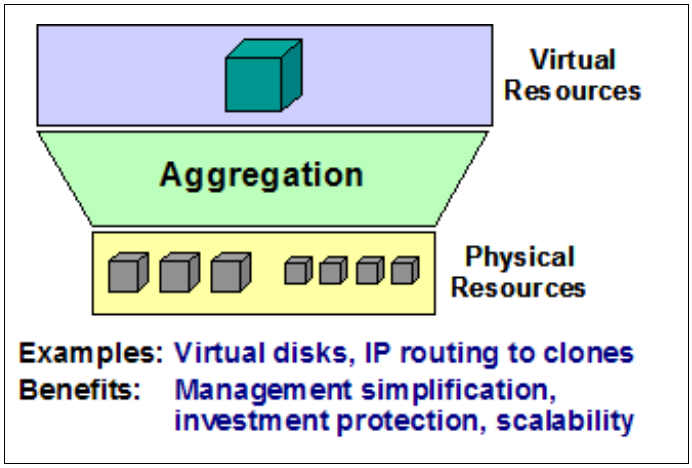


Figure 2-3 Virtualization by resource aggregation

2.3.3 Emulation of function

Virtual resources can have functions or features that are not available in their underlying physical resources. Figure 2-4 on page 36 illustrates virtualization by resource emulation.

Examples include architecture emulation software that implements one processor's architecture using another; iSCSI, which implements a virtual SCSI bus on an IP network; and virtual-tape storage implemented on physical disk storage.

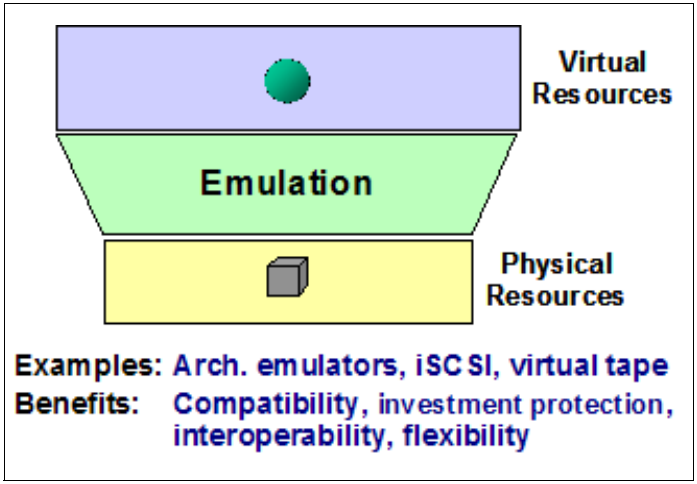


Figure 2-4 Virtualization by resource emulation

Another type of emulation presents virtualized resources as standard components that can differ from the underlying components. An example would be presenting all Ethernet interfaces as one specific model of Ethernet interface. This type of emulation allows users to develop a standardized environment and deploy on multiple different physical environments.

IBM z/VM was based in an era of hardware that we would today consider obsolete. But due to rigorous enforcement of backward compatibility, applications written to exploit an earlier (possibly out of production) tape unit may now operate on a functionally equivalent virtual tape unit that is backed by modern computing hardware.

2.3.4 Insulation

Mapping virtual resources to physical resources enables those underlying physical resources to be changed without affecting the consumers of the virtual resources. Figure 2-5 on page 37 illustrates virtualization by resource insulation.

An example of resource insulation is the CPUgard™ option. Imagine that one of your virtual processor is backed by a physical processor that is failing. With CPUgard enabled, you can continue to use your virtual processor while it is seamlessly moved to another physical processor automatically. Another example is advanced disk controllers that use redundant disks to automatically hide device failures from users.

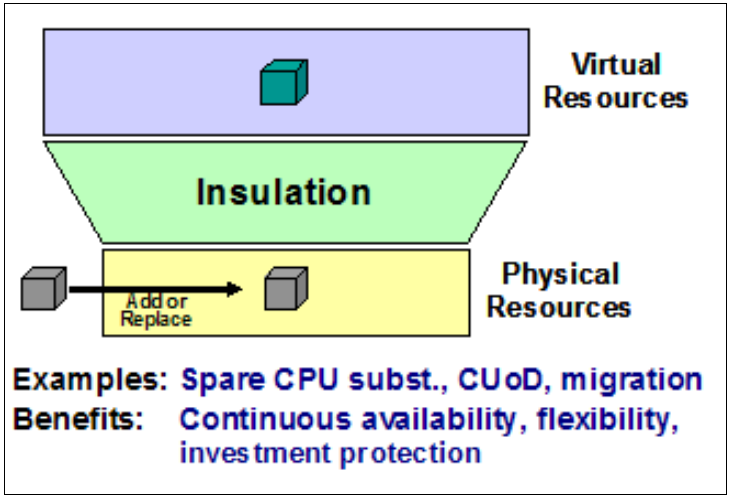


Figure 2-5 Virtualization by resource insulation

Although they can, and often do, act identically, physical systems and virtual systems are much different. A physical system actually exists in the form of hardware and software resources. A virtual system is an *abstraction* of a physical system, and its existence is dependent on the resources of the physical system. An individual physical system can support many virtual systems.

2.4 Server virtualization

Server virtualization allows a single server to be used by multiple applications, middleware, and operating systems concurrently, often without any knowledge or understanding of each other. The earliest forms of server virtualization included virtual memory, virtual I/O, and emulation. These early forms were followed by application and subsystem virtualization, where multiple copies of the application, or middleware stack could be run under the control and management of a single operating system.

Hardware partitioning and hypervisors are the two main implementation approaches for server virtualization.

2.4.1 Hardware partitioning

Hardware partitioning subdivides a physical server into separate computing environments, each of which can run an operating system instance and the

associates compatible applications. Figure 2-6 on page 38 illustrates hardware partitioning.

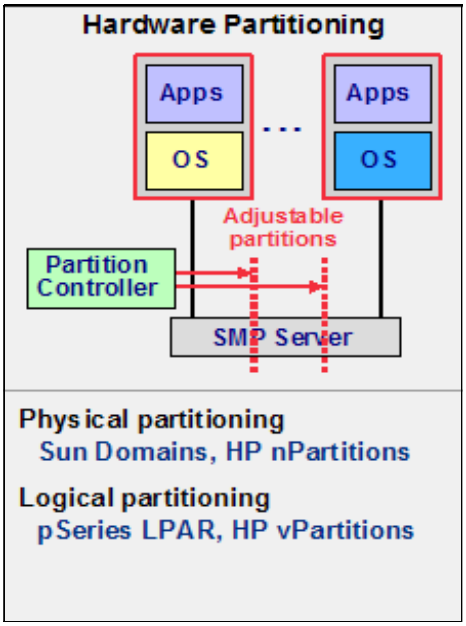


Figure 2-6 Hardware partitioning

Based on the technique used to subdivide a physical server into fractions, hardware partitioning is further divided into two types: physical partitioning, and logical partitioning, as explained here.

Physical partitioning

Physical partitioning, in which hardware resources are physically subdivided, has become less common now that servers are more reliable. Sun Domains and HP nPartitions are examples of physical partitioning.

Logical partitioning (or sometimes, physical hypervisor)

In logical partitioning, hardware resources are logically subdivided. This is the most common hardware partitioning technique used today. When the logical partitioning code is implemented as firmware on the hardware, this type of partitioning is referred to as *physical hypervisor*.

2.4.2 Hypervisor-based partitioning

Hypervisors use a thin layer of code to achieve fine-grained, dynamic resource sharing. There are two different types of hypervisors: Type-1 hypervisors and Type-2 hypervisors, as explained here.

Note: When the hypervisor function is implemented as part of the software, is is sometimes referred to as a “software hypervisor”.

Type-1 hypervisor

Type-1 hypervisors have the function implemented as part of the operating system software or the hardware firmware on a given hardware platform. When the hypervisor is implemented as part of the operating system, this operating system providing the hypervisor function is called the “host” operating system.The operating system running inside a virtual machine (VM) created on the host system is called the “guest” operating system.

Figure 2-7 illustrates a Type-1 hypervisor. The Hypervisor box shown in the center of the figure is either the firmware hypervisor (physical hypervisor) or an operating system (host operating system) with the hypervisor function built in. The boxes labelled OS are the virtual machines (guest OS).

z/VM is an example of a Type-1 Hypervisor. Some other examples are the Xen OpenSource Hypervisor, VMWare ESX Server, Virtual Iron, and ScaleMP.

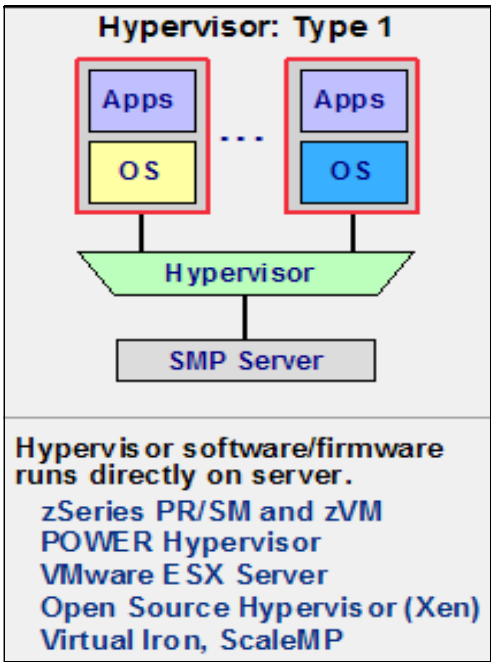


Figure 2-7 Type-1 hypervisor

Type-2 Hypervisor

Type-2 hypervisors are hypervisors running on a host operating system as an application. In this case, the host operating system refers to the operating system on which the hypervisor application is running.

The host operating system runs directly on the hardware. The operating system running inside the virtual machine (VM) created using the hypervisor application is the guest operating system in this context.

Figure 2-8 illustrates a Type-2 hypervisor. The box labelled Host OS is the host operating system. The box labelled Hypervisor is the hypervisor function running as an application on the host operating system. The other OS boxes are the virtual machines (guest operating systems) running on top of the hypervisor application running on the host operating system.

VMWare GSX, Microsoft® Virtual Server, Win4Lin, and UserModeLinux are some examples of this type of hypervisor.

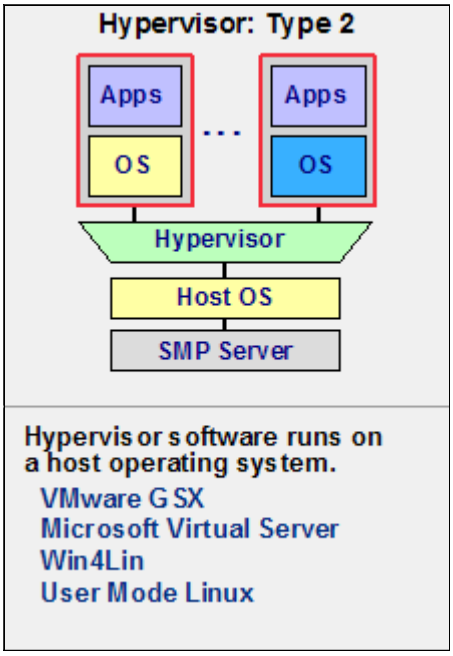


Figure 2-8 Type-2 hypervisor

2.4.3 Hypervisor technologies

Before concluding the server virtualization presentation, it is useful to discuss the important hypervisor technologies used.

Trap and emulate method

In this method, a guest operating system runs in user mode and the hypervisor runs in privileged mode. Privileged instructions issued by guest operating systems are trapped by the hypervisor. This technology was originally used by mainframes in the 1960s and 1970s (VM/370). Figure 2-9 illustrates the trap and emulate method.

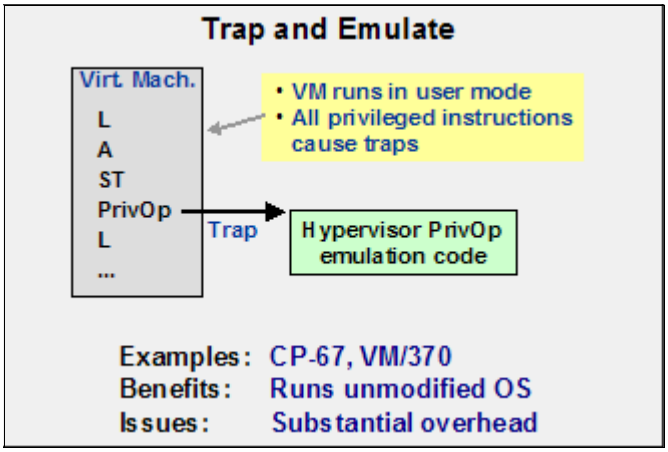


Figure 2-9 Trap and emulate method

Note: The instructions indicated inside the Virt Mach. box in Figure 2-9 on page 42 are assembly language instructions. Assembly language is a low-level language used for programming computers. The term “assembler” is often used in normal professional usage.

Translate, trap and emulate method

The translate, trap and emulate method is almost identical to the trap and emulate method. The difference is that some of the guest instructions must be replaced with trap operations, so some guest kernel binary translation may be required. The translate, trap and emulate method is illustrated in Figure 2-10 on page 43.

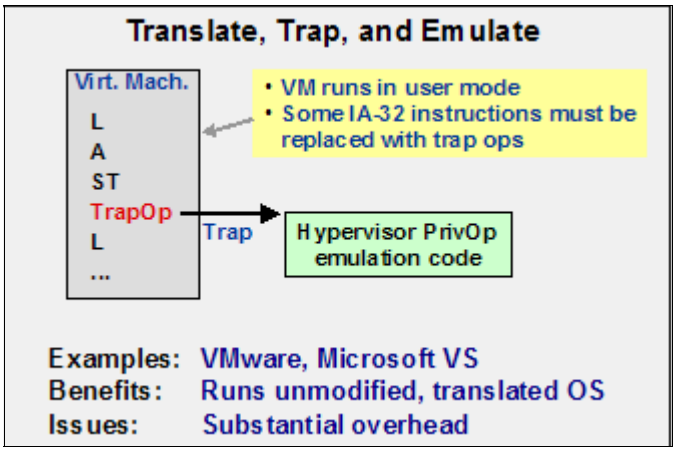


Figure 2-10 Translate, trap and emulate method

Hypervisor call method (paravirtualization)

In the hypervisor call method (also known as *paravirtualization*), a guest operating system runs in privileged mode and the hypervisor runs in super-privileged mode. This method is illustrated in Figure 2-11.

The guest operating system kernel (for example, AIX®, i5/OS®, or Linux®) is modified to do hypervisor calls for I/O, memory management, yield rest of time slice, and so on. Memory mapping architecture is used to isolate guests from each other and to protect the hypervisor.

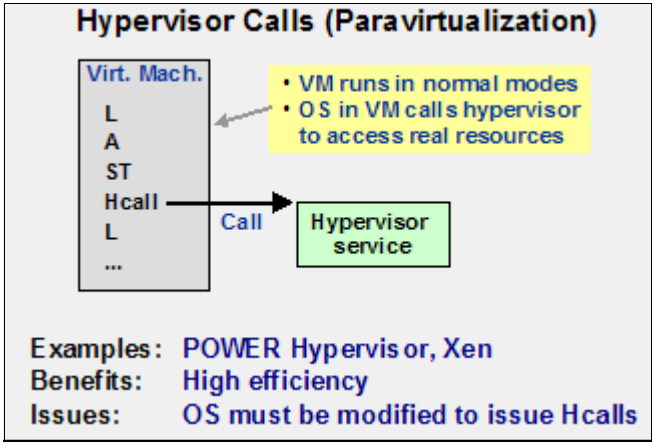


Figure 2-11 Hypervisor call method (paravirtualization)

Direct hardware support method

In the direct hardware support method, the guest operating system runs in privileged mode and the hardware runs most of the virtualization. The guest operating system can be run unmodified, but can issue some hypervisor calls to improve performance or capability such as I/O (z/VM) and yield time slice (PR/SM™ and z/VM). This method provides extensive hardware assists for hypervisor (virtual processor dispatching, I/O pass-through, memory partitioning, and so on).

The direct hardware support method, which is illustrated in Figure 2-12, is used by the IBM System z and zSeries (PR/SM™ and z/VM) family of mainframes.

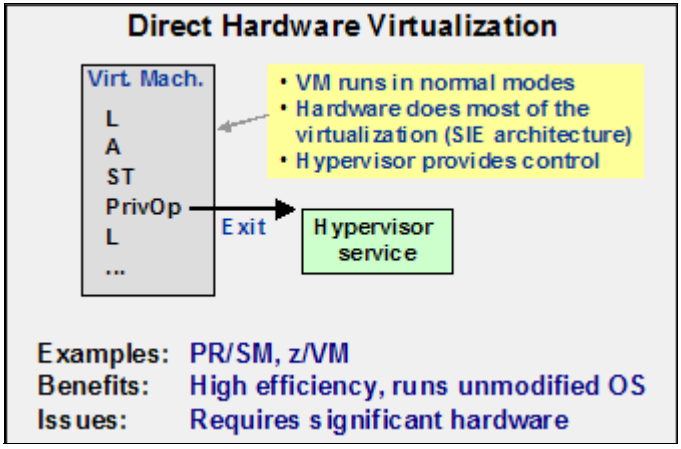


Figure 2-12 Direct hardware virtualization method

2.5 Virtualization on the mainframe

Typically, IBM mainframes can be partitioned in three different ways: basic mode, LPAR mode, and z/VM guest, as explained here.

- In the basic mode of operation, the entire physical system is used as a single system. Logical partitions (LPAR) are not supported in this mode of operation. This is the least used mode of operation. The choice of basic mode of operation is selected during the system activation (POR) time. On the newer mainframes, the basic mode of operation is not supported.
- In the logically partitioned or LPAR mode of operation, a single mainframe system is logically partitioned into multiple partitions. Here, too, the mode of operation is selected during the system activation (POR). The LPAR mode of operation is the most common mode used on mainframes. Depending on the

machine, the LPAR mode may provide additional facilities not available in basic mode, and these facilities can be exploited with operating system support.

- IBM z/VM guest implementations are software level partitioning. The z/VM operating system runs either on a LPAR or, on older hardware, on the complete mainframe (basic mode). Then virtual machines (VM) are created on top of the z/VM system to host “guest” systems.

2.5.1 I/O definition and partition profiles

The information contained in the I/O definitions and the partition profiles determine such items as how the mainframe will be partitioned, which resources each LPAR receives, and so on. Figure 2-13 illustrates this arrangement; note that LP1, LP2, LP3, and LP4 represent four different LPARs.

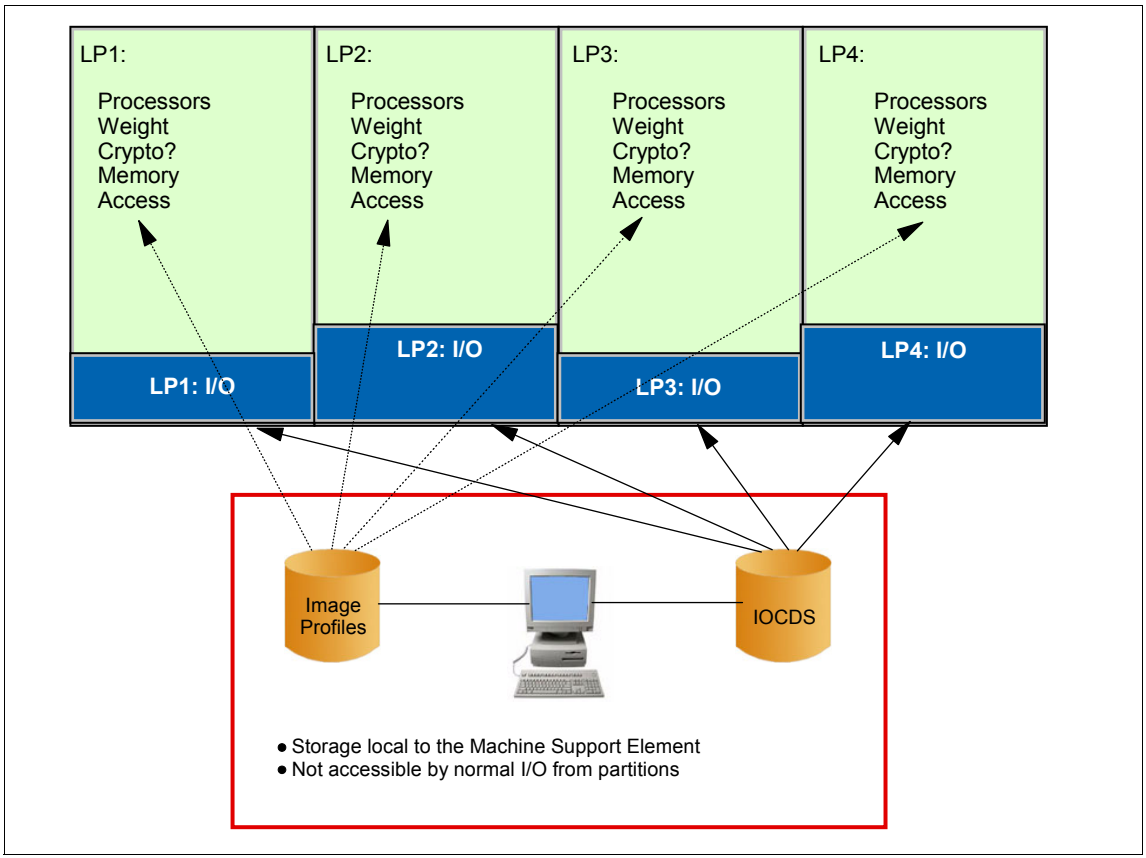


Figure 2-13 I/O definition and partition profile

Figure 2-13 on page 45 contains the following components.

Machine Support Element (SE)

- Internal to the CPC
- A hardware console to be accessed by a trusted operator

I/O Control Data Set (IOCDS)

Maximum four per machine but only one can be active at a time. IOCDS contains:

- Partition names
- Partition numbers
- I/O device and channel access

The IOCDS can only be modified from a partition with special access.

Partition Image Profiles

- One per partition
- Number of logical processors defined here
- Processors are defined as dedicated or shared (shared with “weight”)
- Crypto hardware access is defined here
- Memory is defined here
- Machine facilities access information (for example, IOCDS update authority) is defined here

The IOCDS and Partition Image Profiles are created and maintained by an operator or system programmer.

2.5.2 How LPARs are created

The steps to create and activate a LPAR are as follows.

- Define the LPAR name, LPAR partition number, and LPAR channels and devices and create the IOCDS[2].
- Create individual Partition Image Profiles.
- Perform the CPC activation (POR) and select the IOCDS created.
- LPAR mode of operation should be selected in the POR Activation Profile
- Perform individual partition activation and resource allocation according to the Image Profiles. The individual partition activation can be done automatically, after the CPC activation (POR), or it can be done manually, one at a time.

[2] The IOCDS can be defined from the Support Element (SE) with limited flexibility, or by using the IOCP and HCD tools in z/VM and z/OS for greater flexibility and error checking.

Logical processor assignment

There are two ways in which logical processors can be assigned to an LPAR: by using dedicated logical processors, or by using shared logical processors, as explained here.

Dedicated logical processors

In this method of processor assignment, processors are dedicated to the LPAR. This is backed by assigned physical processors and these processors are “locked” to that partition. So, in this method, processors are not shared between LPARs.

Shared logical processors

In this method of processor assignment, processors are shared across LPARs. This is backed by a “pool” of processors. The partition *weight* determines an LPAR’s share of the pool. The partition weight is a numeric value and higher the value, higher the weight (or preference) given for that partition.

The share is calculated by dividing the assigned partition weight by the sum of all the “active” partition weights. A partition can exceed its share if other partitions are not using their full share. This can be prevented, however, by “hard capping” the share for a partition. The capping method used for preventing a partition from exceeding its Millions of Service Units (MSU) capacity is known as “soft capping”.

Figure 2-14 on page 48 is a logical representation of resource allocation.

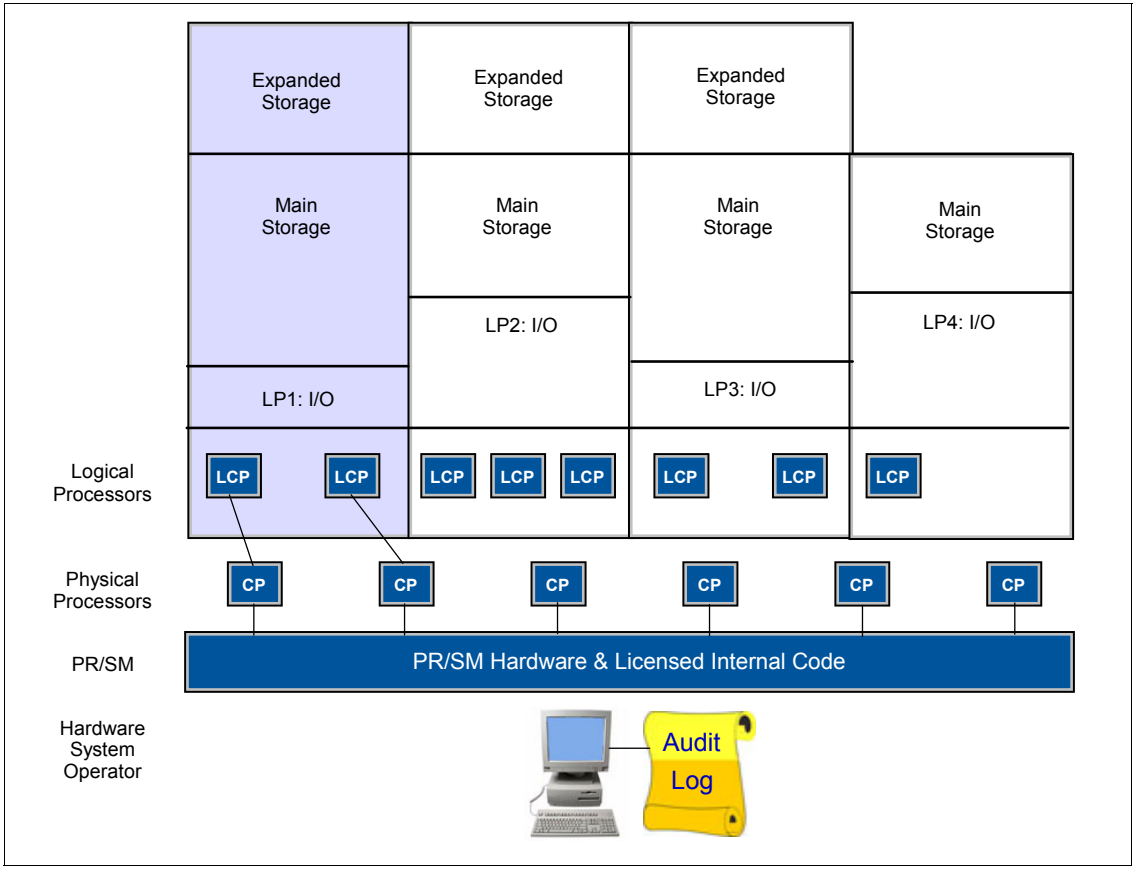


Figure 2-14 Resource allocation and activation

2.5.3 Additional mainframe virtualization facilities

In this section, we discuss additional mainframe virtualization facilities.

Multiple Image Facility (MIF)

MIF enables channel sharing among multiple LPARs. I/O devices on shared channel paths can be accessed simultaneously by sharing LPARs (or restricted to a subset of sharing LPARs).

Logical Channel Subsystems (LCSS) support

LCSS allows an IBM System z9® or zSeries® z990 to be configured with up to 1024 channels (512 channels for zSeries z890). 256 channels can be configured for each LPAR, with selected channel sharing among LPARs possible.

HiperSockets

HiperSockets™ provides high-speed, security-rich TCP/IP connectivity among LPARs. Although not a real device, HiperSockets uses a portion of the memory as the “wire”. Therefore, HiperSockets provides memory speed communications between the LPARs of a physical system.

Host Page-Management Assist

Host Page-Management Assist (HPMA) is the interface to z/VM paging and storage management. HPMA is designed to allow hardware to assign, lock, and unlock page frames without hypervisor assistance.

Layer 2 (MAC) and Layer 3 (IP) network switching

OSA and z/VM Layer 2 and Layer 3 network support enables virtual IP and MAC network switching without requiring a hosting partition.

2.6 Virtualization in action

Next, we examine real-life computing situations where virtualization plays an important role.

2.6.1 Virtualization in a test environment

Maxim: Test virtual, deploy physical

Using virtualization software is a cost-effective and scalable approach to creating a test environment. The virtualization software enables multiple instances of the same or different operating systems to be installed on the same physical machine. As the test environment grows, you can add more physical machines so that the test environment always simulates the production network.

Another positive effect is that, in a test environment, not all systems are working to their full capacity at the same time. Thus the workload is balanced on the same physical hardware concurrently. Greater flexibility is introduced by allowing the testers to conduct testing anywhere, anytime. It also allows other testers to remotely check from home to see what other machines are available, or to perform testing that would normally be performed at work.

A virtualized testing environment allows clients to spend less money on hardware, and enables them to create additional virtual servers by upgrading disk space and RAM, instead of buying complete new machines. In addition to reducing capital requirements, virtualization also reduces space requirements,

electrical power consumption, and cooling costs by enabling multiple test and development systems to be consolidated into far fewer physical systems.

Installing different operating systems on separate virtual machines on the same physical machine emulates a cross-platform environment with, for example, UNIX® or Linux installed on some of the virtual machines and Windows on others. Clients and servers can all be installed on the same physical computer, in separate virtual machines (VMs). Assigning each VM its own IP address enables all of them to communicate with each other on the test network, as well as communicate with other physical machines on the test network.

Virtualization software can be effectively used to test any major changes to the production network. Virtualization can also increase efficiencies through shorter test cycles. Centralized libraries of virtual test environments can be reused as needed, thus drastically reducing configuration times. Because these virtual environments are essentially available on demand, server idle time can also be reduced. Virtualization also simplifies the often time-consuming and resource-intensive testing of distributed server applications.

Ultimately, the increased efficiencies and effectiveness resulting from virtualization empower developers to get products to market more quickly.

2.6.2 Virtualization to maintain outdated software

Virtualization protects the investment of companies in their computer systems by enabling them to extend the lives of both their custom-written and their legacy applications. Without virtualization, organizations would often find it difficult to justify the added support and maintenance costs for existing hardware and operating systems that is needed to keep those applications intact. But virtualization eliminates the need to migrate existing custom applications to the latest hardware and operating systems. Companies can simply create a dedicated virtual machine for each custom of legacy application, and run it alongside other virtual machines running mainstream applications, all on the latest, industry-standard server hardware.

Using virtualization, it is possible to restore a backup copy of any virtual environment in a discrete and efficient manner. Even a system that has been corrupted through different interactions within a test phase can be easily restored.

Separating hardware and software management also enables enterprises to manage and maintain discrete systems far more effectively.

2.6.3 Improving availability and resilience

Network virtualization can be viewed as a collection of capabilities that virtualize the IT resources used to physically connect applications, servers and storage systems to the network or to other virtual servers—resources such as IP addresses, network adapters, LANs, bandwidth management, and more. By virtualizing networks, customers can pool and share network elements to make communication across their IT infrastructure faster, more efficient, cost-effective, secure and available. Virtualized networks are also more flexible, allowing customers to implement changes to the infrastructure as required so they can adapt to business needs in real time, with minimal downtime.

Virtual LAN (VLAN) technology uses software to configure a network of computers to behave as if they were physically connected, even though they are not. VLANs provide the isolation required to ensure that data from customer A is not interspersed with data from customer B as the data flows across the shared network. By provisioning and configuring VLANs from a central point, customers can ensure consistency between the servers and the switches they connect to, and eliminate outages due to device failure or software failure in the device.

Virtualization of business functions is achieved by isolating service definition and usage from service implementation. Services may be implemented using a wide range of technologies. Service requests dispatch service requests to a service provider that offers the capabilities they want, but the service requesters do not need to be aware of how they are implemented.

In simplest terms, an *operating system* is a collection of programs that manage the internal workings of a computer system. Operating systems are designed to make the best use of the computer's various resources, and ensure that the maximum amount of work is processed as efficiently as possible. Although an operating system cannot increase the speed of a computer, it can maximize its use, thereby making the computer seem faster by allowing it to do more work in a given period of time.

A computer's *architecture* consists of the functions the computer system provides. The architecture is distinct from the physical design, and, in fact, different machine designs might conform to the same computer architecture. In a sense, the architecture is the computer as seen by the user, such as a system programmer. For example, part of the architecture is the set of machine instructions that the computer can recognize and execute. In the mainframe environment, the system software and hardware comprise a highly advanced computer architecture, the result of decades of technological innovation.

2.7 Introducing z/VM

z/VM is a operating system for the IBM System z platform that provides a highly flexible test and production environment. The z/VM implementation of IBM virtualization technology provides the capability to run full-function operating systems such as Linux on System z, z/OS, and others as “guests” of z/VM. z/VM supports 64-bit IBM z/Architecture guests and 31-bit IBM Enterprise Systems Architecture/390® guests. For more information about the types of guests that can be hosted by z/VM, refer to Chapter 13, “Guest operating systems” on page 387.

z/VM provides each user with an individual working environment known as a *virtual machine*. The virtual machine simulates the existence of a dedicated real machine, including processor functions, memory, networking, and input/output (I/O) resources. Operating systems and application programs can run in virtual machines as guests. For example, you can run multiple Linux and z/OS images on the same z/VM system that is also supporting various applications and end users. As a result, development, testing, and production environments can share a single physical computer.

A virtual machine uses real hardware resources, but even with dedicated devices (like a tape drive), the virtual address of the tape drive may or may not be the same as the real address of the tape drive. Therefore, a virtual machine only knows “virtual hardware” that may or may not exist in the real world. In Figure 2-15 on page 53 you can see the layout of the general z/VM environment.

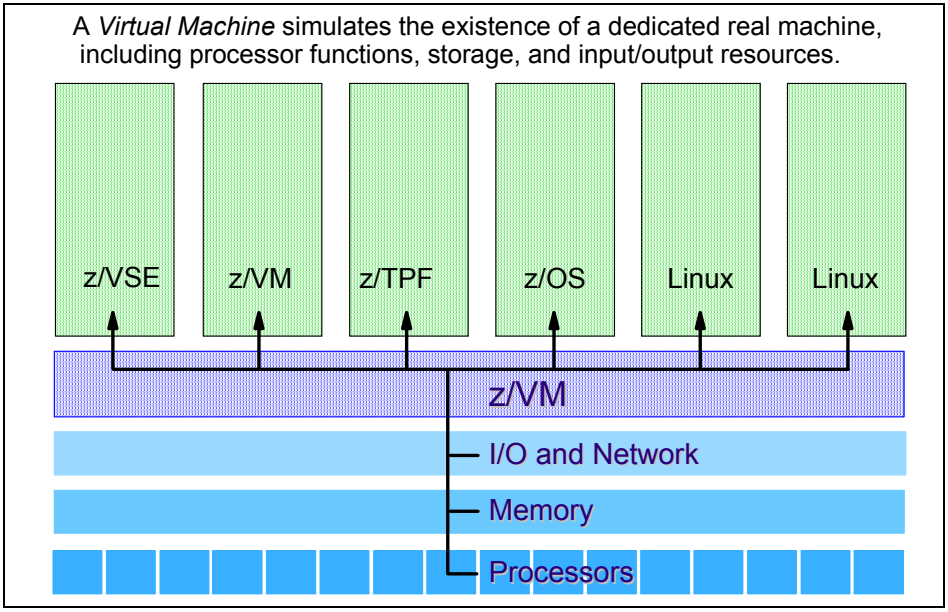


Figure 2-15 General z/VM environment

2.7.1 The virtual machine capability of z/VM

The virtual machine capability of z/VM allows you to perform the following tasks:

- Test programs that can cause abnormal termination of real machine operations and, at the same time, process production work.

  The isolation that is provided for a virtual machine enables system-oriented programs and teleprocessing applications, for example, to be tested on the virtual machine while production work is in progress, because this testing cannot cause abnormal termination of the real machine.

- Test a new operating system release.

  A new release of an operating system can be generated and tested at the same time that the existing release is performing production work. This enables the new release to be installed and put into production more quickly.

  The ability to operate multiple operating systems concurrently under z/VM may enable an installation to continue running programs that operate only under a back-level release (programs that are release-sensitive and uneconomical to convert, for example) concurrently with the most current release.

- Test a new operating system.

  The existing operating system can be used to process production work concurrently with the generation and testing of a new operating system. Experience with the new system can be obtained before it is used on a production basis, without dedicating the real machine to this function.

- Perform operating system maintenance concurrently with production work.

  The installation and testing of program temporary fixes (PTFs) for an operating system can be done at the same time that normal production operations are in progress.

- Provide backup facilities for the primary operating system.

  A generated z/VM system is not model-dependent and can operate on various server models as long as the minimum hardware requirements are present. This enables a smaller server model that has less real storage, fewer channels, fewer direct access devices, and fewer unit record devices than a larger server model to provide backup for the larger model (normally at a reduced level of performance).

- Perform operator training concurrently with production work processing.

  The real machine does not have to be dedicated to training additional or new operators or to providing initial training when a new operating system is installed. Operator errors cannot cause termination of real machine operations.

- Simulate new system configurations before the installation of additional channels and I/O devices.

  The relative load on channels and I/O devices can be determined using the simulated I/O configuration rather than the real I/O configuration. Experience with generating and operating an I/O configuration for multiple guests can be obtained using one real machine.

- Test customer-written system exits.

  Customer-written system exits can be tested without disrupting production work.

2.7.2 Types of operating environments

The System z architecture provides three different operating environments in which the operating systems can run: native, in an LPAR, or as guest under z/VM.

Native mode

In the native mode of operation (sometimes also called “basic mode”), the entire physical system is used as a single system. This is the least-used mode of operation and on newer mainframes, basic mode of operation is not supported.

LPAR mode

In the logically partitioned or LPAR mode of operation, a single mainframe system is logically partitioned into multiple partitions. The LPAR mode of operation is the most common mode of hardware partitioning used on mainframes. Depending on the machine, the LPAR mode may provide additional facilities not available in basic mode, which can be exploited with operating system support.

As a guest under z/VM

z/VM guest implementations are software-level partitioning. The z/VM operating system runs either on a LPAR or, on older hardware, on the complete mainframe (basic mode). Then virtual machines (VM) are created on top of the z/VM system to host guest systems.

2.7.3 First-level versus second-level guest system

A first-level z/VM is the base operating system that is installed on top of the real hardware. A second-level operating system is a system that is created upon the base z/VM operating system. Therefore, z/VM as a base operating system runs on the hardware, while a guest operating system runs on the virtualization technology.

In other words, there is a first-level z/VM operating system that sits directly on the hardware, but the guests of this first-level z/VM system are virtualized; see Figure 2-16 on page 56. By virtualizing the hardware from the guests, we are able to create and use as many guests as needed with a small amount of hardware.

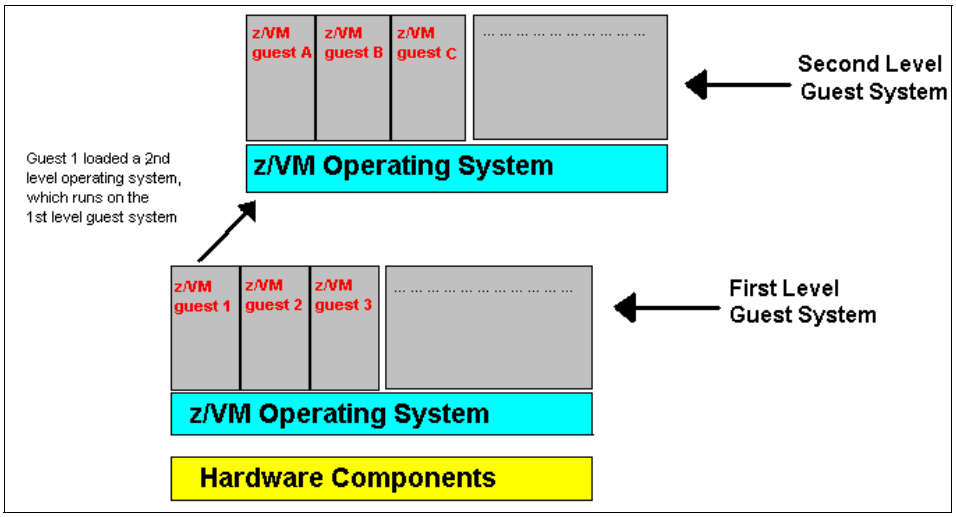


Figure 2-16 First level versus second level

As previously mentioned, operating systems running in virtual machines are often called “guests”. Other terms and phrases you might encounter are:

- “Running first level” means running directly on the hardware (which is what z/VM does).
- “Running second level”, “running under VM”, or “running on (top of) VM” means running as a guest.

An example of the functionality of z/VM is, if you have a first-level z/VM system and a second-level z/VM system, you could continue to create more operating systems on the second-level system. This type of environment is particularly useful for testing operating system installation before deployment, or for testing or debugging operating systems.

Note: For more detailed information about running guest operating systems under z//VM, refer to Chapter 13, “Guest operating systems” on page 387.

2.7.4 z/VM strengths

z/VM is built on a foundation of system integrity and security, and incorporates many design features for reliability and availability. In the following sections, we look at these strengths in more detail.

Integrity and security

- z/VM supports guest use of the cryptographic facilities provided by supported IBM servers.
- IBM will correct any integrity exposures introduced by unauthorized programs into the system.
- Kerberos authentication and Secure Sockets Layer (SSL) support are provided through TCP/IP for z/VM.
- Integrated access control and authentication services may be augmented with the addition of an external security manager (ESM), such as the RACF® Security Server for z/VM.

Availability and reliability

- Application recovery

  z/VM provides services which permit recovery of incomplete interactions with resource managers.

- Automated operations

  z/VM offers several levels of automated system management support. One example is the Programmable Operator. For a higher degree of automation, IBM SystemView® Host Management Facilities/VM can be added. Both the Programmable Operator and Host Management Facilities/VM can interact with NetView® on z/VM, which in turn can interact with NetView on z/OS.

- Transparent synchronization of duplexed data

  z/VM provides duplexed data with transparent ongoing synchronization between the primary and backup copy, and automatic transparent switching to the backup copy in case of an error in the primary copy.

- Dynamic reconfiguration

  Many aspects of z/VM can be reconfigured dynamically, thereby eliminating the need to reboot the system because of a configuration change.

- Connectivity

  z/VM systems can be connected for improved server and user availability.

Fast restart reduces the end-user impact of any outage.

2.8 Exercises

To help test your understanding of the material in this chapter, answer the following questions:

1. Which method of virtualization is used in grid computing?
2. How do you compare physical hypervisors and software hypervisors?
3. What could be the reason for removing the “basic mode” of operation option on the latest mainframes?
4. True or False - The processor weight can be adjusted in the IOCDS.
5. *What* is needed (and not how) to create an LPAR? Where does an LPAR get it?
6. What are the three types of operating environments that we introduced in this chapter?
7. List at least three operating systems that can run as z/VM guests.

3

History of z/VM

This chapter offers a brief history of z/VM and discusses various elements of the environment in which it was conceived and developed.

Objectives

After completing this chapter, you will be able to:

- Understand early data processing and its origins
- Understand the beginnings of z/VM
- Understand the evolution of VM to the current day's z/VM

© Copyright IBM Corp. 2007. All rights reserved.

3.1 Life before VM

In the early days of information technology, computing largely consisted of very specialized, mainly mechanical, machines that could be used to sort and collate data that was entered into them on media such as punch cards and paper tape. The output from these was normally more punched cards, paper tape or printed output. Any programming that was needed was built into the machine or could be changed using a patch panel that could be wired to change things such as sort and collate criteria.

Note the panel on the front of the IBM 711 punched card reader shown in Figure 3-1. When opened, this would reveal the panel of cross-connected wires that comprised the patch panel.

Figure 3-1 IBM 711 punched card reader

The challenges of the 1940s encouraged a huge growth in computing technology in what we would now call the “scientific sector”, where large amounts of computing power handled small amounts of data.

During the 1950s this technology was adapted to more commercial use and the processing of large amounts of data that required small amounts of computing

power. These machines, such as the IBM 705 introduced in 1954, were among the first to offer programming capabilities similar to the ones that we see in today's mainframes. Input to these systems was still by using punch cards or paper tape for both programs and data, and output was still to printer or punch/paper tape. Today, this type of data processing would be known as "batch processing". Operator consoles, often modified typewriters, allowed some form of control over the system.

It was not until 1951 that tape drives were introduced, followed by the first commercially available disk drive, the IBM RAMAC® in 1956, which provided 5 megabytes of storage. Data processing using punch cards and paper tape continued up to the late 1980s, coexisting with the newer technologies that had evolved alongside.

Having discussed the input and output requirements of these systems, it is useful to explain what punch cards and paper tape were and how they have impacted computing since those days. Let us take a look at the punch card in Figure 3-2.

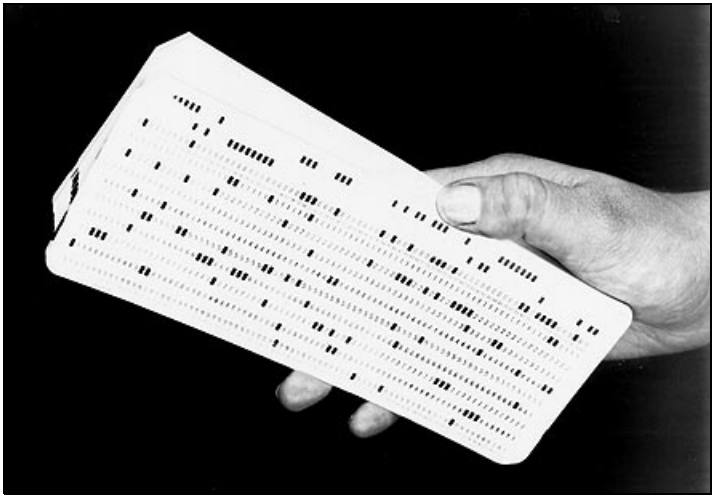


Figure 3-2 A typical punch card

Punch cards had developed into an industry standard. They had precisely 80 columns and 12 rows. The holes from top to bottom were numbered 12, 11, and then 0 to 9.

Using a card punch, small rectangular holes were punched that represented the data. For instance, a 12 hole and a 1 hole in the same column represented the character A, a 12 and a 2 hole represented the character B.

Some readers may have recognized the legacy of this as, in hexadecimal notation, 12 is represented as the character C and 1 is still 1; therefore the character A becomes C1—which is the EBCDIC representation that we still use today. Similarly, paper tape was the origin of ASCII character representation.

A punch card contains one record and, as you can see, is of fixed length. Paper tape is a continuous stream of data and therefore could be very long. This is probably the origin of different file systems used today. That is, record-based file systems such as we use in the IBM mainframe environment are based on the punch card. Data stream file systems, as found in UNIX, are based on paper tape.

3.2 VM from the beginning

In the 1950s the president of IBM, T.J. Watson, Jr., gave the Massachusetts Institute of Technology (MIT) an IBM 704 machine for use by MIT and other New England schools. Then, each time IBM built a newer, bigger processor, it upgraded the system at MIT.

In 1959 various academic papers began exploring the idea of time sharing on these large, fast processors. Research continued in this area and in 1961 the Compatible Time Sharing System (CTSS) was demonstrated on an IBM 709 processor. This was the beginning of VM as we know it today.

As CTSS evolved, it became apparent that not all problems with this new technology could be solved by using software alone. The researchers worked with IBM to get changes made to the hardware and with these changes in place, CTSS was in full production by 1963. The file system and some commands such as LOAD, LISTF and RENAME would be recognizable by CMS users today.

While CTSS was being developed, IBM began to develop what was to become one of the defining architectures of modern computing, the IBM System 360.

System 360

At the end of 1964, work began on a project to develop a new kind of operating system, Control Program-40 (CP-40). It was a system that would provide not only virtual memory, but also virtual machines (see 2.7, “Introducing z/VM” on page 52). Each user would have a complete System/360™ virtual machine (which at first was called a “pseudo-machine”).

These “virtual” machines would emulate the System/360 architecture, making the virtual environment transparent to any operating system running in a virtual machine. Hardware changes were also made to the System/360 to allow better

use of virtual memory. At this time work also started on a console monitoring system that eventually became the Conversational Monitor System (CMS). CMS drew heavily on the lessons taught by CTSS although, unlike CTSS, each CMS user would have its own virtual machine.

The key concept of the CP/CMS relationship was that CP solved the problem of multiple use by providing separate computing environments at the machine instruction level for each user. CMS then provided a single user service that did not have to worry about the problems of sharing, allocation, and protection of resources such as memory and disk storage.

This was followed in 1968 by CP-67, which added many features to CP and CMS and took advantage of new hardware facilities that had been developed in conjunction with the software. Version 1 of CP-67 was released in 1968 followed by Version 2 in 1969 and Version 3 in 1970, by which time it was running in 44 sites, of which a quarter were IBM sites.

System 370

In 1970, IBM announced the next generation of its mainframe processors: System/370™. Development on VM continued to support this new architecture. In 1972 IBM announced the following developments.

- Two new computers: the 370/158 and the 370/168
- Address relocation hardware on all 370s (important for virtual memory management).
- Four new operating systems:

VM/370	
DOS/VS	A virtual storage version of DOS (the ancestor of what is now z/VSE)
OS/VS1	A virtual storage version of MFT
OS/VS2	A virtual storage version of MVT (the ancestor of what is now z/OS)

The design point for VM/370 Release 1 was a 512 kilobyte main storage IBM 370/145 processor. Compare this to the many gigabytes that we can now support on mainframes. Many CP-67 users migrated to VM/370. One of the strengths of the platform was much appreciated by the users; that is, the file system was upwardly compatible to the new operating system and hardware. This upward compatibility continues to this day and is one of the strengths of mainframe computing.

Throughout the 1970s, development continued and new releases of VM were delivered that added more features for virtual machine support, real device support, and hardware exploitation. The final release of VM/370 was Release 6

which appeared in 1979. This release also included a new networking product called Remote Spooling Communications Subsystem (RSCS), which was to become the backbone of IBM internal communications until the adoption of Internet-based communications in the late 1990s.

By the late 1970s, work had also begun on the next major advance in the mainframe hardware, the introduction of Extended Architecture (XA). Before this, the operating system was responsible for all of the input/output (I/O) by using CPU instructions that could otherwise be used to perform useful work for users.

XA architecture took much of this work and ran it on specialized I/O processors that were part of the hardware. There was also a new instruction specifically for the use of CP: the Start Interpretive Execution (SIE) instruction.

This instruction was used by CP to run the virtual machines of users almost directly on the hardware, thus reducing the processor cycles that had been used to do this by CP in System 370s. VM was developed to emulate this new hardware environment in virtual machines in a non-XA hardware environment. This provided a very useful tool for users to migrate, because they could run both 370 and XA virtual machines simultaneously on the same processor.

By 1980, VM was becoming increasingly popular and IBM released a packaged version of VM called VM/SP which made installation much easier than before and also facilitated the installation of the many new products that had been developed for this platform.

An example of this growth in applications, announced late in 1981, was a product called Professional Office System (PROFS®) which, by 1987, was estimated to have more than one million users worldwide. VM/SP Release 1 included other functionality that ensured its popularity: XEDIT and EXEC2. XEDIT, an easy-to-use editor, and EXEC2, a programming language, enabled some complex applications to be built very easily.

In 1981, IBM also announced the Extended Architecture (XA) platform and with it a version of VM that would assist customers migrating their MVS™ systems to the new environment, known as VM/XA Migration Aid. This program was destined to be developed by a different IBM laboratory to VM/SP and thus the two roles of VM separated: VM/XA became purely a provisory, and VM/SP supported the application and end-user environment.

As the 1980s progressed, VM/SP continued to add function with the subsequent releases of VM/SP2 through VM/SP6. The following list gives you some idea of the enhancements that were included during this period. (The list contains some

words and acronyms that you may not recognize at this point, but you will learn about them later as you progress through this publication.)

- RETRIEVE, FILELIST, NAMES, NOTE, PEEK, RDRLIST, RECEIVE, SENDFILE, and TELL - end-user enhancements.
- REXX - a powerful new (at that time) scripting language.
- Shared File System (SFS) - an enhanced file system for CMS.
- VM Serviceability Enhanced Staged (VMSES) - a tool to automate installation and maintenance.

In 1981 IBM also introduced VM High Performance Option (HPO), which was an additional feature to VM/SP that enhanced performance and memory addressability.

In parallel with VM/SP, VM/XA steadily acquired function, graduating from “Migration Aid” to “System Facility” in 1985 and then to “System Product” in 1988. VM/XA System Product Release 1, which became available March 1988, was called a “full-function” XA VM system. Its CP had been made compatible with HPO 5, and it provided a high-capacity, XA-capable CMS: CAMS 5.5. It would be followed by VM/XA SP Release 2 in 1988—but even this was not as functionally rich as VM/HPO, because it lacks some of the more advance networking capabilities.

By 1989 more than 20,000 VM licenses has been sold, but something needed to change.

System 390

On September 5, 1990, IBM announced the next step in the evolution of the platform, System/390® and the Enterprise Systems Architecture (ESA). VM/ESA® Releases 1.1.0 and 1.1.1 were announced the same day. VM/ESA shipped in two different guises: the ESA version, and the 370 version. They were functionally the same but ran on different architectures, one on the older System 370 architecture and one on the newer ESA. The two branches of VM that had separated in the early 1980s were now back together.

VM/ESA 1.1.1 included another major productivity aid: CMS PIPELINES (see Chapter 8, “CMS pipelines” on page 259). This had previously been a chargeable feature, but was now rolled in to the base product.

The VM/ESA 370 feature was the last release to support the 370 architecture.

VM/ESA was enhanced throughout the 1990s to support additional processor hardware and new devices, culminating in VM/ESA 2.4, which became available in 1999. This provided support for the latest IBM S/390 Enterprise Server Generation 5 and Generation 6. Guest support was also added for Queued

Direct I/O (QDIO) and FICON channels. It would also support the new zSeries architecture that became available in 2001.

zSeries

In 2001, IBM shipped z/VM Version 3 Release 1, which was very closely followed by z/VM 4.1. Also included in z/VM 4.1 was support for virtual LANs and Hipersockets and the shipment of TCP/IP as part of the base z/VM product, which reflected an acceptance of the importance of its role in modern computing.

Further releases followed; z/VM 4.2 later in 2001, z/VM 4.3 in 2002, and z/VM 4.4 in 2003. All of these releases added support for the latest hardware developments, and also focused on enhancements to the support of Linux guests. This environment was becoming increasingly important to the vitality of VM.

In 2004, IBM shipped z/VM 5.1, which contained numerous enhancements to support the new TotalStorage® disk subsystems, cryptography, additional VSWITCH features and much more. This was followed by z/VM 5.2 in 2005 and then, to bring us up to date, z/VM 5.3 in 2007. These latest releases introduced support for the latest developments in hardware, and delivered enhancements to guest support and manageability.

3.3 Exercises

1. What was the major enhancement in XA architecture that was used to dispatch guests more efficiently?
2. When were the first virtualization ideas presented?
3. What is the EBCDIC representation of the letter D?
4. What is SFS, and which decade saw its introduction?
5. What was a major difference between z/VM 3.1 and z/VM 4.1?

4

z/VM - job roles and basic concepts

In this chapter we explain the various job roles and responsibilities involved in managing z/VM on a mainframe, and present the basic concepts underlying z/VM. We introduce you to the z/VM directory, and explain how to log on to a z/VM system. Finally, we discuss basic z/VM commands.

Objectives

After completing this chapter, you will be able to:

- Describe the different job roles and responsibilities in the mainframe world
- Give examples of some major components of z/VM
- Understand the function of the z/VM directory
- Understand how to log on to z/VM

© Copyright IBM Corp. 2007. All rights reserved.

4.1 Roles in the mainframe world

Mainframe systems are designed to be used by large numbers of people. Most of those who interact with mainframes are “end users”, which are people who use the applications that are hosted on the system.

However, because of the large number of end users who use the system, the multiple applications running on the system, and the sophistication and complexity of the system software that supports the users and applications, a variety of job roles are needed to operate and support the system. This section introduces these roles and the industry terminology that describes them.

4.1.1 Introduction to roles

Job roles in the IT field are often referred to by a number of different titles; this text uses the following terms:

- System programmer
- System administrator
- Application designer and programmer
- System operator
- Production control analyst

In a distributed systems environment, many of the same roles are needed as in a mainframe environment. In a distributed environment, however, the job responsibilities are often not as well-defined.

Mainframe roles have evolved and expanded to provide an environment in which the system software and applications can function smoothly and effectively and serve many users efficiently. Although the mainframe support staff may involve a large number of people, the ratio of the staff is relatively small when you take into consideration the number of users supported, the number of transactions run, and the high business value of the work that is performed on the mainframe.

There are several major job responsibilities involved in supporting and maintaining a mainframe environment. Mainframe activities such as the following often require cooperation among various roles:

- Installing and configuring system software
- Designing and coding new applications to run on the mainframe
- Introduction and management of new workloads on the system, such as batch jobs and online transaction processing
- Operation and maintenance of the mainframe software and hardware

In the following sections, we describe each role in more detail.

System programmer

In a mainframe IT organization, the system programmer plays a central role by installing, customizing, and maintaining the operating system, and also installing or upgrading products that run on the system.

System programmer responsibilities cover a range of tasks. For example, a programmer might be presented with the latest version of the operating system and be expected to upgrade the existing systems. Or an installation project might be as simple as upgrading a single program (such as a sort application).

The following tasks are typically performed by a system programmer:

- Planning hardware and software system upgrades and changes in configuration
- Training system operators and application programmers
- Automating operations
- Capacity planning
- Running installation jobs and scripts
- Performing installation specific customizing tasks
- Integration testing new products with existing applications and user procedures
- Performing system-wide performance tuning to meet required levels of service

The system programmer must be also skilled at solving (“debugging”) problems with system software. These problems are often captured in a copy of the computer's memory contents called a *dump*, which the system produces in response to a failing software product, user job, or transaction. Armed with a dump and specialized debugging tools, the system programmer can determine where the components have failed. If the error occurred in a software product, the system programmer works directly with the software vendor’s support representatives to discover whether the problem’s cause is known and whether a patch is available.

System programmers are also needed to install and maintain the *middleware* on the mainframe, such as database management systems, online transaction processing systems and Web servers. Middleware is a software “layer” between the operating system and the end user or end-user application. It supplies major functions that are not provided by the operating system. Major middleware products can be as complex as the operating system itself, if not more so.

Proving complete information about the system programmer role is beyond the scope of this book, but it does provide a solid basis for the requisite continued education.

System administrator

The distinction between the system programmer role and the system administrator role varies widely among mainframe sites. In smaller IT organizations, where one person might be called upon to perform several roles, the terms may be used interchangeably.

In larger IT organizations with multiple departments, the job responsibilities of these two roles tend to be more clearly separated. The system administrator performs more of the day-to-day tasks related to maintaining the critical business data that resides on the mainframe. The system programmer, in contrast, focuses on maintaining the system itself.

One reason for the separation of duties is to comply with auditing procedures, which often require that no single person in the IT organization be allowed to have unlimited access to sensitive data or resources. Examples of system administrators include the database administrator (DBA) and the security administrator.

Often system programmer expertise lies mainly in the mainframe hardware and software areas. System administrator expertise is more likely to involve application experience. System administrators often interface directly with application programmers and end users to ensure that the administrative aspects of the applications are met. These roles are not necessarily unique to the mainframe environment, but they are key to its smooth operation nonetheless.

In matters of problem determination, the system administrator generally relies on the software vendor support center personnel to diagnose problems, read dumps, and identify corrections for cases in which these tasks are not performed by the system programmer.

This chapter provides an overview of the basic topics needed for an understanding of z/VM system administration. The topics include basic commands, security, networking, performance, and maintenance tasks. Each of these topics is vital for the system administrator role.

Application designer and application programmer/developer

The application designer and application programmer (or application developer) design, build, test, and deliver mainframe applications for an IT organization's end users and customers. Based on requirements gathered from business analysts and end users, the designer creates a design specification from which the programmer constructs an application.

In addition to creating new application code, the application programmer is responsible for maintaining and enhancing the enterprise's existing mainframe applications. In fact, this is often the primary job for many of today's mainframe

application programmers. Although mainframe installations still create new programs with Common Business Oriented Language (COBOL) or PL/I, languages such as Java have become popular for building new applications on the mainframe, just as they have on distributed platforms.

Widespread development of mainframe programs written in high-level languages such as COBOL and PL/I continues at a brisk pace. Many thousands of programs are in production on mainframe systems around the world, and these programs are critical to the day-to-day business of the corporations that use them. COBOL and other high-level language programmers are needed to maintain existing code and make updates and modifications to existing programs. Also, many enterprises continue to build new application logic in COBOL, as well as other traditional languages.

Although this publication is not specifically aimed at application development, we do explain how to use the editors and basic commands needed to begin programming in the z/VM environment. The sections on Xedit, CMS basics, and REXX are particularly instructive.

System operator

The system operator monitors and controls the operation of the mainframe hardware and software. The operator starts and stops system tasks, monitors the system output for unusual conditions, and works with the system programming and production control staff to ensure the health and normal operation of the systems.

As applications are added to the mainframe, the system operator is responsible for ensuring that they run smoothly. New applications from the application programming staff are typically delivered to the operations staff with a run book of instructions.

A *run book* identifies the specific operational requirements of the application, which operators need to be aware of during job execution. Run book instructions might include application-specific console messages that require operator intervention, recommended operator responses to specific system events, and directions for modifying job flows to accommodate changes in business requirements[1].

[1] Console messages were once so voluminous that operators often had a difficult time determining whether a situation was really a problem. In recent years, tools have been developed to reduce the volume of messages and automate message responses to routine situations. Such tools have made it easier for operators to concentrate on unusual events that might require human intervention.

The operator is also responsible for starting and stopping the major subsystems, such as transaction processing systems, database systems, and the operating system itself.

In case of a failure or an unusual situation, the operator communicates with system programmers, who assist the operator in determining the proper course of action. The production control analyst may also become involved, and work with the operator to ensure that production workloads are completing properly.

Production control analyst

The production control analyst is responsible for making sure that batch workloads run to completion without error or delay. Some mainframe installations run interactive workloads for online users, followed by batch updates that run after the prime shift when the online systems are not running.

Although this execution model is still common, worldwide operations at many companies with real-time, Internet-based access to production data are finding the “daytime online/night time batch” model to be obsolete. Batch workloads continue to be a part of information processing, however, and skilled production control analysts play a key role.

The production control analyst understands that the use of well-structured rules and procedures to control changes is a strength of the mainframe environment and helps to prevent outages. In fact, one reason that mainframes have attained such an outstanding reputation for high levels of availability and performance is because there are controls on change and it is difficult to introduce change without following proper procedures.

Vendors

A number of vendor roles are commonplace in the mainframe shop. Because most mainframe computers are sold by IBM, and the operating systems and primary online systems are also provided by IBM, most vendor contacts are IBM employees. However, independent software vendor (ISV) products are also used in the IBM mainframe environment, and customers use original equipment manufacturer (OEM) hardware, such as disk and tape storage devices, as well.

Here we list and explain typical vendor roles:

- Hardware support or customer engineer

  Hardware vendors usually provide onsite support for hardware devices. The IBM hardware maintenance person is often referred to as the customer engineer (CE). The CE provides installation and repair service for the mainframe hardware and peripherals. The CE usually works directly with the operations teams when hardware fails or new hardware is being installed.

- Software support

  A number of vendor roles exist to support software products on the mainframe[2]. IBM has a centralized Support Center that provides entitled and extra-charge support for software defects or usage assistance. There are also information technology specialists and architects who can be engaged to provide additional pre-sales and post-sales support for software products, depending upon the size of the enterprise and the particular customer situation.

- Field technical sales support, systems engineer, or client representative

  For larger mainframe accounts, IBM and other vendors provide face-to-face sales support. The vendor representatives specialize in various types of hardware or software product families, and call on the part of the customer organization that influences the product purchases. At IBM, the technical sales specialist is referred to as the field technical sales support (FTSS) person, or sometimes by the term systems engineer (SE).

  For larger mainframe accounts, IBM frequently assigns a client representative, who is attuned to the business issues of a particular industry sector and works exclusively with a small number of customers. The client representative acts as the general “single point of contact” between the customer and the various organizations within IBM.

4.1.2 Role review

Next, we examine how these IT roles relate to the z/VM system and its software components and concepts.

4.2 Components of z/VM

z/VM consists of the following components and facilities (base products):

- Control Program (CP)
- Conversational Monitor System (CMS)
- Transmission Control Protocol/Internet Protocol (TCP/IP) for z/VM
- Advanced Program-to-Program Communication/Virtual Machine (APPC/VM) Virtual Telecommunications Access Method (VTAM®) Support (AVS)
- Dump Viewing Facility
- Group Control System (GCS)

[2] This book does not examine the marketing and pricing of mainframe software. However, the availability and pricing of middleware and other licensed programs is a critical factor affecting the growth and use of mainframes.

- Hardware Configuration Definition (HCD) and Hardware Configuration Manager (HCM) for z/VM
- Language Environment®
- Open Systems Adapter Support Facility (OSA/SF)
- Restructured Extended Executor/Virtual Machine (REXX/VM)
- Transparent Services Access Facility (TSAF)
- Virtual Machine Serviceability Enhancements Staged/Extended (VMSES/E)

In addition to these basic products, z/VM also offers the following optional features:

- Data Facility Storage Management Subsystem for VM (DFSMS/VM™)
- Directory Maintenance Facility for z/VM (DirMaint™)
- Performance Toolkit for VM
- RACF Security Server for z/VM
- Remote Spooling Communications Subsystem (RSCS) Networking for z/VM

In the following sections we discuss each component and point you to references where you can learn more about each topic.

4.2.1 Control Program

Control Program (CP) is primarily a real-machine resource manager. CP provides each user with an individual working environment known as a “virtual machine”. Each virtual machine is a functional equivalent of a real system, sharing the real processor function, storage, console, and input/output (I/O) device resources.

When you first log on to z/VM, CP controls the working environment. Many of the facilities of z/VM are immediately available to you. For example, you can use CP commands to do various system management tasks. However, most of the work done on z/VM requires the Conversational Monitor System (CMS) or a guest operating system (such as z/OS) to help with data processing tasks and to manage work flow.

CP provides connectivity support that allows application programs to exchange information with each other and to access resources residing on the same z/VM system or on different z/VM systems.

Note: For more detailed information about the Control Program (CP), refer to Chapter 5, “Control Program for new users” on page 103.

4.2.2 Conversational Monitor System

Conversational Monitor System (CMS) provides a high-capacity environment that supports large numbers of interactive users. CMS can help you perform a wide variety of tasks. For example, you can write, test, and debug application programs for use on CMS or a guest system's base product, as listed here:

- Run application programs developed on CMS or guest systems
- Create and edit data files
- Process jobs in batch mode
- Share data between CMS and guest systems
- Communicate with other system users
- Provides a useful file system for storing data

CMS File System

The *file* is the essential unit of data in CMS. Files in CMS are unique and cannot be read or written using other operating systems. When you create a file in CMS, you name it using a file identifier (file ID). The file ID consists of three fields:

1. File Name (fn)
2. File Type (ft)
3. File Mode (fm) or Directory Name (dirname)

When you use CMS commands and programs to modify, update, or refer to files, you must identify the file by using these fields. Some CMS commands allow you to enter only the file name, or the file name and file type; others require you to enter the file mode or directory name as well.

Under z/VM, your files can be stored within a Shared File System (SFS) file space or on minidisks. Depending on your system configuration, you may have the option to use both methods of storing files. In this situation, you could store those files that you may want to share in your SFS file space; other files could be stored on minidisks.

z/VM OpenExtensions (POSIX support) includes another type of file called a byte file system (BFS) file. BFS files are organized in a hierarchy, as in a UNIX system. All files are members of a *directory*. Each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the BFS file space. Typically, a user has all or part of a BFS file space mounted as the root directory.

z/VM views an entire file hierarchy as a byte file system. Each byte file system is a mountable file system. The *root* file system is the first file system mounted. Subsequent file systems can be mounted on any directory within the root file system, or on a directory within any mounted file system.

All files in the byte file system are called BFS files. BFS files are byte-oriented, rather than record-oriented, like CMS record files on minidisks or in the Shared File System. You can copy BFS files into CMS record files, and you can copy CMS record files into the Byte File System.

Note: For more detailed information about CMS, refer to Chapter 6, “Conversational Monitor System” on page 147.

4.2.3 TCP/IP

TCP/IP for z/VM brings the power and resources of your mainframe server to the Internet. Using the TCP/IP protocol suite of TCP/IP for z/VM, you can reach open, multivendor networking environments from your z/VM system.

TCP/IP for z/VM allows z/VM systems to act as peers of other central computers in TCP/IP open networks. Applications can be shared transparently across z/VM, Linux, and other environments. Users can send messages, transfer files, share printers, and access remote resources across a broad range of systems from multiple vendors

TCP/IP for z/VM provides the following types of functions:

- Connectivity and gateway functions, which handle the physical interfaces and routing of data
- Server functions, which provide a service to a client (that is, send or transfer a file)
- Client functions, which request a certain service from a server anywhere in the network
- Network status and management functions, which detect and solve network problems
- Application programming interfaces, which allow you to write your own client/server application

Note: For more detailed information about TCP/IP for z/VM, refer to Chapter 11, “Networking and connectivity” on page 353.

4.2.4 APPC/VM VTAM Support (AVS)

AVS (APPC/VM VTAM Support) is a Virtual Telecommunications Access Method (VTAM) application that provides advanced program-to-program communication (APPC) services between VM and non-VM systems in an SNA network.

AVS and VTAM run in the same GCS group on a z/VM system. Together, AVS and VTAM enable APPC/VM application programs in a TSAF or communication services (CS) collection to communicate with the following applications:

- Other APPC/VM applications residing in other VM systems within the SNA network
- APPC applications residing in non-VM systems in the SNA network

4.2.5 Dump Viewing Facility

The Dump Viewing Facility (DVF) helps you interactively to diagnose system problems. Using this facility, you can display, format, and print data interactively from virtual machine dumps, as well as display and format recorded trace data.

The BLOCKDEF utility lets you display, format, and print control block information. The VIEWSYM command lets you display symptom records, making it easier to identify duplicate problems when they occur.

4.2.6 Group Control System (GCS)

Group Control System (GCS) runs in an XA or XC virtual machine in place of CMS. It is a virtual machine supervisor, providing multitasking services that allow numerous tasks to remain active in the virtual machine at one time.

One of the functions of GCS is to support a native Systems Network Architecture (SNA) network. The SNA network relies on ACF/VTAM, VTAM SNA Console Support (VSCS), and other network applications to manage its collection of links between terminals, controllers, and processors. GCS provides services for ACF/VTAM, VSCS, and the others, which eliminates your need for VTAM Communications Network Application (VM/VCNA) and a second operating system like VSE.

4.2.7 HCD and HCM for z/VM

HCD and HCM for z/VM, or Hardware Configuration Definition and Hardware Configuration Manager for z/VM, provides a comprehensive I/O configuration management environment, similar to that available with the z/OS operating system.

HCM runs on a Windows-based personal computer connected to the z/VM system through a TCP/IP network connection. HCM provides a graphical user interface, as well as commands, to help you configure your system. You supply the needed I/O configuration information to HCM, which processes the information and passes it to HCD.

HCD runs in a z/VM server virtual machine and performs the work of actually creating and changing the hardware and software aspects of your I/O configuration.

Although HCM provides the primary user interface to HCD, HCD also provides a backup user interface on your z/VM host for certain I/O configuration tasks, in case HCM is not available.

The original dynamic I/O configuration capabilities of z/VM are still valid. These consist of a set of system operator commands for changing the hardware server's I/O configuration while the system continues to run, or for managing the hardware I/O configuration of all of the logical partitions in your server.

You now have the choice of either using these commands, or using HCM and HCD, to manage your I/O configuration. Note, however, that the use of HCM and HCD is incompatible with the original dynamic I/O configuration capabilities. You should select one method to use for the duration of any given IPL of your z/VM system.

4.2.8 Language Environment

Language Environment (LE) provides the runtime environment for programs written in C/C++, COBOL, or PL/I. Language Environment helps you create mixed-language applications and gives you a consistent method of accessing common, frequently-used services.

Language Environment consists of:

- Basic routines that support starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling conditions.
- Common library services, such as math services and date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- Language-specific portions of the runtime library. Because many language-specific routines call Language Environment services, behavior is consistent across languages.

4.2.9 OSA/SF

The Open Systems Adapter-Express (OSA-Express) and Open Systems Adapter Express2 (OSA-Express2) are integrated hardware features that allow the System z platform to provide industry-standard connectivity directly to clients on local area networks (LANs) and wide area networks (WANs).

The Open Systems Adapter Support Facility (OSA/SF) is a host-based tool supplied with z/VM that allows you to customize an OSA's modes of operation. You can access OSA/SF by a CMS user ID, by a REXX call to the OSA/SF API, or through a Java-based graphical user interface (GUI).

4.2.10 REXX/VM

REXX/VM contains the REXX/VM Interpreter, which processes the English-like Restructured Extended Executor (REXX) programming language. It also contains the z/VM implementation of the SAA® REXX programming language. REXX/VM provides a single source base for the REXX/VM Interpreter in the CMS and GCS components. The REXX/VM Interpreter exploits 31-bit addressing.

The REXX/VM Interpreter helps improve the productivity of your organization. Using REXX, you can write customized application programs and command procedures, tailor CMS commands, and create new XEDIT macros.

4.2.11 TSAF

TSAF, or Transparent Services Access Facility, provides communication services within a collection of VM systems without using VTAM. TSAF runs in a CMS virtual machine.

A group of up to eight VM systems that each have TSAF installed and running can form a TSAF collection. APPC/VM programs on one VM system in the TSAF collection can communicate with other APPC/VM programs on the other VM systems in the collection. The routing is transparent to the application programs. Communications between the applications proceed as if the applications were running on the same system.

4.2.12 VMSES/E

VMSES/E, or Virtual Machine Serviceability Enhancements Staged/Extended, helps you install z/VM and other VMSES/E-enabled products and apply code changes that correct or circumvent reported problems. VMSES/E handles both source code and object code.

VMSES/E can also help you define, build, and manage saved segments. The VMFSGMAP command provides a saved segment mapping interface that lets you modify saved segment definitions and view saved segment layouts prior to actually building them on your system.

The next section provides overviews of the optional features of z/VM.

4.2.13 DFSMS/VM

DFSMS/VM, or Data Facility Storage Management Subsystem for VM, or allows you to control your data and storage resources more efficiently. DFSMS/VM provides the following support.

Space management

DFSMS/VM improves DASD utilization by automatically managing space in SFS file pools. As the SFS administrator, DFSMS/VM allows you to:

- Convert SFS storage to DFSMS™-managed storage by assigning management classes to files and directories. Each management class tells DFSMS/VM how to treat its members in the course of its management of the file pool.
- Automatically manage files based on the criteria in each management class. This management may consist of deletion of files, automatic migration of files, or both.
- Migrate (or move) files from DFSMS-managed storage to DFSMS-owned storage by using the assigned management class. This function also compresses the data. The files can be automatically recalled when referenced (opened and browsed), or they can be explicitly recalled.

Minidisk management

Using DFSMS/VM for minidisk management allows you to check the integrity of CMS minidisks and move them from one location to another. DFSMS/VM helps you migrate CMS minidisks to new DASD quickly, efficiently, and with minimal impact to users.

Interactive Storage Management Facility (ISMF)

DFSMS/VM uses the ISMF to provide a consistent user interface for storage management tasks.

IBM Tape Library Dataserver Support

DFSMS/VM provides native VM support for the IBM 3494 and 3495 Tape Library Dataservers.

4.2.14 Directory Maintenance Facility for z/VM

The Directory Maintenance Facility for z/VM (DirMaint) provides efficient and secure interactive facilities for maintaining your z/VM system directory. Directory management is simplified by DirMaint's command interface and automated facilities. DirMaint provides a corresponding command for every z/VM directory statement, including Cross System Extensions (CSE) cluster directory statements. DirMaint's error checking ensures that only valid changes are made to the directory, and that only authorized personnel are able to make the requested changes.

DirMaint offers the following functionality:

- DirMaint operates as a CMS application and uses CMS interfaces for CMS and CP services. As a CMS application, DirMaint is not dependent on specific hardware, although it does verify that the device types specified in DirMaint commands are only those supported by the z/VM host.
- DirMaint functions are accomplished by two disconnected virtual machines equipped with an automatic restart facility. The use of virtual machines takes advantage of the inherent reliability, availability, and serviceability of the system architecture.
- Any transaction requiring the allocation or deallocation of minidisk extents can be handled automatically.
- All user-initiated transactions can be password-controlled and can be recorded for auditing purposes.
- Command authorization is controlled by assigning DirMaint commands to privileged command sets. Users may be authorized to issue commands from multiple command sets. DirMaint provides nine predefined command sets, but up to 36 sets are supported.
- User exit routines enable centralized directory maintenance of remote systems. Some exit routines also enable DirMaint to interact with other facilities, such as RACF.
- The open command structure allows you to replace all commands with your own user-written commands.
- An automated process for copying CMS minidisk files minimizes the possibility of human error. This process optionally formats the old (source) minidisk before returning it to the available minidisk pool.
- The integrity of CMS files is ensured by preventing new minidisk space from being inadvertently allocated over existing extents.
- DirMaint improves overall system efficiency by minimizing the number of DIRECTXA utility runs required. The update-in-place facility (DIAGNOSE code X'84') can be used to place many of the changes online immediately.

- System security is enhanced by providing the ability to enforce regular password changes. When changing the password, the user is required to enter the new password twice to guard against typographical errors.
- An additional level of security can be implemented by requiring that a password be entered for every user transaction. This is the default.

Note: The VM directory is described in more detail in 4.3, “VM Directory” on page 85.

4.2.15 Performance Toolkit for VM

The Performance Toolkit for VM, which is derived from the FCON/ESA program (5788-LGA), assists operators and systems programmers or analysts in the following areas:

- Operation of the system operator console in full screen mode
- Support for managing multiple VM systems
- Post-processing of VM history files
- Performance monitoring
- Serving data through a Web server for viewing with Web browsers
- Personal computer-based graphics
- TCP/IP performance reporting

In addition to analyzing VM performance data, the Performance Toolkit processes Linux performance data obtained from the Resource Management Facility (RMF™) Linux performance gatherer, rmfpms. The rmfpms application is available from the following site:

http://www.ibm.com/servers/eserver/zseries/zos/rmf/rmfhtmls/pmweb/pmlin.html

The Linux performance data obtained from RMF can be viewed and printed in a manner similar to the presentation of VM data.

4.2.16 RACF Security Server for z/VM

The RACF Security Server for z/VM, or Resource Access Control Facility, is a security tool that works together with existing functions in the z/VM base system to provide improved data security for an installation. RACF protects information by controlling access to it.

RACF also controls what you can do on the operating system and protects your resources. It provides this security by identifying and verifying users, authorizing

users to access protected resources, and recording and reporting access attempts.

To help each installation meet its unique security needs and objectives, RACF provides:

- Flexible control of access to protected resources
- The ability to store information for other products
- A choice of centralized or decentralized control profiles
- Transparency to end users
- Exits for installation-written routines

Your organization can define individuals and groups who use the system that RACF protects. A security administrator uses RACF to define a profile for each individual that identifies that person's user ID, password, and other information.

A *group* is a collection of individuals who have common needs and requirements. For example, a whole department may be defined as one group. Your organization can also define what authorities you have, or what authorities a group you belong to has. RACF controls what you can do on the system. Some individuals have a great degree of authority, while others have little authority. The degree of authority you are given is based on what you need to do your job.

In addition to defining user and group authorities, RACF protects resources. You can protect system resources and user resources. System resources include system minidisks, system SFS files and directories, certain VM events, and terminals. User resources include user minidisks and user SFS files and directories.

RACF stores all this information about users, groups, and resources in profiles. A *profile* is a record of RACF information that has been defined by the security administrator. There are user, group, and resource profiles.

Using the information in its profiles, RACF authorizes access to certain resources. RACF applies user attributes, group authorities, and resource authorities to control use of the system. The security administrator or someone in authority in your organization controls the information in your user profile, in group profiles, and in resource profiles. You, as an end user, control the information in profiles describing your own resources, such as your own minidisks. You can protect your data by setting up resource profiles. You can set up an access list in your resource profile to control who has read-access and who has write-access to your data.

In addition to uniquely identifying and authorizing users, RACF can record what users do on the system. It keeps track of what happens on the system so that an organization can monitor who is logged on to the system at any given time.

RACF reports if persons have attempted to perform unauthorized actions. For example, RACF can record when someone who does not have the proper authority tries to use or change your data. The security administrator can monitor these activities and generate reports.

4.2.17 RSCS Networking for z/VM

RSCS Networking for z/VM, or Remote Spooling Communication Subsystem, commonly referred to as RSCS, is a networking program that enables users on a z/VM system to send messages, files, commands, and jobs to other users within a network. RSCS connects nodes (systems, devices, and workstations) using links. These links allow data to be transferred between the nodes.

Running under the GCS component of z/VM, RSCS uses the spooling facilities of z/VM to store and retrieve data. z/VM handles data transfer within its system by means of spooling. RSCS extends the basic spooling capabilities of z/VM, handling data transfer between the z/VM system and outside sources. Data is stored on a spool after RSCS receives it and until RSCS can forward it to its destination. RSCS uses communications equipment to transfer data between the local z/VM system and other systems or remote locations

A node in an RSCS network is either a system node or a station node. A station node can originate and receive information. It can be a computer, a workstation, or a printer. A system node, however, must be a computer. In addition to originating and receiving information, system nodes can also relay information between two other nodes.

RSCS can communicate with system nodes that are running under the control of network job entry (NJE)-compatible subsystems, such as:

- JES2 or JES3
- RSCS
- VSE/POWER
- AS/400® Communications Utilities
- Products that provide NJE functions for Linux or AIX

RSCS can communicate with the following types of station nodes:

- ASCII printers or plotters
- Computers running under the control of a system that can provide a multileaving protocol
- IBM 3270 Information Display System Printers
- Line printer router (LPR) daemons and clients in a TCP/IP network
- Unsolicited File Transfer (UFT) daemons and clients in a TCP/IP network
- Workstations running under the control of remote job entry (RJE)

Each link in an RSCS network is associated with a programming routine, called a *driver*, that manages the transmission and reception of files, messages, and commands over the link. The way that a driver manages the data is called a *protocol*. All file transmission between networking nodes uses NJE protocol, 3270 printers use 3270 data streams, workstations use RJE protocol, and ASCII printers use data streams appropriate to that printer.

Systems Network Architecture (SNA) provides one set of protocols that governs communications on links. The method that RSCS uses for sending data to a node varies, depending on the type of connection used to establish the link. RSCS can support non-SNA (such as binary synchronous communication or channel-to-channel), SNA, and TCP/IP connections.

4.3 VM Directory

The user directory (USER DIRECT) is a file containing all the information about all of the system's guests. It is usually owned by the MAINT user. In larger installations add-on products, such as DIRMAINT, may convey ownership of the source directory to another user.

When a user definition is created, we need to specify the CP privilege classes, Minimum/Maximum Virtual memory that a user can access, and the mode in which the VM is going to run (for example, ESA/XA).

Example 4-1 shows an entry for the user TCPIP in the user directory.

Example 4-1 User direct entry for user TCPIP

```
USER TCPIP TCPIP 32M 128M ABG 1
INCLUDE TCPCMSU    2
  OPTION QUICKDSP SVMSTAT MAXCONN 1024 DIAG98 APPLMON    3
  SHARE RELATIVE 3000         4
  IUCV ALLOW                  5
  IUCV ANY PRIORITY
  IUCV *CCS PRIORITY MSGLIMIT 255
  IUCV *VSWITCH MSGLIMIT 65535
  LINK 5VMTCP30 491 491 RR    6
  LINK 5VMTCP30 492 492 RR    6
  LINK TCPMAINT 591 591 RR    6
  LINK TCPMAINT 592 592 RR    6
  LINK TCPMAINT 198 198 RR    6
  MDISK 191 3390 3135 005 LX6W01 MR READPASS WRITPASS MULTPASS 7
```

The entries that make up the z/VM guest definition in USER DIRECT have the following meanings:

1. The USER line sets the user ID of this virtual machine, which is TCPIP. The password is also TCPIP. The next value represents the CP classes of this user ID (in our example A, B, and G).

   This machine is defined to have a default memory storage of 32 MB when it logs in. If the user were to use the DEFINE STORAGE command, the user would be allowed to increase the memory allocation to a maximum of 128 MB space. (Note that this is a disruptive change.)
2. INCLUDE TCPCMSU will include the TCPCMSU profile, shown in Example 4-2.
3. The OPTION statement to set, for example, the Quickdispatch option to ensure that this z/VM user has immediate access to system resources.
4. The SHARE RELATIVE option specifies that this user is to receive a target minimum relative share of nnnnn. The amount of scheduled system resources available to relative share users is the total of resources available less the amount allocated to absolute share users.
5. IUCV statements to use the Inter User Communication Vehicle (IUCV).
6. With LINK, a minidisk of another user ID will be linked.
7. Each MDISK statement creates a minidisk for a z/VM guest.

If certain directory control statements are repeated for several users, you can make use of directory profiles to save space in the directory. Because you can potentially create many hundreds of Linux z/VM userids on a single z/VM image, you can create a profile to define the things that many user IDs have in common. The profile we included in our TCPIP userid is shown in Example 4-2.

Example 4-2 Include profile TCPCMSU

```
PROFILE TCPCMSU
  IPL CMS                    1
  MACH XA                    2
  SPOOL 000C 2540 READER *   3
  SPOOL 000D 2540 PUNCH A    3
  SPOOL 000E 1403 A          3
  CONSOLE 009 3215 T         4
  LINK MAINT 0190 0190 RR    5
  LINK MAINT 019D 019D RR
  LINK MAINT 019E 019E RR
  LINK MAINT 0402 0402 RR
  LINK MAINT 0401 0401 RR
  LINK MAINT 0405 0405 RR
```

The following statements, displayed in the example, are in the TCPCMSU include profile:

1. During logon load (IPL), Conversational Monitor System (CMS).
2. MACH XA sets the architecture in effect for this virtual machine. When you select an architecture mode, your virtual machine must obey that particular modes architecture conventions.
3. The SPOOL statements set control options for the virtual spool devices. The options you select modify operational functions associated with your virtual reader, printer, punch, or console. These options also control the disposition of files after they have been processed.
4. The type of virtual I/O support CP provides for your display and the virtual address is defined.
5. With LINK, a minidisk of another user ID will be linked.

4.4 How to log on to z/VM

Previously, we explained the basic concepts and discussed the different components of z/VM. Here, we explain how to log on to a z/VM system and show you the layout of a z/VM screen. You will also learn some basic z/VM commands.

The software needed to connect to a mainframe varies.

- If you are running a distribution of GNU/Linux, skip ahead to the X-Windows 3270 emulator.
- If you are using a Microsoft Windows operating system, you will want to configure your IBM Personal Communications software with the IP address of the z/VM system to which you are connecting.

4.4.1 Connecting with IBM Personal Communications

Go to the Communication pull-down menu and select **Configure**, as shown in Figure 4-1 on page 88.

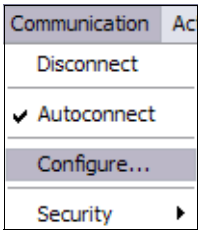


Figure 4-1 Open the configuration screen of Personal Communications

At the screen shown in Figure 4-2, select **Link Parameter**.

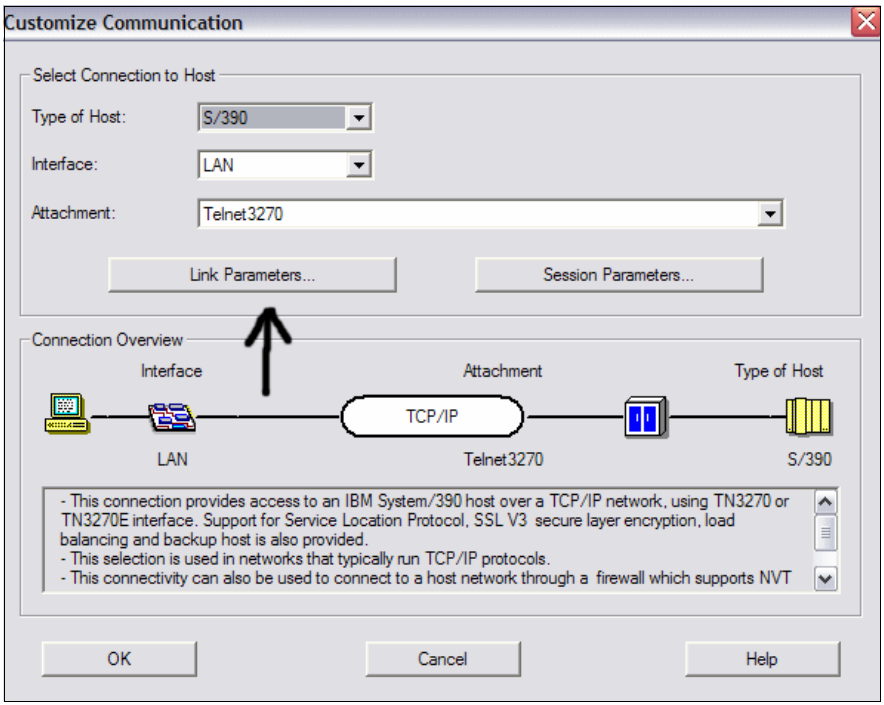


Figure 4-2 Customize screen of Personal Communications

On the screen shown in Figure 4-3 on page 89, enter the IP address of your z/VM system in the field primary host name or IP address, then press **OK**.

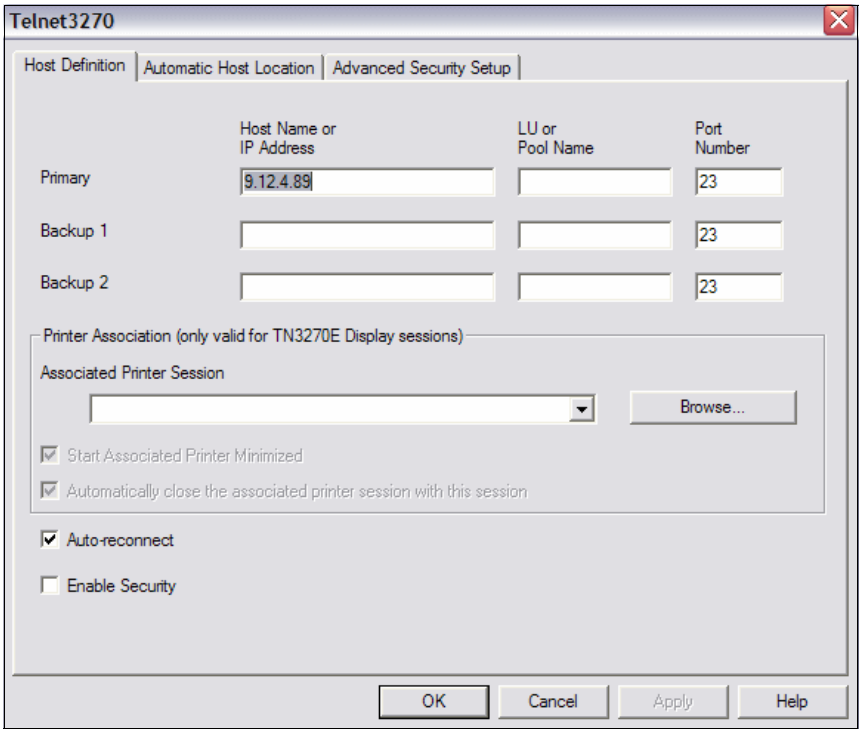


Figure 4-3 Personal Communications customizing

The customize screen of Personal Communications shown in Figure 4-2 on page 88 will appear again; select **OK**.

On the screen shown in Figure 4-4, confirm your customization by selecting **OK**.

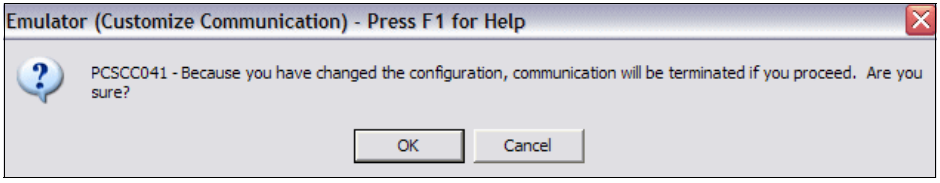


Figure 4-4 Confirm your changes

Now Personal Communications establishes a connection to the z/VM system, and the z/VM logon screen shown in Figure 4-5 on page 90 will display.

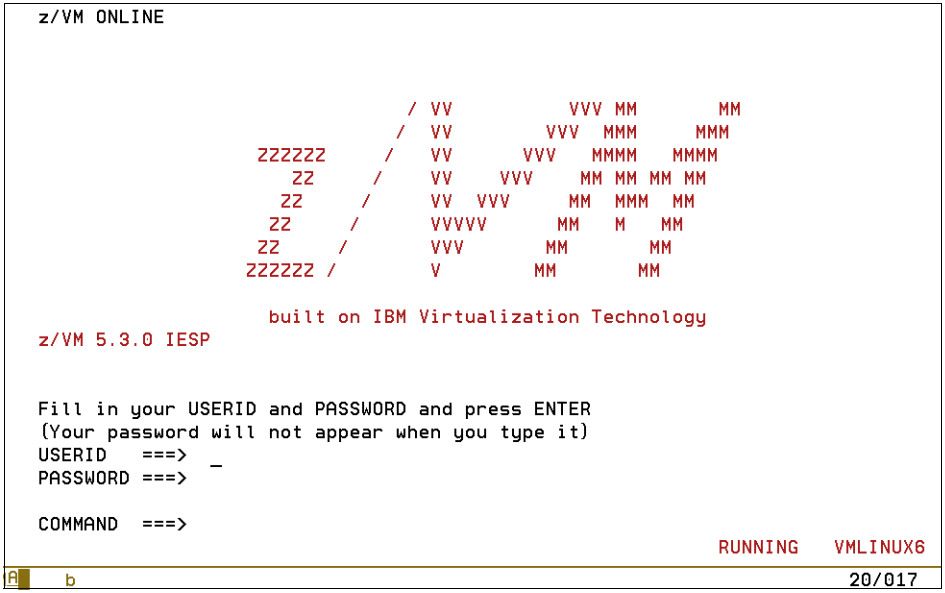


Figure 4-5 z/VM logon screen

4.4.2 Connecting with x3270

If you are using a Linux distribution, the process to connect is simpler. After installing your distribution's x3270 software package, launch x3270 from the command line or menu.

The initial display of the x3270 program will look similar to Figure 4-6 on page 91.

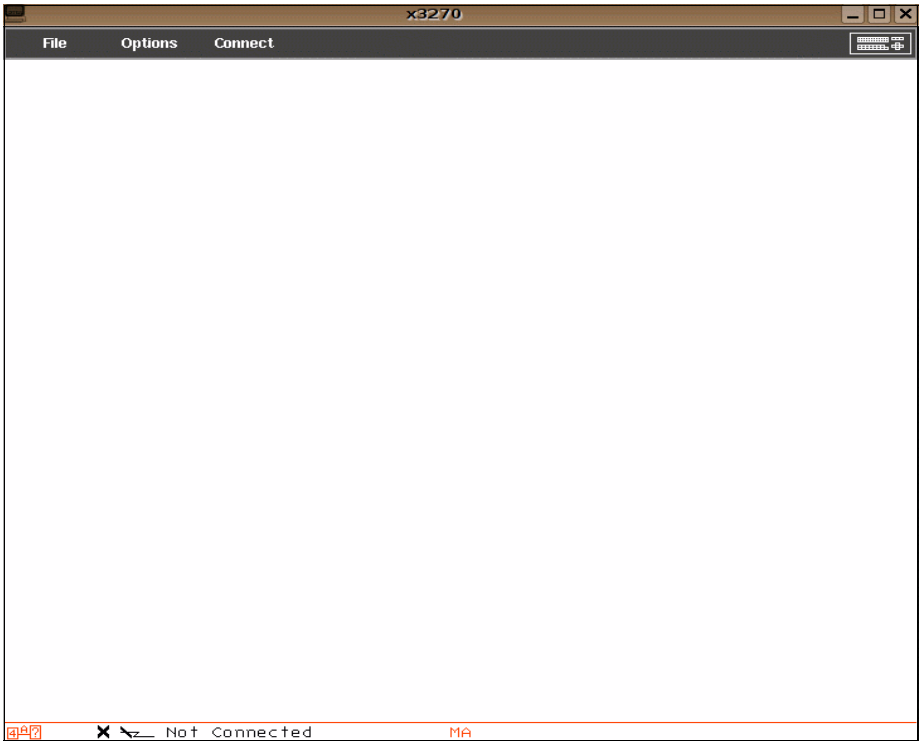


Figure 4-6 x3270 application upon initial startup

From the x3270 main window, click the **Connect** menu item and you will be prompted with a connection dialog as shown in Figure 4-7 on page 92. Enter the IP address or hostname of the virtual machine you want to connect to.

Most often, as a beginner, this information will have been provided to you by a system programmer. Future attempts to connect to the server are far easier. Hosts added with the new connection dialog are remembered and displayed in the Connect menu.

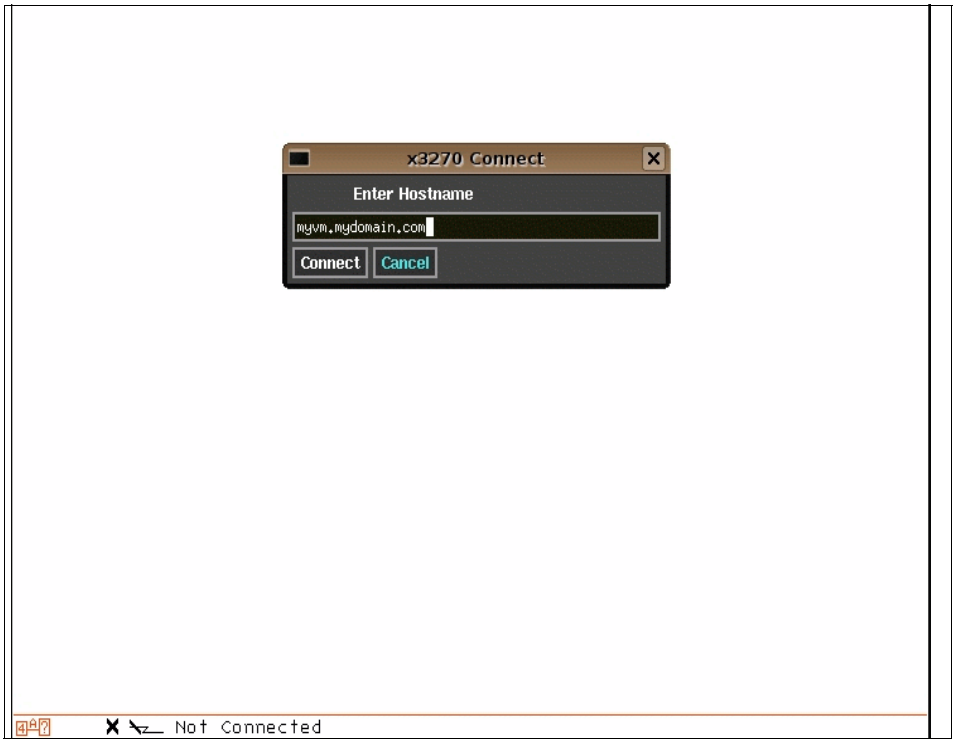


Figure 4-7 The Hostname prompt for a new x3270 connection

Your server will be added to the list in the Connect menu; see Figure on page 93.

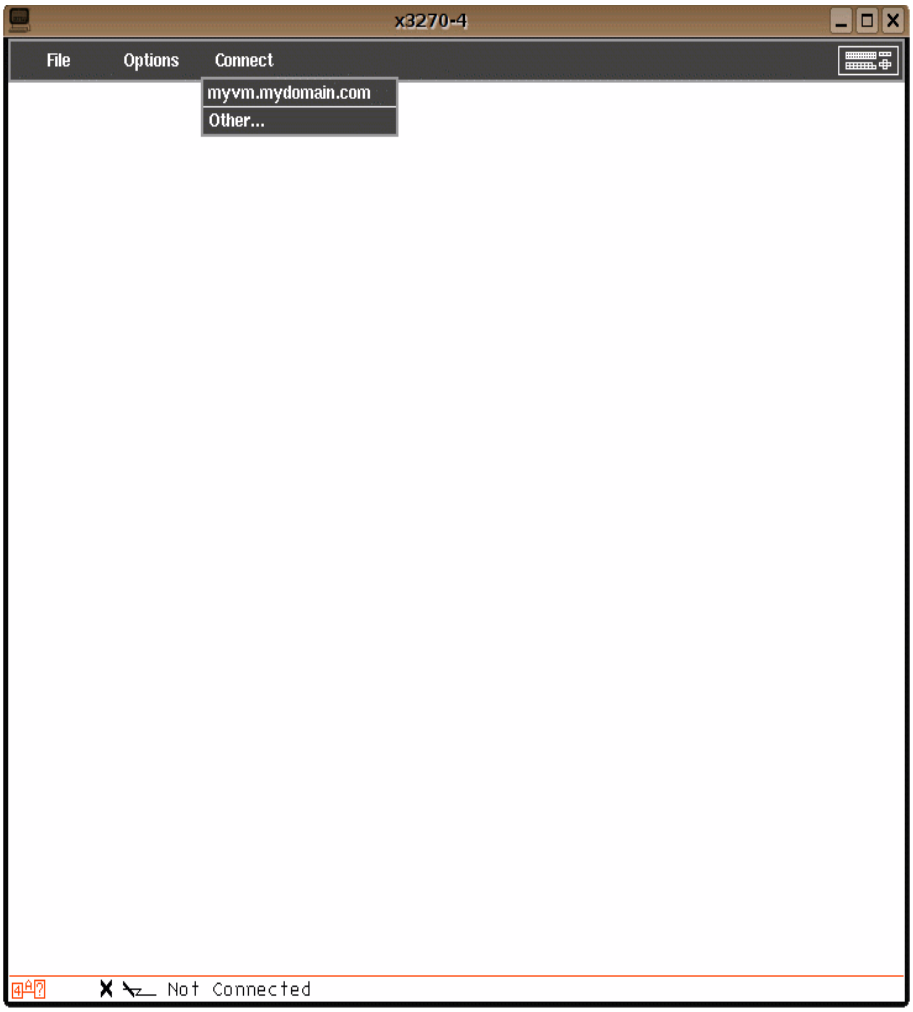


Figure 4-8 Your server is added to the list in the Connect menu

Your connection will be established if you see the z/VM logon prompt (see Figure 4-5 on page 90) using the IBM Personal Communications tool. The process of logging on is the same from this point on.

Note: To alter the information stored for a previously saved connection, edit the file .3270connect in your home directory.

4.4.3 Logging on

To log on to the z/VM system, type in your user ID and password in the corresponding areas on the z/VM logon screen (shown in Figure 4-5 on page 90), and then press **Enter**.

You can also log on by positioning the cursor in the command line, typing `LOGON` followed by your user ID, then pressing Enter as shown in Figure 4-9.

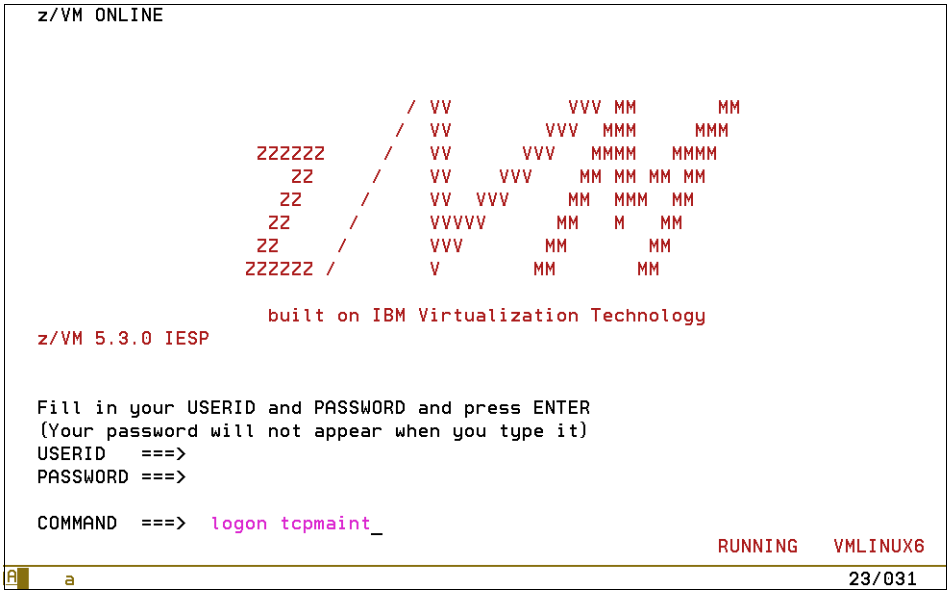


Figure 4-9 Logging on using the command line

The system prompts you for the password (see Example 4-3).

Example 4-3 Prompt for the password

```
LOGON MAINT
ENTER PASSWORD  (IT WILL NOT APPEAR WHEN TYPED):
```

After you are logged in, you may be presented with the CP READ prompt. Press **Enter** again and the CP will load CMS.

CMS then executes the PROFILE EXEC scripts of the z/VM user ID, if any exists. In the PROFILE EXEC, you can define settings for your z/VM user ID, such as define PF keys, link to MDISKs or IPL a guest operating system like Linux or z/VSE.

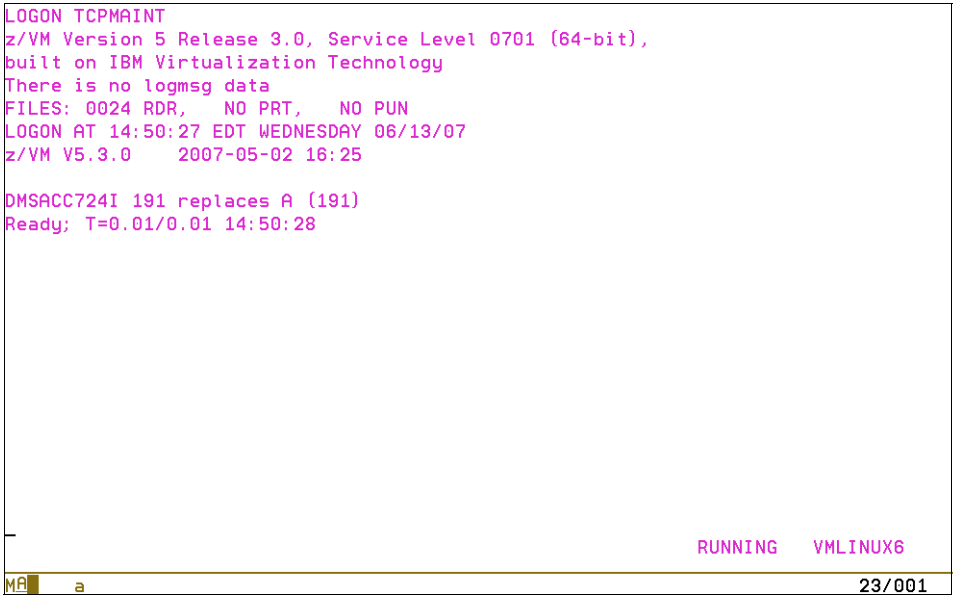


Figure 4-10 z/VM screen after logon

At this point, you are logged on to your z/VM virtual machine. Before discussing some basic commands that can be used to check your virtual machine, we cover some introductory information about VM commands.

Here we use the term “command” generically; it refers to both CP commands and CP utilities. z/VM uses command languages to correspond to the two environments it creates: Control Program and virtual machine.

Use the Control Program (CP) command language when:

- You are a z/VM system operator and you want to control the resources of the real machine located in your computer room.
- You are a virtual machine user and you want to control your virtual machine’s configuration and environment.

Use a virtual machine command language when:

- You are communicating with the operating system you loaded into your virtual machine.
  - To perform production or test work, load your virtual machine with one of the operating systems supported by the z/VM system. Your virtual machine command language is the command language of the operating system you load. This command language is described in the library that documents that particular operating system.

– To perform service, installation, and maintenance tasks, along with editing and text creation, communicating with others, and problem solving, load your virtual machine with the conversational monitor system (CMS). CMS is a single user, conversational operating system.

You can use CP commands in the following situations:

- Your virtual machine is in the control program (CP) command environment.

  Your virtual machine is in the CP environment when you log on to z/VM and CP READ is displayed in the lower right corner of the screen.

  On a line-mode ASCII device, there is no status area to display CP READ, so CP is displayed in the output area.

- You press the break key while in full-screen mode before entering a command.

  The break key may be PA1, the VM default break key, or another key that you have defined as the break key using the TERMINAL BRKKEY command. Also, the break key may be totally disabled by some application programs or when in the protected application environment.

- You are in a virtual machine command environment, not running in full-screen mode, and enter the #CP command (and # is your logical line end character).
- You are in the CMS virtual machine environment and enter the CP command.
- You are in the CMS virtual machine environment and have the IMPCP function set ON.

To determine the current command environment on a 3270 device, look at the status area in the lower right corner of the display screen. `CP READ` indicates the CP environment, `VM READ` indicates the virtual machine environment.

If you are running CMS in your virtual machine, `VM READ` indicates the CMS environment. When `RUNNING` appears in the status area, enter a null input line to determine your environment. To enter a null line, press Enter but do not enter any data. When you enter the null line, the status area displays either `CP READ` or `VM READ`.

Also, if you are in a read state, in either the CP command environment (with RUN set OFF) or the CMS command environment, and you enter a null line, the system responds with the name of your command environment: CP or CMS in the system output area.

You enter CP commands using any combination of upper case and lower case letters. When you have typed the command and its operands, press Enter to process the command.

4.5 Working in a 3270 terminal

Previously, we explained how to log on to the z/VM system, and you saw the response from the system. Now we discuss what you need to know when working with a 3270 terminal on a z/VM system.

Figure 4-11 displays the layout of the 3270 screen.

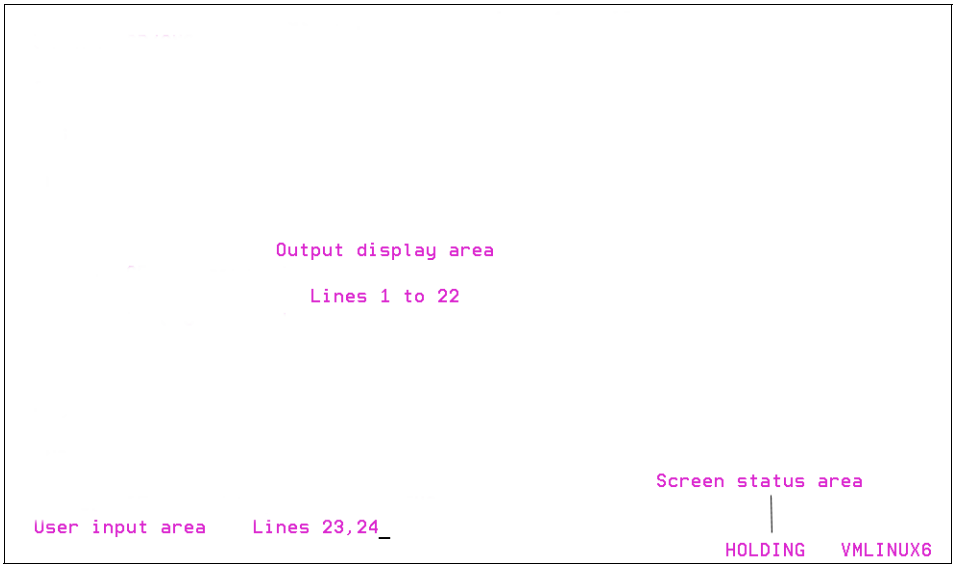


Figure 4-11 Layout of the 3270 screen

The screen is divided into three areas: the output display area, the user input area, and the screen status area, as described in Table 4-1.

Table 4-1 Screen display areas

Area on screen	Description
Output display area	► Consists of lines 1 to 22. ► All messages will be displayed here. ► The commands you entered will be displayed here.
User input area	► Consists of last two lines, excluding the right-most 21 character positions of the last line. ► Commands can be entered here. ► Cursor control keys, DELETE key, INSERT Key and logical text editing characters can be used to alter data in this area. ► After pressing Enter, CP redisplays the data entered here in the output display area.

Area on screen	Description
Screen status area	► Consists of the right-most 21 character positions of the last line. ► Screen status indicators appear here. ► Indicates whether the current environment is CP or VM. ► The status displayed here can be one of the following: – `CP READ` - CP issued a read request to the display and is waiting for the user to enter something before it can continue processing. – `VM READ` - The virtual machine is awaiting a response from the user before it can continue processing. – `RUNNING` - CP or VM is either ready to accept commands, or processing an already executed command – `MORE ...` - Indicates that the output display area is full and there is more data to display.[1] • CP waits for 60 seconds (default) and then displays the next screen automatically. • Use the PA2 key (3270) to see the next screen without waiting 60 seconds. • Use the Enter key (3270) to keep the current screen (status changes to `HOLDING`). – `HOLDING` - Indicates that user pressed Enter (3270) in response to `MORE`. – `NOT ACCEPTED` - Indicates that the previous input was not accepted (CP locks keyboard for 3 seconds).
	[1]Use SET FULLSCREEN ON from CMS if the scroll back option is needed. This is disabled by default. (Note that this works only for CMS.)

4.5.1 Keyboard mapping

Note that the Enter key in your 3270 terminal session in different from the Enter key you use when working with your personal computer. Both of the methods of connecting to a 3270 session we have shown you provide mechanisms to alter the keyboard mapping to suit your preference.

Figure 4-12 on page 99 shows the keyboard layout and the position of the Enter key.

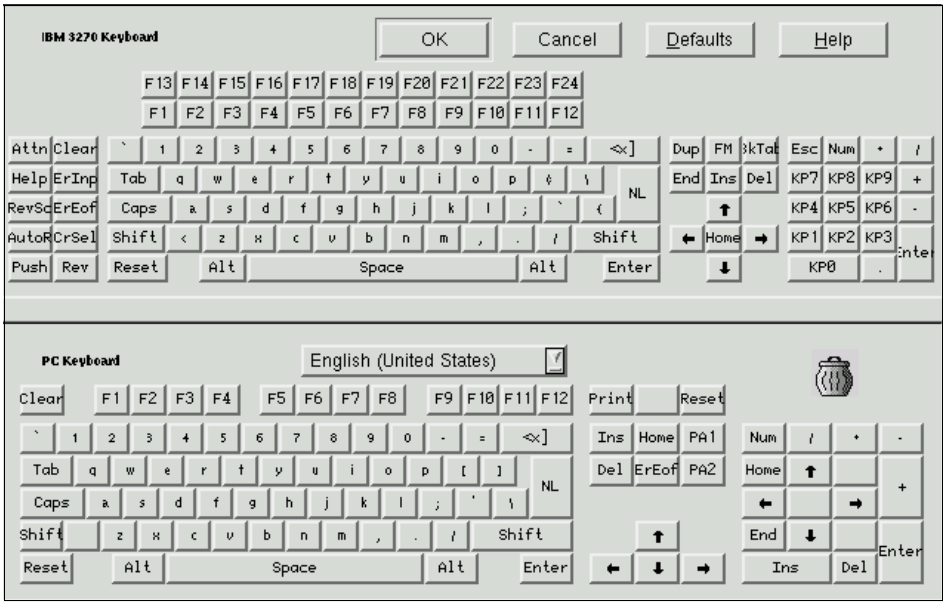


Figure 4-12 Keyboard layout

4.6 Session management

Your session begins when you log on to the system, and it ends when you log off. Here we describe the processes of logging in and logging out, as well as a few other actions you can perform during a session.

4.6.1 Logging on

For an overview of the process of logging on, see 4.4, “How to log on to z/VM” on page 87. Here, we explain how to log on in detail.

The logon screen contains three basic fields:

- Username
- Password
- Command line

If you want to log on normally, enter your user name and password in the Username and Password fields.

By using the command line, however, you can issue the CP **LOGON** command, which has many parameters that can customize how you log on to the system.

To use the command line, press the Tab key until your cursor is positioned on the command line. (Or you can use the arrow keys or your mouse to manually move the cursor to the command line.)

After you are on the command line, issue the LOGON command. The simplest use of the LOGON command is shown in Example 4-4.

Example 4-4 Simple use of LOGON command

```
LOGON TUX1
```

This command will prompt you for a password. If the password that you enter is correct, the system will log on the user TUX1 just as if you had entered the user name and password in the normal Username and Password fields.

Notes:

- If you type a user name or password incorrectly on the login screen, then the normal input fields will disappear and you will be forced to use the **LOGON** command from the command line if you want to try to log in again. Therefore, it is important to know how to do this in case you make a typo the first time you try logging in.
- The CP **LOGIN** command is identical to the CP **LOGON** command. Both exist because some people prefer to use the term “log in” and others prefer “log on”.

4.6.2 Disconnecting

CP provides a very useful feature known as disconnecting. You may *disconnect* your terminal from your virtual machine without stopping or interfering with the guest operating system running in your virtual machine in any way. This effectively severs your actual session with the virtual machine (like logging off), but the virtual machine and its guest operating system will continue to run. You disconnect by using the **DISC** command (for disconnect); see Example 4-5.

Example 4-5 Disconnecting from a guest

```
disc
DISCONNECT AT 11:34:15 EDT WEDNESDAY 06/13/07

Press enter or clear key to continue
```

You can use the **QUERY NAMES** command to tell which guests are disconnected. As shown in Example 4-6, disconnected guests have DSC displayed next to their names.

Example 4-6 Querying disconnected guests

```
QUERY NAMES

EDI      -LOOO5, DIRMAINT - DSC , TCPIP    - DSC , RSCS     - DSC
PVM      - DSC , DATAMOVE - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC
VMSERVR  - DSC , VMSERVU  - DSC , VMSERVS  - DSC , GCS      - DSC
OPERSYMP - DSC , DISKACNT - DSC , EREP     - DSC , OPERATOR - DSC
CLIVE    -LOOO8, MAINT    -LOOO4, EDI2     -LOOO6, JASON    -LOOO7
VSM     - TCPIP
```

When disconnecting from a guest, it is important to note your virtual machine's RUN parameter. If the RUN parameter is set to OFF when you disconnect (or reconnect), then your guest operating system may be suspended, in some cases.

To ensure that this does not occur, ensure that RUN is set to ON before you disconnect. You can do this by using the **SET** command as shown in Example 4-7.

Example 4-7 Setting the RUN parameter to ON

```
SET RUN ON
```

4.6.3 Reconnecting

To reconnect to a disconnected virtual machine, perform a normal logon and you will be reconnected. No special action is required.

4.6.4 Stealing a virtual machine session

If you ever find yourself in a situation where you need to log on to a virtual machine but another user is already logged on, you will be presented with the output shown in Example 4-8.

Example 4-8 Logon failed because guest is already logged in

```
LOGON TUX1
HCPLGA054E Already logged on LDEV LOOO8
```

In this case, you may perform an action known as "stealing" the session by passing the **HERE** parameter to the **LOGON** command, as shown in Example 4-9.

Example 4-9 Stealing a virtual machine session

```
LOGON TUX1 HERE
RECONNECTED AT 11:48:17 EDT WEDNESDAY 06/13/07
```

As shown in Example 4-9 we have stolen the session from the other terminal connected to TUX1. What actually happens is that CP disconnects the other user and then immediately connects the user doing the steal. Anything that was on the screen before the disconnect will appear on your screen after the reconnect.

4.6.5 Logging out

When you are finished running your virtual machine, you can log out by using the **LOGOFF** command. This will cause your virtual machine to go away and it will cease to exist until you log on again, at which time it will be recreated by CP. Logging off will immediately stop any guest operating system that may be running.

4.7 Exercises

To help test your understanding of the material in this chapter, answer the following questions:

1. Which components of z/VM do you know? List at least five components.
2. What are the main responsibilities of a z/VM system programmer?

5

Control Program for new users

This chapter introduces the z/VM Control Program (CP) and discusses what a new user needs to know in order to understand and operate CP for the purposes of managing a z/VM virtual machine.

Objectives

After completing this chapter, you will be able to:

- Become familiar with the concepts of the Control Program and virtual machines
- Interact with CP by using the command line interface
- Understand basic CP resources such as processors and main memory
- Become familiar with various virtual devices and the commands needed to manage them
- Manage a terminal and login session

© Copyright IBM Corp. 2007. All rights reserved.

5.1 Introduction to the Control Program (CP)

z/VM is comprised of two primary components: CP and CMS. In this section, we explain the concept and function of CP, which stands for Control Program.

The Control Program (CP) is the operating system that underlies all of z/VM. CP is responsible for virtualizing your z/Series machine's real hardware, and allowing many virtual machines to simultaneously share the hardware resource.

CP also handles the creation and management of virtual machines. It can be considered the operating system component of z/VM because it is responsible for managing real devices and resources and sharing them among various tasks and users that need them. CP is analogous to the Linux kernel in a GNU/Linux operating system.

As a resource manager, CP can provide you with a set of virtual machines which can run any operating system that would ordinarily run on your z/Series hardware. Without CP you could only be running one operating system on your hardware at any given time (logical partitioning aside). CP allows you to have multiple virtual machines (also known as “guests”), and each one can be running an instance of an operating system simultaneously. As shown in Figure 5-1, CP allows you to run many Linux servers simultaneously on the same piece of hardware.

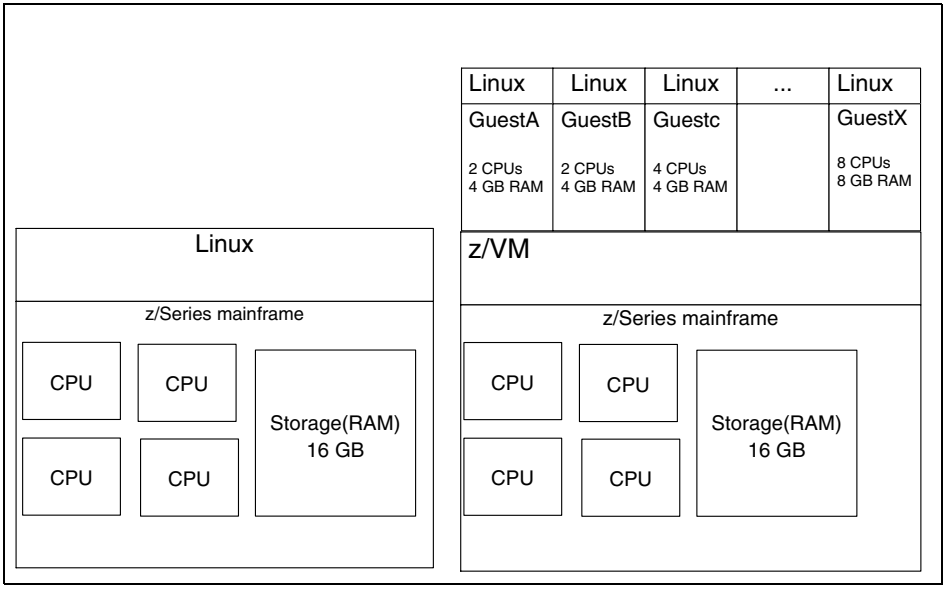


Figure 5-1 CP executing many guest operating systems

Notice in the figure that, even though we only have 4 CPUs and 16 GB of RAM, our guest virtual machines are collectively defined to have 16 CPUs and 20 GB of RAM. This is because the CPUs and RAM in a guest's operating system are virtual. You can create a guest that has 64 CPUs if you prefer, and CP will take turns executing tasks on behalf of all 64 virtual CPUs on all of the available real processors. The CP components that handle this task are known as the *scheduler* and the *dispatcher*. These components are very much like the process scheduler of Microsoft Windows or GNU/Linux, except they schedule entire virtual machines for execution instead of simply processes.

Note that the ability to overcommit resources like this only works because an operating system typically does not need 100% of the resources allocated to it all the time that it is running. Sometimes that operating system will be idle, and when it is, another operating system can use the hardware resources.

Virtual machines are defined in a text file known as the *user directory*. For each virtual machine on the system, there is a section in the directory that describes that virtual machine. One of these descriptions is known as a *directory entry*.

The directory entry for a guest details its guest name, its privileges to execute CP commands, the number of CPUs it has, the amount of memory it has, as well as information on which virtual devices are defined for this particular VM. It is the job of the system administrator to set up and maintain the user directory, and general users typically do not have access to it.

5.1.1 What CP is not

It is important to note that CP is not a full-fledged operating environment like GNU/Linux or Microsoft Windows. CP alone does not understand the concept of "files". It does not provide convenient methods of loading and running different programs, and it does not allow users to run meaningful tasks such as Web serving, text editing, or data processing.

Instead, CP acts only as a resource manager. It is the job of the *guest operating system* (CMS, Linux, z/OS, and so on) to use the resources provided by CP to complete meaningful work.

5.1.2 CP modes of execution

At any time, your virtual machine can be in one of two basic states of execution.

- Running a guest operating system (guest mode)
- Not running a guest operating system (CP mode)

When a guest operating system is running, it is controlling the devices and resources of your virtual machine. Any commands you issue to the virtual machine through your terminal command line will likely be interpreted by the guest operating system and not CP.

However, when your virtual machine is *not* running a guest operating system, then CP has control and your processors (as well as your other virtual devices) are generally sitting idle and not consuming real machine resources. In this state, any commands issued to the virtual machine through your terminal command line will be interpreted by CP and not a guest operating system.

Before you start a guest operating system, you are in CP mode. When you start a guest operating system, you are in guest mode. If you shut down your guest operating system, you will be returned to CP mode. Your virtual machine can only be in one of these modes at any instance in time.

However, you are free to switch between these modes at will even when your guest operating system is running. If you switch to CP mode while a guest operating system is running, that operating system will be frozen and will not run again until you leave CP mode and go back to guest mode. At that point the guest operating system will continue running as if it had never been interrupted.

5.1.3 CP commands

You will primarily interact with CP via a command line interface. You issue commands to CP by typing them on the command line and pressing Enter to submit them for processing.

CP commands are always between1 and 12 characters in length, and are case-insensitive. A command will often allow the specification of parameters, all of which are separated by a space (as in most command line environments).

Many commands also have abbreviations which are shorter to type but more cryptic. For example, the **QUERY NAMES** command can be shortened to **Q N**. Executing either of the two commands will accomplish the same task.

This book always uses the complete command and not the abbreviation. If you are interested in knowing the abbreviation for a command, refer to the **HELP command** or to *CP Commands and Utilities Reference*, SC24-6081.

5.2 Learning about the system

Before you can give commands to CP, you need to log in to a virtual machine via a 3270 terminal emulator, as explained in 4.4, “How to log on to z/VM” on page 87. If you have access to a 3270 terminal emulator and a virtual machine, it is recommended that you follow along with the commands and examples in the rest of this chapter.

5.2.1 Getting to CP mode

After logging in, any number of things could be going on, depending on the guest operating system your virtual machine is running. Because we are only interested in CP at this point, in our scenario we are going to shut down the guest operating system that is running and get into CP mode so we can issue commands directly to CP.

In your case, it is likely that your virtual machine will be running the CMS operating system. You can tell if you are running CMS by the output you get when you log in. If you see output similar to Example 5-1, then you are running CMS. The last line (starting with z/VM V5.3.0) is the first line of output from CMS.

Example 5-1 Output seen when running CMS at logon

```
z/VM Version 5 Release 3.0, Service Level 0701 (64-bit),
built on IBM Virtualization Technology
There is no logmsg data
FILES: 0001 RDR,   NO PRT,   NO PUN
LOGON AT 09:54:43 EDT MONDAY 06/04/07
z/VM V5.3.0    2007-05-02 16:25
```

If no output appears you might still be running CMS, so enter the **QUERY CMSLEVEL** command to check further. If you see output similar to Example 5-2, then you *are* running CMS.

Example 5-2 Output from QUERY CMSLEVEL command

```
QUERY CMSLEVEL
CMS Level 23, Service Level 701
Ready; T=0.01/0.01 09:57:17
```

Important: To stop CMS and get into CP mode, you can issue the **#CP SYSTEM CLEAR** command. Note, however, that if you are *not* running CMS, then this command will still probably work—but it may damage your guest operating system. Issuing a **SYSTEM CLEAR** command is the equivalent of pulling your desktop personal computer's power cord out of the wall socket. It halts the system immediately and does not give the operating system a chance to finish what it is doing and perform cleanup tasks. And if the operating system is engaged in writing a file when this happens, you could lose the contents of the file or even permanently corrupt the file system.

Therefore, if you are *not* running CMS, then consult your guest operating system's manual for proper shutdown instructions.

Example 5-3 Output from #CP SYSTEM CLEAR command

```
#CP SYSTEM CLEAR
Storage cleared - system reset.
```

This command clears your virtual machine's memory and stops and resets all of its virtual processors.

If you are truly in CP mode and not running a guest operating system, then you should be able to issue the **BEGIN** command and see the output displayed in Example 5-4.

Example 5-4 Output of BEGIN command when not running a guest operating system

```
BEGIN
HCPGIR453W CP entered; program interrupt loop
```

If different content is displayed, then chances are you are still running a guest operating system.

5.2.2 Examining your virtual machine

CP has many commands that involve getting information about your virtual machine and defining its virtual hardware configuration. Here we examine some of the commands you can use to get information about your virtual machine.

At this point, we focus on the **QUERY** command. **QUERY** will provide a great deal of useful information about your virtual machine and the system it is running on. It takes several arguments, most of which have sub-arguments of their own. We will only look at a subset of the capabilities of the query command.

CP version

The **QUERY CPLEVEL** command will tell you which version (version is commonly referred to as “level” in z/VM terminology) of CP you are running; see Example 5-5.

Example 5-5 Output from QUERY CPLEVEL command

```
QUERY CPLEVEL
z/VM Version 5 Release 3.0, service level 0701 (64-bit)
Generated at 05/02/07 16:28:04 EDT
IPL at 05/03/07 13:06:26 EDT
```

Note: The **QUERY CPLEVEL** command is run when you log on to your virtual machine.

Your guest name

If you just logged in, then you probably know what your guest name is (a *guest name* is the uniquely identifying name given to your virtual machine). But in case you have more than one terminal open at the same time and forget which terminal goes with which guest, you use can use the CP command **QUERY USERID** to find out the guest name; see Example 5-6.

Example 5-6 Output from the QUERY USERID command

```
QUERY USERID
TUX1    AT VMLINUX6
```

The **QUERY USERID** command also tells you which “node” you are on. The term *node* refers to the instance of the z/VM operating system that your guest is running on. In this case, VMLINUX6 is the name given to the z/VM instance that the TUX1 guest is running on. When your system administrator installed z/VM, the administrator set the node name.

Note: The terms “virtual machine”, “guest”, and “user” all refer to a virtual machine. We use these terms interchangeably throughout this book.

Your virtual machine's resources

Your virtual machine (like any computer) is made up of CPU, memory, and devices. The **QUERY** command can show you basic information about your virtual machine, as well as information about these three different types of resources.

You can use the **QUERY VIRTUAL CPUS** command to display which virtual CPUs are defined for your virtual machine; see Example 5-7 on page 110.

Example 5-7 Output from the QUERY VIRTUAL CPUS command

```
QUERY VIRTUAL CPUS
CPU 00  ID  FF02991E20948000 (BASE) CP   CPUAFF ON
```

Example 5-7 shows there is one virtual CPU. If you have more than one CPU, then your output will include an additional line for each additional CPU that you have. For more information about CPUs, refer to 5.4, “Your virtual machine's virtual devices” on page 119.

You can use the **QUERY VIRTUAL STORAGE** command to display how much storage (or RAM) your virtual machine has. (z/VM refers to RAM as “storage”. We use the term storage to refer to RAM throughout this book, unless otherwise specified.)

The output in Example 5-8 shows that our virtual machine has 32 megabytes of storage. The amount of storage may be given in Kilobytes (K), megabytes (M), gigabytes (G) or terabytes (T).

Example 5-8 Output from the QUERY VIRTUAL STORAGE command

```
QUERY VIRTUAL STORAGE
STORAGE = 32M
```

You can use the **QUERY VIRTUAL ALL** command to display a list of all of the devices on your virtual machine; see Example 5-9.

Example 5-9 Output from the QUERY VIRTUAL ALL command

```
QUERY VIRTUAL ALL
STORAGE = 32M
XSTORE = none
CPU 00  ID  FF02991E20948000 (BASE) CP   CPUAFF ON
No AP Crypto Queues are available
CONS 0009 ON LDEV L0007   TERM STOP  HOST TCPIP    FROM 10.0.0.1
   0009 CL T NOCONT NOHOLD COPY 001    READY FORM STANDARD
   0009 TO TUX1     PRT DIST TUX1      FLASHC 000 DEST OFF
   0009 FLASH       CHAR       MDFY       0 FCB       LPP OFF
   0009 3215   NOEOF CLOSED    NOKEEP NOMSG NONAME
   0009 SUBCHANNEL = 0001
RDR  000C CL * NOCONT NOHOLD   EOF       READY
   000C 2540         CLOSED    NOKEEP NORESCAN  SUBCHANNEL = 0002
PUN  000D CL A NOCONT NOHOLD COPY 001    READY FORM STANDARD
   000D TO TUX1     PUN DIST TUX1      DEST OFF
   000D FLASH      000 CHAR       MDFY       0 FCB
   000D 2540   NOEOF CLOSED    NOKEEP NOMSG NONAME
   000D SUBCHANNEL = 0003
PRT  000E CL A NOCONT NOHOLD COPY 001    READY FORM STANDARD
```

```
   000E TO TUX1     PRT DIST TUX1      FLASHC 000 DEST OFF
   000E FLASH       CHAR       MDFY       0 FCB       LPP OFF
   000E 1403   NOEOF CLOSED    NOKEEP NOMSG NONAME
   000E SUBCHANNEL = 0004
DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
```

Each virtual device in your virtual machine has a virtual device number (commonly referred to as a *virtual device address*) that is specific to your virtual machine. In Example 5-9 on page 110, 191 is the virtual device number for the minidisk on DASD pack (disk) labeled DKCD37. The number 0009 is the virtual console device that is used by your terminal session. We discuss this topic in greater detail in 5.4, “Your virtual machine's virtual devices” on page 119.

But the point here is to recognize that there are a variety of virtual devices, including some DASD packs, a punch card reader, and a printer. You may also have other devices, depending on how your virtual machine is defined.

Privilege classes

CP uses what are known as *privilege classes* to regulate the types of things that virtual machines are allowed to do. Privilege classes are conceptually similar to “permissions: in Microsoft Windows and GNU/Linux environments.

When your system administrator created your virtual machine, the administrator assigned you a certain subset of the privilege classes available on the system. Each privilege class is denoted by either a single letter or a single number, each one defining a certain role for the user having that class.

The IBM default privilege classes are A to G, and all but class G define certain system administrator roles. Class G defines the general user, and it is the default class for users of the system that have no administrative capability.

You can use the **QUERY PRIVCLASS** command to list the privilege classes for which your virtual machine is authorized; Example 5-10 on page 112 shows output from this command.

s

Example 5-10 Output from the QUERY PRIVCLASS command

```
QUERY PRIVCLASS
Privilege classes for user TUX1
Currently: G
Directory: G
```

This means that your virtual machine can execute any command that is valid for class G users. Because the set of commands in any given privilege class is customizable by the system administrator, the set of commands you are able to run may vary slightly from the example.

CP also provides a command, QUERY COMMANDS, that will list all of the commands available to you, so you do not need remember your privilege classes or which commands go with them. Example 5-11 displays output from the command **QUERY COMMANDS**.

Example 5-11 Output from the command QUERY COMMANDS

```
QUERY COMMANDS
ADSTOP      ATTN        BEGIN       CHANGE      CLOSE
COMMANDS    COUPLE      CPFORMAT    CPU         DEFINE      DETACH
DIAL        DISCONNECT  DISPLAY     DUMP        ECHO        EXTERNAL
FOR         INDICATE    IPL         LINK        LOADVFCB    LOCATEVM
LOGON       LOGOFF      MESSAGE     NOTREADY    ORDER       PURGE
QUERY       READY       REDEFINE    REQUEST     RESET       RESTART
REWIND      SCREEN      SEND        SET         SIGNAL      SILENTLY
SLEEP       SMSG        SPOOL       SPXTAPE     STOP        STORE
SYSTEM      TAG         TERMINAL    TRACE       TRANSFER    UNCOUPLE
UNDIAL      VDELETE     VINPUT      VMDUMP      XAUTOLOG    XSPOOL
DIAG00      DIAG08      DIAG0C      DIAG10      DIAG14      DIAG18
DIAG20      DIAG24      DIAG28      DIAG40      DIAG44      DIAG48
DIAG4C      DIAG54      DIAG58      DIAG5C      DIAG60      DIAG64
DIAG68      DIAG70      DIAG7C      DIAG88      DIAG8C      DIAG90
DIAG94      DIAG98      DIAG9C      DIAGA0      DIAGA4      DIAGA8
DIAGB0      DIAGB4      DIAGB8      DIAGBC      DIAGC8      DIAGD0
```

As you can see in Example 5-11, even for a class G user (which by default is the most restrictive permission class) there are many commands available. The entries starting with `DIAG` represent diagnose functions that your virtual machine can call. A *diagnose function* is used by programmers and is similar to a Windows API call or a Linux system call. You cannot execute these by specifying the name on the command line.

5.2.3 Other users on the system

CP also has commands for examining the environment external to your virtual machine. Your virtual machine is probably not the only virtual machine on the system, which means that there are other things going on around you. There may be other virtual machines running other operating systems.

Other users

You can use the **QUERY USERS** command to find out how many guests are currently logged on to the system; see Example 5-2 on page 107.

Example 5-12 Output from the QUERY USERS command.

```
QUERY USERS
20 USERS,     0 DIALED,     0 NET
```

As shown from the output in Example 5-12, there are 20 users on this system. To determine what the total number of users on the system is from this output, look at the first number (USERS) displayed.

The second number (DIALED) shows how many users have used the DIAL command to connect to the system. The third number (NET) tells how many users are connected via an older communications protocol. The distinction is not very important, so the DIALED and NET portions of the output are not covered further in this book.

The **QUERY NAMES** command gives a complete list of all of the guests currently logged on the system; see Example 5-3 on page 108.

Example 5-13 Output from the QUERY NAMES command

```
QUERY NAMES

EDI      -L0005, DIRMAINT - DSC , TCPIP    - DSC , RSCS     - DSC
PVM      - DSC , DATAMOVE - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC
VMSERVR  - DSC , VMSERVU  - DSC , VMSERVS  - DSC , GCS      - DSC
OPERSYMP - DSC , DISKACNT - DSC , EREP     - DSC , OPERATOR - DSC
CLIVE    -L0008, MAINT    -L0004, EDI2     -L0006, JASON    -L0007
VSM     - TCPIP
```

The output in Example 5-13 shows all of the guests logged on to the system. The first guest in the list is EDI. The information following the dash (-) tells you how this user is connected to the system. L0005 shows that the user is connected through logical device 0005. DSC means the guest is disconnected.

A disconnected guest is a guest that is logged on to the system, but has no terminal connected to it. Programs may still be running, but no user has a terminal open, actively watching the programs. This is similar to unplugging a monitor from your desktop workstation.

CP also provides methods for obtaining information about the real hardware connected to the mainframe you are running on. However, most of that information is probably unavailable to you as a class G user (unless you are an administrator).

5.3 Working with a guest operating system

CP contains commands that allow you to start, stop and resume guest operating systems. There is also a command that allows you to issue commands directly to CP without stopping your guest operating system.

5.3.1 Starting a guest operating system

In z/VM, starting an operating system is known as “IPLing” (IPL stands for Initial Program Load). IPLing a guest operating system is the same concept as booting your desktop computer. When you IPL an operating system, it takes control of the system and any other operating system that was running before is cleared from memory and no longer controls the system. Keep in mind that CP will still be running, because CP is not a guest operating system.

In CP, you perform an IPL by using the **IPL** command. CP can IPL directly from a device, in which case the operating system is read from the data residing on the device. Or CP can IPL from a chunk of memory known as a *Named Saved System* (NSS). To IPL from a device, give the virtual device number of the device you wish to IPL; see Example 5-4 on page 108.

Example 5-14 IPL of the 190 disk

```
IPL 190
z/VM V5.3.0    2007-05-02 16:25

Ready; T=0.01/0.01 09:37:40
```

In Example 5-14 we IPLed, the 190 disk which typically contains the CMS operating system. At this point CMS is running and any command issued will be processed by CMS, not CP.

An NSS is a copy of an operating system's kernel or nucleus, which that has been saved in a chunk of CP's storage. Using an NSS to IPL an operating system has several advantages over using disks.

First, only one copy of the operating system will exist in memory no matter how many guests have IPLed it. This can lead to tremendous storage savings if you have numerous guests. Second, because only one copy of the operating system exists, then updating everyone who uses it to a newer version is as simple as replacing that single NSS.

To IPL from an NSS, provide the name of the saved system; see Example 5-15. Most z/VM installations have a CMS NSS set up.

Example 5-15 IPL of the CMS named saved system

```
IPL CMS
z/VM V5.3.0    2007-05-02 16:25

Ready; T=0.01/0.01 09:37:40
```

Example 5-15 shows we IPLed CMS via the CMS NSS instead of the CMS disk. Notice that CMS starts exactly the same way that it did when we IPLed the 190 disk.

Note: Some guest operating systems require that you perform a SYSTEM CLEAR command to clear out the guest's virtual memory before IPLing the guest. You may, alternatively, tell the **IPL** command to clear the memory for you by specifying the CLEAR parameter as shown here.

```
IPL 190 CLEAR
```

5.3.2 Issuing CP commands while running a guest operating system

After a guest operating system is started, then all commands entered at the terminal will typically be processed by that operating system and not by CP. Sometimes, however, you may need to interact with CP for various reasons, such as linking to a new disk, finding out how many other users are on the system, changing your virtual networking hardware, or any other CP-related task.

CP provides a way for you to issue commands that bypass the guest operating system and go directly to CP. We use the **#CP** command for this purpose. You issue the command **#CP** followed by whatever CP command you want to execute.

For example, assume you start your guest operating system which only detects one CPU, but you think that your virtual machine has three CPUs. Example 5-16

on page 116 shows how you can verify that your virtual machine does indeed have three CPUs.

Example 5-16 Issuing a command to CP from within a guest operating system

```
#CP QUERY VIRTUAL CPUS
CPU 00  ID  FF02991E20948000 (BASE) CP   CPUAFF ON
CPU 01  ID  FF02991E20948000 STOPPED CP   CPUAFF ON
CPU 02  ID  FF02991E20948000 STOPPED CP   CPUAFF ON
```

When you issue a **#CP** command, CP will temporarily suspend your guest operating system long enough to complete your command, and then it will resume it. To your guest operating system, it does not appear as though time has stopped or that it has stopped running. This process is generally very fast and seamless.

Notes:

- This method of interacting with CP from within your guest operating system works flawlessly for almost all guest operating systems. However, it fails when you are running z/VM in a z/VM guest. In this case, you will actually need to pause the z/VM that is running as your guest operating system before you can issue commands directly to the underlying CP. Refer to 5.3.3, “Pausing a guest operating system” on page 116, for more detailed information about this topic.
- CMS will assume that any commands you issue that it doesn’t recognize are CP commands and it will pass them directly to CP for you. This means that you do not need to use the **#CP** command to force CP to execute commands when you are in CMS. You may simply execute the CP command normally and CMS will pass the command along to CP for you.

5.3.3 Pausing a guest operating system

CP also has a command for pausing your guest operating system temporarily. You may want to do this if you need to issue many CP commands and do not want the guest operating system running or interfering with your work.

You may also want to do this if you are a programmer who is debugging a guest operating system. To stop the guest operating system and return control to CP, enter the **#CP STOP** command; see Example 5-17 on page 117.

Example 5-17 Pausing a guest operating system

```
#CP STOP
```

As you can see from Example 5-17, there is no output from the #CP STOP command. However, the end result is that the guest operating system will be stopped and you will be able to interact directly with CP until you specifically resume running the guest operating system.

This method of pausing your guest operating system works flawlessly for almost all guest operating systems. However, it fails when you are running z/VM in a z/VM guest. When you are running z/VM in a z/VM guest, you essentially have CP running as your guest operating system. Therefore, you have *two* instances of CP to deal with:

- The first level CP is the instance of CP that started when you logged on to the system and your virtual machine was created.
- The second level CP was created when you IPLed it from within your first level CP instance.

When you are running a second level CP, the **#CP** command will issue commands to your second level CP—and *not* to your first level CP, as you might expect.

To drop out of your second level CP and issue commands directly to your first level CP, press the PA1 key on the 3270 terminal keyboard. When you press the PA1 key (known as the Program Attention 1 key), then the second level CP (including any guest operating system it may be running) will be paused and your next command will be executed by your first level CP.

Note: When you press the PA1 key to pause a guest operating system, you may find that it only remains paused while you execute one command. After this it may resume again automatically. If this happens, it is because the RUN parameter is turned on for your first level virtual machine. The RUN parameter controls this behavior.

To turn off the RUN parameter after you have pressed the PA1 key and your first level CP is in control, execute the following command.

```
SET RUN OFF
```

From this point on you will be able to execute as many first level CP commands as you want and the second level CP will not resume until you tell it to.

You may also issue the **SET RUN ON** command to turn the RUN parameter on if you wish. Having RUN set to ON will ensure that your guest operating system remains running when you disconnect from the guest.

5.3.4 Resuming a guest operating system

You may resume your guest operating system (no matter what guest operating system you are running) with a simple **BEGIN** command. **BEGIN** takes no parameters and produces no output of its own, although you may see output from your guest operating system as it continues to execute.

Example 5-18 Restarting a stopped guest operating system

```
BEGIN
```

Note: In some cases you may be required to clear your 3270 terminal screen before your guest operating system will resume operation.

5.3.5 Halting a guest operating system

When you are finished with a guest operating system, use whatever command or procedure that operating system has for a proper shutdown, because not doing so can lead to data loss.

CP does, however, provide a command to halt the running guest operating system, clear the systems storage, and return directly to CP; see Example 5-19 on page 119.

Example 5-19 Halting a guest operating system

```
SYSTEM CLEAR
```

You may want to halt your guest operating system in this way if it has crashed or is stopped and is unresponsive to commands.

Note: Because you will be running a guest operating system, you may need to precede the **SYSTEM CLEAR** command with **#CP**, as discussed earlier.

5.4 Your virtual machine's virtual devices

A *virtual device* is a device specific to one virtual machine that either emulates a real device, or provides an abstracted view of a real device. Virtual devices are the primary components that make up your virtual machine. A virtual machine is simply a collection of resources and devices on which to run a guest operating system. This section focuses on learning about some of the different types of virtual devices and how to manage them.

All virtual devices have what is known as a *virtual device number* (often referred to as a *virtual device address* or simply as *vdev* in IBM documentation). This number uniquely identifies a virtual device for your virtual machine. The number range is always represented in hex and spans from 0000 to FFFF. No two virtual devices in the same virtual machine may have the same number. When you define a new device for your virtual machine, you can use any free number you choose.

The virtual devices that your virtual machine will contain when you log on are defined in the user directory by your system administrator. Because the user directory can only be updated by an administrator, you cannot simply define a new virtual device in your directory entry. In many cases, however, you can still use the **DEFINE** command to create the device you need,

A device that you create with the **DEFINE** command is known as a *temporary device* because it is not in your directory entry. And because the device is not in your directory entry, CP does not recreate it for you when you log on.

If you want to add a new virtual device to your virtual machine permanently, ask your system administrator to update your directory entry to define the device for you.

5.4.1 Querying your virtual devices

You can obtain a list of all defined devices and their device numbers by using the **QUERY VIRTUAL ALL** command; see Example 5-20. Any number not listed in this output is not defined. It is considered to be free and is available for use by any new virtual device you create.

Example 5-20 Output from the QUERY VIRTUAL ALL command

```
QUERY VIRTUAL ALL

STORAGE = 32M
XSTORE = none
CPU 00  ID  FF02991E20948000 (BASE) CP   CPUAFF ON
No AP Crypto Queues are available
CONS 0009 ON LDEV L0007   TERM STOP  HOST TCPIP    FROM 10.0.0.1
   0009 CL T NOCONT NOHOLD COPY 001    READY FORM STANDARD
   0009 TO TUX1     PRT DIST TUX1      FLASHC 000 DEST OFF
   0009 FLASH       CHAR       MDFY       0 FCB       LPP OFF
   0009 3215   NOEOF CLOSED    NOKEEP NOMSG NONAME
   0009 SUBCHANNEL = 0001
RDR  000C CL * NOCONT NOHOLD   EOF       READY
   000C 2540         CLOSED    NOKEEP NORESCAN  SUBCHANNEL = 0002
PUN  000D CL A NOCONT NOHOLD COPY 001    READY FORM STANDARD
   000D TO TUX1     PUN DIST TUX1      DEST OFF
   000D FLASH      000 CHAR       MDFY       0 FCB
   000D 2540   NOEOF CLOSED    NOKEEP NOMSG NONAME
   000D SUBCHANNEL = 0003
PRT  000E CL A NOCONT NOHOLD COPY 001    READY FORM STANDARD
   000E TO TUX1     PRT DIST TUX1      FLASHC 000 DEST OFF
   000E FLASH       CHAR       MDFY       0 FCB       LPP OFF
   000E 1403   NOEOF CLOSED    NOKEEP NOMSG NONAME
   000E SUBCHANNEL = 0004
DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
```

The basic format of data is simple. Ignore the first four lines and start with the line that begins with `CONS`. CONS (for console) is the type of the device that is represented in this part of the output.

Immediately following the device type is the devices virtual device number. Anything after this is specific to the type of the device you are looking at. Notice that our example contains the devices listed in Table 5-1 on page 121.

Table 5-1 Virtual devices in example virtual machine

Device	Virtual device number
Console	0009
Card reader	000C
Card punch	000D
Printer	000E
DASD	0190
DASD	0191
DASD	019D
DASD	019E
DASD	0401
DASD	0402
DASD	0405

To obtain information about a specific type of device, you can filter the output by specifying a type to the **QUERY VIRTUAL** command (instead of specifying **ALL**); see Example 5-21.

Example 5-21 Output from the QUERY VIRTUAL DASD command

```
QUERY VIRTUAL DASD

DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 5 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
```

Some valid device types that you can specify to the **QUERY VIRTUAL** command are listed in Table 5-2 on page 122.

Table 5-2 Virtual device types you can query

Device type	Description
CONS	Console device
DASD	Disk
PRT	Printer
PUN	Card punch
RDR	Card reader
OSA	Open Systems Adapter (network card; refer to Chapter 11, “Networking and connectivity” on page 353 for more information)
SWITCH	A virtual network switch (refer to Chapter 11, “Networking and connectivity” on page 353 for more information)

5.4.2 Processors (CPUs)

Processors are known as *CPU*s when they are virtual because CP reserves the term “processor” for real processors. A CPU is not actually a device because it does not have a virtual device number like all other devices do. Nevertheless, you can think of CPUs as devices because we manage them in a similar fashion.

Each virtual CPU has a *CPU address* which is like a virtual device number, but does not conflict with the virtual device number space. The virtual CPU address space is always represented in hexadecimal and ranges from 00 to 3F.

You can define up to 64 virtual CPUs per virtual machine. It is also interesting to note that you can have a greater number of virtual CPUs defined than the system has real processors. Virtual CPUs are scheduled to run on real processors when real processor time is available.

Having more CPUs than processors is like running more programs than you have processors on your desktop workstation. CP handles this intricate scheduling of resources much like your desktop operating system handles scheduling for your programs and their processes.

Querying CPUs

Your virtual machine must have at least *one* CPU, but it may have more. You can obtain information about the CPUs on your virtual machine by using the **QUERY VIRTUAL CPUS** command (do not confuse this with the **QUERY VIRTUAL CPU** command, which is a completely different command). See Example 5-22 on page 123.

Example 5-22 Output from the QUERY VITUAL CPUS command

```
QUERY VIRTUAL CPUS
CPU 00  ID  FF02991E20948000 (BASE) CP   CPUAFF ON
```

This output contains one line per CPU. The virtual machine in this example only has one CPU, and that CPU address is 00. Its CPUID is FF02991E20948000.

Encoded in the CPUID is information about the hardware and the environment your virtual machine is running in. The details are not critically important and thus not covered here. Setting CPUAFF on means that CPU affinity is on for this CPU. “CPU affinity” refers to how having different types processors and CPUs is handled on your system; this is a complex subject and is beyond the scope of this textbook.

Notice the text `(BASE)`; this denotes the *base CPU*. This means that this CPU was the first one created when you logged on and instantiated your virtual machine. This CPU is tied to your virtual machine and cannot be detached or redefined. All other CPUs can be detached and redefined.

Defining CPUs

If you have the appropriate privileges, you can define more CPUs for your virtual machine. The ability to define more CPUS is defined in your virtual machine's directory entry; this is controlled by the system administrator.

If you are allowed to have more than one CPU defined, then you are given a limit somewhere between 2 and 64. You may define a new CPU by using the **`DEFINE CPU`** command; see Example 5-23.

Example 5-23 Defining a single virtual CPU

```
DEFINE CPU 1
CPU 01 defined
```

`DEFINE CPU` takes an argument that is either the CPU address you want the new CPU to have, or a dash-separated range of addresses if you wish to define more than one CPU; see Example 5-24.

Example 5-24 Defining multiple virtual CPUs

```
DEFINE CPU 1-3
CPU 01 defined
CPU 02 defined
CPU 03 defined
```

Note: CP does not provide a command for checking the maximum number of CPUs you are allowed to define. You need to determine this by either trial and error, or by checking your user directory entry for the MACH statement.

Detaching CPUs

If you decide that you do not need a virtual CPU that is already defined, then you may get rid of it by using the **DETACH CPU** command. **DETACH CPU** takes the same argument format as **DEFINE CPU**, which means you can detach a single CPU or a whole range of them with a single command; see Example 5-25.

Example 5-25 Detaching a CPU with the DETACH command

```
DETACH CPU 1
CPU 01 detached
```

Note: Detaching a CPU will cause your virtual machine to be reset and any running operating systems will be halted immediately—so use this command carefully.

5.4.3 Storage (main memory)

Every virtual machine has memory. How much storage any particular virtual machine is allowed to have is determined by the system administrator and is stored in the user directory.

“Storage” is not really a device, because it does not have a device number like all other devices do. Managing storage is simple because there is very little you need to know about it.

Your virtual machine’s memory is virtual, just like all of its other resources. Because it is virtual, it may exist anywhere in physical memory, or even on a disk (backing physical memory with disk-based storage is commonly referred to as “paging” or “swapping”).

As with CPUs, CP will allow you to overcommit real storage. When a guest is not logged on, it is not consuming any memory at all. It is only when a guest logs on that CP allocates some memory for it and its virtual CPUs and devices are created.

But even when a guest with 1 GB of storage logs on, it is not immediately using all 1 GB of its memory. If it is not running an operating system, then it is actually using an extremely small chunk of its memory. CP only allocates physical

memory to back a guest's virtual memory when that guest actually uses the memory.

How much memory your VM has

Determining how much memory you have is easy if you use CP's **QUERY VIRTUAL STORAGE** command; see Example 5-26.

Example 5-26 Output from the QUERY VIRTUAL STORAGE command

```
QUERY VIRTUAL STORAGE
STORAGE = 32M
```

In Example 5-26, the virtual machine has access to 32 Megabytes of storage. If you need more storage than is initially provided for you by CP, you might have the capability to increase your storage allotment.

When an administrator sets up a directory entry for a virtual machine, the administrator specifies two values for that virtual machine's storage. The first value is the initial amount of storage that virtual machine is given at logon time, and the second value is the maximum amount of storage that virtual machine is allowed to use.

Changing your storage size

You can change the amount of memory of your virtual machine by using the **DEFINE STORAGE** command. This command can be used to either reduce or increase storage capacity. It takes a single argument, which is the amount of storage that you would like your virtual machine to have.

The argument has two parts: a number, and then the unit of measure. Valid units of measure include those shown in Table 5-3.

Table 5-3 Valid units of storage measurement

Abbreviation	Amount of memory
K	Kilobytes
M	Megabytes
G	Gigabytes
T	Terabytes

Use this command carefully, because it causes a system reset which means it *clears* all of the virtual machine's memory and *stops* any running guest operating systems.

If you are simply running CP and not running a guest operating system, however, then there is nothing to worry about. In Example 5-27, we redefine our virtual machine to have 4 MB of storage.

Example 5-27 Setting the virtual machine's storage size to 4 megabytes

```
DEFINE STORAGE 4M
STORAGE = 4M
Storage cleared - system reset.
```

As shown in Example 5-28, we revert to our original size.

Example 5-28 Setting the virtual machine's storage size to 32 megabytes

```
DEFINE STORAGE 32M
STORAGE = 32M
Storage cleared - system reset.
```

Note: Your storage needs will vary depending on what your are doing with your virtual machine but chances are you do not need as much storage as you probably initially think. With a typical desktop computer we tend to estimate memory needs very high as memory grows increasingly cheaper and a high estimate imposes no performance penalty to the system. With z/VM it is important to understand your guest operating system, the workloads that will be run on it and their memory consumption characteristics.

Note: CP does not provide a command for checking the maximum storage size you are allowed to define. You can, however, determine this by simply attempting to define a storage size of 99 Terabytes (**DEFINE STORAGE 99T**) which will more than likely be far more than you are allowed to define. In this case, the resulting error message will tell you your maximum storage size.

5.4.4 DASD (disk devices)

DASD is an acronym for *Direct Access Storage Device*. A DASD device is a simple magnetic disk that you have access to. This is exactly like a hard drive in your desktop computer. The disk contains files important to some guest operating system.

Terminology

Before discussing DASD, it is important to understand some of the associated terminology.

The mainframe that your z/VM instance is running on is connected to some type of physical disk that we will refer to as “real DASD”. This DASD is segmented into units called DASD packs (also referred to as “volumes”). General users do not have the capability to examine the real DASD packs on the system.

Note: Real DASD packs can be varying sizes. The names and sizes are as follows:

- 3390-3 (also known as mod 3) 3339 cylinders - roughly 3 GB in size
- 3390-9 (also known as mod 9) 10017 cylinders - roughly 9 GB in size
- 3390-27 (also known as mod 27) 30051 cylinders - roughly 27 GB in size

The system administrator can dedicate a DASD pack to a guest. This means that access to the entire pack is given directly to that guest. This is known as a *dedicated DASD pack*.

A real DASD pack can be home to many smaller virtual DASD packs (much as modern hard disks can be partitioned into many partitions that each appear to be a single disk). One of these smaller partitions is called a *minidisk*. Any number of minidisks can be created from a DASD pack, and each one can be arbitrarily assigned to a different guest. A guest can have many minidisks.

A guest uses a minidisk and a dedicated DASD device in exactly the same ways. In both cases, the guest sees a virtual DASD device. A virtual DASD device is sometimes just referred to as a *disk*.

There are three different types of DASD to discuss here:

- Real DASD lives on a real physical disk, and is used to create minidisks.
- TDISK-based DASD lives on real DASD, but is only temporary and is destroyed when you log off.
- VDISK-based DASD lives in storage and is also destroyed when you log off.

Examining your DASD

To obtain a list of all of the virtual DASD devices that are available to you, use the **QUERY VIRTUAL DASD** command; see Example 5-29.

Example 5-29 Output from the QUERY VIRTUAL DASD command

```
QUERY VIRTUAL DASD
DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
```

```
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
```

Example 5-29 on page 127 displays many different DASD devices available to us. The output shows one DASD device per line; look at the first line in the example.

- The number immediately following the word DASD is the virtual device number of this DASD device (0190, in this case).
- The number immediately following that (3390, in all cases in this example) is the type of actual DASD hardware in use.
- The volume label (LX6RES, in this case) of the real DASD pack that this virtual DASD device lives on is shown next.
- The level of access (R/O, in this case) you have to the device is shown next.
- The size of this DASD device (107, in this case) is shown in cylinders (CYL).
- The real address of the real DASD pack that this virtual DASD device lives on is displayed next (CD31, in this case).
- Finally, the subchannel number for our virtual device is listed (SUBCHANNEL = 0005, in this case). A discussion of subchannels is beyond the scope of this book.

Here we explain the most significant parameters in more detail.

Virtual device number

As previously mentioned, the virtual device number uniquely identifies this device to your virtual machine.

Type

The type used to be important many years ago when there were multiple types of real DASD, but except for the oldest installations, you should not run across any type of DASD today other than 3390. The type parameter is kept in place in order to maintain backward compatibility. The only exception to this is when you are dealing with VDISK (see 5.4.6, “Virtual DASD (VDISK)” on page 133 for more information about this topic).

Volume label and real device address

The volume label and the real device address both apply to the real DASD pack that houses this particular virtual DASD device. These parameters are provided for informational purposes and you are not likely to need to refer to them unless you are a system administrator.

Size in cylinders

The size of DASD devices is given in cylinders, and not in bytes like most people are familiar with. For 3390 DASD, a single cylinder is exactly equivalent to 849,960 bytes—which in turn is roughly equivalent to 850 KB.

Note:

- To determine how many megabytes your x cylinder DASD device holds, multiply x by 0.85.
- To determine how many cylinders you need for x MB of data, multiply x by 1.2.

Creating DASD

At some point, you may find that you need more disk space. However, the definition of a new DASD pack must be performed by an administrator, because a new virtual DASD device must be backed by a real disk somewhere, and general users do not have the authority to allocate from real disks.

You may, however, have the authority to create a TDISK or a VDISK; refer to 5.4.5, “Temporary DASD (TDISK)” on page 131 and 5.4.6, “Virtual DASD (VDISK)” on page 133s for details about these topics. Another option might be to ask your administrator for shared file pool (SFS) space; refer to 6.6.7, “CMS Shared File System” on page 178 for further details.

Accessing someone else's DASD

CP provides a way for you to access a DASD device owned by someone else on the system. This is known as “linking to another DASD”. Linking to another DASD device is a very useful way for you to share files with another user on the system, or to get access to parts of CP (or CMS, or some other guest operating system) that you do not initially have access to when you log on.

For example, all of the z/VM networking tools (such as telnet, netstat and ftp) are usually located on the 592 disk of the guest named TCPMAINT. To get access to TCPMAINT's 592 disk in your virtual machine, use the **LINK** command.

LINK takes four basic parameters:

1. It needs the name of the guest owning the disk you wish to link.
2. It needs the virtual device number of that disk (relative to the owning guest).
3. It then needs the virtual device number you wish to define in your own virtual machine for this linked disk.
4. Finally, it needs a two-letter code describing the level of read/write access you want to have for this disk.

Optionally, a password can follow all other parameters. Disk passwords can be defined by the system administrator to protect disks from general access. If you need access to a password-protected disk, ask the disk's owner or the system administrator.

Example 5-30 Example of the LINK command

```
LINK TCPMAINT 592 592 RR
DASD 0592 LINKED R/O
```

Example 5-30 shows we are linking TCPMAINT's 592 disk as 592 in our own virtual machine. When we execute a **QUERY VIRTUAL DASD** command, we see that the 592 disk is indeed present, as shown in Example 5-31.

Example 5-31 Verifying that we linked to a disk

```
QUERY VIRTUAL DASD
DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
DASD 0592 3390 LX6W01 R/O 070 CYL ON DASD  CD32 SUBCHANNEL = 000B
```

As shown in Example 5-31 we now have a virtual disk mapped to the exact real disk as TCPMAINT's 592 virtual disk.

Note: We chose to link TCPMAINT's 592 disk to our virtual device number 592 to keep things simple and uniform. However, you are free to choose any unused virtual device number in your virtual machine to be used for the link.

For example, to link TCPMAINT's 592 disk to your virtual device number 300, execute this command:

```
LINK TCPMAINT 592 300 RR
```

Notice that in our case, we have R/O (read only) access to this disk. This is because when we executed the **LINK** command to link this disk, we specified RR as the access mode. Table 5-4 on page 131 shows common access modes you can use when linking disks.

Table 5-4 LINK access modes

Access mode	Description
RR	Read only access.
WW	Write access. Only granted if no one else is linked to this disk.
M	Multiple access. Only granted if no one else has write access to this disk.
MW	Multiple write. Allow write access even if other users have this disk linked with write access.
SR	Stable read. When granted, no other write links can be create for this disk.
SW	Stable write. When granted, no other write links can be create for this disk.
ER	Exclusive read. When granted, no other links can be created for this disk.
EW	Exclusive write. When granted, no other links can be created for this disk.

Removing DASD

You can remove a DASD device just like you would remove any other virtual device, by using the **DETACH** command. DETACH takes the virtual device number of the device to detach as its only parameter.

Example 5-32 Example of the DETACH command

```
DETACH 592
DASD 0592 DETACHED
```

Do not be concerned that you will lose anyone else's data by detaching a disk that you linked. When you detach any disk, all you are doing is removing the virtual device within *your* virtual machine that points to the real disk. (This is not true, however, for TDISKs and VDisks.)

5.4.5 Temporary DASD (TDISK)

A TDISK is a virtual DASD device that is allocated from a pool of real DASD packs set aside specifically for the creation of temporary disks. TDISK is an abbreviation of temporary disk.

It is important to note that when you log off or the system fails, any TDISKs that you have created are destroyed. A TDISK is meant to be a temporary storage space.

Important: Do not store the only copy of important data on a TDISK.

Creating a TDisk

Not all z/VM installations will allow the creation of TDISKs. This might be the case for your installation if the system administrator has not set up the system for TDISK allocation, or if all TDISK space is in use at the time of your request.

However, you can request a TDISK allocation by using the **DEFINE T3390** command. The T3390 argument specifies that you want to create a temporary model 3390 DASD device.

Note that **DEFINE T3390** takes two parameters: the virtual device number you want to assign to the new disk, and the size of this disk in cylinders. So, if you want a TDISK at device number 9FF that is 100 cylinders in size, use the command shown in Example 5-33.

Example 5-33 Defining a TDISK

```
DEFINE T3390 9FF 100
DASD 09FF DEFINED
```

Note: If you want an x megabyte TDISK, multiply x by 1.2 to determine how many cylinders to specify during creation.

You can use the **QUERY VIRTUAL DASD** command to verify that your 9FF disk exists, as shown in Example 5-34.

Example 5-34 Verifying that a TDISK was created

```
QUERY VIRTUAL DASD
DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
DASD 0592 3390 LX6W01 R/O 070 CYL ON DASD  CD32 SUBCHANNEL = 000B
DASD 09FF 3390 (TEMP) R/W 100 CYL ON DASD  59D4 SUBCHANNEL = 000C
```

Notice that there is no volume label for the TDISK, because it is residing in TDISK, and not on a specific real DASD pack.

Removing a TDISK

You may remove a TDISK just like you would remove most other virtual devices, by using the **DETACH** command; see Example 5-35. **DETACH** takes the virtual device number of the device to detach as its only parameter.

Example 5-35 Removing a TDISK

```
DETACH 9FF
DASD 09FF DETACHED
```

5.4.6 Virtual DASD (VDISK)

A VDISK is a virtual DASD pack that resides in the machine's storage instead of on a real disk somewhere. This is exactly like the concept of a RAM disk that you might be familiar with from other operating systems. Because they exist in storage, VDISKs tend to be significantly faster than normal DASD devices, and also significantly smaller.

It is important to note that a VDISK exists exclusively in storage and is *not* backed by a real disk. This means that when you log off or the system fails, your VDISK is destroyed. A VDISK is meant to be a temporary storage space.

Note: Do not store the only copy of important data on a VDISK.

Creating a VDisk

Not all z/VM installations will allow the creation of VDISKs. This might be the case for your installation if the system administrator has not set up the system for VDISK allocation, or if free memory is too scarce.

However, you can request a VDISK allocation by using the **DEFINE VFB-512** command. The VFB-512 portion is an acronym for Virtual Fixed Blocks of 512 bytes.

Note that the **DEFINE VFB-512** command takes two parameters: the virtual device number you want to assign to the new disk, and the size of this disk in 512-byte blocks. So, if you want a VDISK at device number 8FF that is 16 MB in size, use the command shown in Example 5-36 on page 134.

Example 5-36 Creating a VDISK

```
DEFINE VFB-512 8FF 31250
DASD 08FF DEFINED
```

Note: If you want an x megabyte VDISK, multiply x by 2000 to determine how many blocks to specify during creation.

You can use the **QUERY VIRTUAL DASD** command to verify that your 8FF disk exists, as shown in Example 5-37.

Example 5-37 Verifying that a VDISK was created

```
QUERY VIRTUAL DASD

DASD 0190 3390 LX6RES R/O 107 CYL ON DASD  CD31 SUBCHANNEL = 0005
DASD 0191 3390 DKCD37 R/W 005 CYL ON DASD  CD37 SUBCHANNEL = 0000
DASD 019D 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0006
DASD 019E 3390 LX6W01 R/O 250 CYL ON DASD  CD32 SUBCHANNEL = 0007
DASD 0401 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0009
DASD 0402 3390 LX6W01 R/O 146 CYL ON DASD  CD32 SUBCHANNEL = 0008
DASD 0405 3390 LX6W01 R/O 156 CYL ON DASD  CD32 SUBCHANNEL = 000A
DASD 0592 3390 LX6W01 R/O 070 CYL ON DASD  CD32 SUBCHANNEL = 000B
DASD 08FF 9336 (VDSK) R/W 31256 BLK ON DASD  VDSK SUBCHANNEL = 000C
```

Notice that in this example, the DASD type is 9336 for the VDISK, and not 3390 like it is for normal DASD. Also notice that there is no volume label or real DASD pack address listed for the VDISK, because it is not residing on a real disk. Also, the size is given in 512 byte blocks instead of cylinders.

You can see that the size of the disk that we received (31256 blocks) is slightly larger than the size we requested (31250). This is because VDISKs must be allocated in increments of 8 blocks. If you specify a size that is not an increment of 8, then CP will automatically round up for you so you get at least as much space as you asked for, and possibly more (but never less).

Note: Other users will not be able to link to your VDISK unless it is defined in the user directory.

Also, if a VDISK is defined in your directory and you log off, it will *not* be destroyed if another user has linked to it.

Removing a VDISK

You can remove a VDISK just like you would remove any other virtual device, by using the **DETACH** command. The **DETACH** command takes the virtual device number of the device to detach as its only parameter; see Example 5-38.

Example 5-38 Removing a VDISK.

```
DETACH 8FF
DASD 08FF DETACHED
```

5.4.7 Spool devices

A *spool device* is typically used to read, write, or process an ordered list of files or data kept in a queue. CP's spool devices are slightly different from normal devices. Spool devices are virtual, just like all of the other devices we have discussed.

There are three different spool devices that you can have, each of which exists primarily to work with spool files. The specifics of what spool devices and files are and how they work can be quite complex, but most of the details are not very important for the vast majority of users.

Note: This book presents an extremely simplified view of spool devices, because most users may never need to know all of the specifics. For more information about this topic, however, you can refer to *Virtual Machine Operation*, SC24-6128.

Table 5-5 Types of spool devices

Spool device	Description
Reader	A virtual punch card reader
Punch	A virtual punch card punch
Printer	A virtual printer

For the purpose of our discussion, these devices can all be viewed as being the same. More specifically, they are all spool devices and all of the commands that we use for managing files within the spool device queues can be used on all of them in the same manner. In our examples, we will concentrate on the reader device type.

Uses for spool devices

In the early days of z/VM, most installations would include real printers, punch card readers and punches. Back then these virtual devices would be used to interface to the real hardware, so you could actually work with the real devices. Today, however, most installations do not have the need for these real devices. However, spool devices (mainly the reader and punch) can still be used for a few very important things.

The main use of your reader today is to act as a mail box for files. You may, by using the CMS command **SENDFILE**, send a file from one of your disks to other users. Those users are then free to save the file from their reader to one of their disks. The **SENDFILE** command actually uses your punch device to “punch” the file into the appropriate reader queue.

The virtual card reader is also often used for booting other operating systems, including Linux installers.

Note: CMS expects to find your reader at virtual device number 000C, your punch at 000D, and your printer at 000E. For this reason, those devices are typically always created with the given device numbers.

Querying a spool device

Querying a spool device gives you a great deal of information, and it can be done by using the **QUERY** command in a similar way to querying other virtual devices.

In Example 5-39 we query our reader, which is typically defined with the virtual device number 000C. (If your reader is not addressed as 000C, use the **QUERY VIRTUAL ALL** command to find your reader as discussed in 5.4.1, “Querying your virtual devices” on page 120.)

Example 5-39 Querying a spool device

```
QUERY VIRTUAL 000C
RDR  000C CL * NOCONT NOHOLD   EOF       READY
000C 2540         CLOSED    NOKEEP NORESCAN  SUBCHANNEL = 0002
```

Most of the parameters shown in the output deal with the spool device configuration or how your spool files are treated by the spool device.

Creating a spool device

You create a spool device by using the **DEFINE** command. In most cases, your reader will have already been defined for you. But if it has not been defined, give as an argument to the **DEFINE** command the type of device you are creating and the virtual device number, as shown in Example 5-40 on page 137.

Example 5-40 Defining a spool device

```
DEFINE READER 00C

RDR  000C DEFINED
```

You can specify PUNCH and PRINTER in place of READER, to create those types of spool devices instead.

Removing a spool device

You can remove a spool device by using the **DETACH** command, just like any other virtual device. Specify the virtual device number as an argument; see Example 5-41.

Example 5-41 Removing a spool device

```
DETACH 100
RDR  0100 DETACHED
```

This will work on punch and printer devices, as well.

Spool files

A spool file is similar to a normal file that you would use in Microsoft Windows, GNU/Linux or even in CMS. The main differences between the files you are probably used to and spool files are the naming convention used and how the data is organized within the file itself.

Spool files are arranged in fixed size rows (known as *records*) instead of being a simple stream of bytes. Because spool files work exactly like CMS files, you can refer to 6.6.1, “The CMS file system” on page 168 for more detailed information about this topic.

Note: We mentioned earlier that CP does not understand the concept of a “file”, and to some degree that is still true because you really cannot examine the contents of a file in a spool device in CP, and what you can do with a spool file is extremely limited without the help of a guest operating system like CMS.

When your z/VM system was set up, your system administrator set aside some disk space for spool files. Any files that are in one of your virtual spool device queues actually exist on this disk space set aside as spool space.

Displaying files in your reader queue

CP allows you to query a small amount of information about the files in your reader queue. Use the **QUERY READER ALL** command for this purpose, as shown in Example 5-42.

Example 5-42 Output from the QUERY READER ALL command

```
QUERY READER ALL
ORIGINID FILE CLASS RECORDS  CPY HOLD DATE TIME     NAME      TYPE
FRED   0008 A PUN 00000017 001 NONE 06/07 13:46:01 TUX1    NETLOG
FRED   0009 A PUN 00000008 001 NONE 06/07 13:49:44 PROFILE   EXEC
```

Example 5-42 shows that there are two files in the reader queue: a file named TUX1 NETLOG and a file named PROFILE EXEC. (For a discussion about the file naming system, refer to 6.6.2, “Filename structure” on page 168.) There is a significant amount of output shown in this example, but the most important parts are listed in Table 5-6.

Table 5-6 Output from QUERY READER ALL command

Column	Description
ORIGINID	This is the user name of the virtual machine that sent the file to your reader queue.
FILE	This is the unique number assigned to the file. We will use this with other commands.
DATE & TIME	The date and time that this file was placed in the reader queue.
NAME & TYPE	The file name and type given to this file by the creator.

You can query the files in your printer and punch queues by substituting PRINTER or PUNCH for READER in the QUERY command shown in Example 5-42.

Adding files to your reader

Adding a file to the reader queue is typically done with a guest operating system.

Deleting files in your reader

You can remove a file from your reader queue by using the **PURGE** command. **PURGE** can be told to clear a single file, multiple files, or even all of the files in your reader queue. To clear a single file with **PURGE**, specify which queue to delete from and the number of the file within that queue to delete; see Example 5-43 on page 139.

Example 5-43 Deleting a single file from your reader

```
PURGE READER 8
0000001 FILE  PURGED
```

CP responds that it deleted one file. If you want to delete multiple files, specify all of the file numbers for the files you want to delete, separated by spaces; see Example 5-44.

Example 5-44 Deleting multiple files from your reader

```
PURGE READER 8 9 10
0000003 FILES PURGED
```

CP now responds that it has deleted all three of the files specified. To delete all of the files in your reader queue, specify the **ALL** parameter to **PURGE**, as shown in Example 5-45.

Example 5-45 Deleting all files from your reader

```
PURGE READER ALL
0000004 FILES PURGED
```

You can delete the files in your printer and punch queues by substituting PRINTER or PUNCH for READER in the PURGE command shown in Example 5-45.

Moving a file in your reader to someone else's reader

The CMS **SENDFILE** command provides a useful way to move a file from one of your disks to someone else's reader queue. If, instead, you want to send a file from your reader queue to someone else's reader queue, use the CP **TRANSFER** command, as shown here.

```
TRANSFER yourID READER fileNumber destID READER
```

Table 5-7 on page 140 lists the TRANSFER command parameters.

Table 5-7 TRANSFER commands parameters

Argument	Description
yourID	This is your user ID. If you are a system administrator, you may actually have permission to transfer files from anyone's reader queue by specifying their guest name here.
fileNumber	This is the number of the file in your reader queue you want to send.
destID	This is the user ID of the person that you want to receive the file you are sending.

As shown in Example 5-46, we transfer file number 13 from user TUX1 to user FRED.

Example 5-46 Moving a file from TUX1's reader queue to FRED's reader queue

```
TRANSFER TUX1 READER 13 FRED READER
RDR FILE 0013 SENT TO FRED RDR AS 0025 RECS 0008 CPY 001 A NOHOLD NOKEEP
```

The output shown in Example 5-46 states that the file was sent to FRED, as we have requested. If someone is logged on to FRED's console, then they will see a similar message alerting them to the fact that someone has sent them a file.

The **TRANSFER** command works with PRINTER and PUNCH devices as well. Simply substitute PRINTER or PUNCH for READER in the example.

5.4.8 Communication devices

CP allows you to define several different types of devices that allow you to communicate with other virtual machines, as briefly discussed here. For detailed information about these devices, refer to Chapter 11, “Networking and connectivity” on page 353.

Virtual Network Interface Card

A virtual Network Interface Card (NIC) is similar to a real network card for your desktop computer. It allows your guest operating system to establish a TCP/IP-based connection to another virtual machine.

Virtual NICs, like real NICs, must be connected to another networking device such as a hub or a switch, in order to communicate. In z/VM we connect virtual NICs to guest LANs or VSWITCHES, which are two types of virtual networking devices.

You can create a virtual NIC by using the **DEFINE NIC** command. You can remove a virtual NIC by using the **DETACH NIC** command.

Channel-to-channel adapter

A channel-to-channel adapter (CTCA) is used similar to a standard network crossover cable. A CTCA allows you to create a private communication link between your virtual machine and another virtual machine. This communication channel can then be used by the guest operating system to transfer information bidirectionally between systems.

You can create virtual CTCA devices by using the **DEFINE CTCA** command. You can remove virtual CTCA devices by using the **DETACH** command, just as used for other devices.

5.5 Terminal management

Your virtual terminal is represented by your terminal emulator. A *terminal emulator* is the program you are looking at when you are issuing commands to CP and reading the results. The terminal emulator typically presents a black background with mostly green text. CP provides several ways for you to modify what your terminal looks like and how it behaves.

5.5.1 Setting the clear screen timeout

You may have noticed that when your screen fills up, CP displays the text `More...` in the lower right corner of the screen and waits for you to press the key on your keyboard that clears the screen before displaying more output.

By default, CP will wait 60 seconds for you to press the clear key before clearing it automatically. However, you can customize this behavior by using the **TERM** command:

```
TERM MORE t1 t2
```

Table 5-8 on page 142 lists the TERM MORE parameters.

Table 5-8 TERM MORE command parameters

Argument	Description
t1	This refers to the number of seconds that CP will wait before sounding a bell or beep to alert you to the fact that the screen is about to be auto-cleared for you.
t2	This refers to the number of seconds after the bell is sounded before the auto-clear actually takes place.

You can prevent CP from holding the screen and forcing you to press the clear key by setting both timeouts to zero (0) seconds, as shown here:

```
TERM MORE 0 0
```

Keep in mind, however, that by setting the timeouts to zero, you may not have time to read all of the messages presented to you on the terminal. In some cases, this may be acceptable (for example, if you are running a command or a program within a guest operating system that is generating a significant amount of output that you do not care about).

You can return to the default behavior of 60 seconds (50 seconds before the bell, 10 additional seconds before the clear) by using this command:

```
TERM MORE 50 10
```

Note: The maximum value you can specify for both timeouts is 255 seconds. This means that the maximum wait before an auto-clear is 8 minutes and 30 seconds.

5.5.2 Highlighting user input

The CP **TERM** command has a parameter that causes all user input (that is, all of the text that you specifically enter) to be highlighted in a color different than the color used for other text output.

You can turn highlighting on by using the **TERM HILIGHT ON** command. Any command executed after this will be highlighted in a different color on the screen. This makes it particularly easy to visually separate input and output and generally makes your terminal text easier to read.

You can turn highlighting off by using the **TERM HILIGHT OFF** command.

5.5.3 Changing screen colors

You can change the colors that CP uses to display input, output, and status on your 3270 terminal emulator window by using the **SCREEN** command:

```
SCREEN area color effect
```

Table 5-9 lists the SCREEN command parameters.

Table 5-9 SCREEN command parameters

Argument	Description
area	This refers to the area of the screen you are interested in changing. Valid areas are listed in Table 5-10.
color	This refers to the color you want to use for the text in your chosen area. Valid colors are listed below.
effect	This refers to the effect you want to apply to the text in your chosen area. Valid effects are listed in Table 5-11.

Table 5-10 lists the valid areas for the SCREEN command.

Table 5-10 Valid areas for the SCREEN command

Area	Description
INAREA	The command line area where you issue commands to CP and guest operating systems.
STATAREA	The display of the terminal status at the bottom of the screen.
OUTAREA	The area where messages from CP and guest operating systems are placed for you to see. This is the area that takes up the vast majority of the screen.
CPOUT	This is not really an area. Rather, it is used to cause all messages that come from CP to be formatted as specified in the rest of the command.
VMOUT	This is also not really an area. Rather, it is used to cause all messages that come from a guest operating system to be formatted as specified in the rest of the command.

The following colors are valid for the **SCREEN** command:

- BLUE
- TURQUOISE
- RED

- PINK
- GREEN
- WHITE
- YELLOW

Table 5-11 lists the valid effects for the **SCREEN** command.

Table 5-11 Valid effects for the SCREEN command

Effect	Description
BLINK	This effect causes the text on the screen to blink.
REVVIDEO	Reverse video. This swaps the text color and the background color. For example, green text on a black background would appear as black text on a green background with the reverse video effect applied.
UNDERLIN	This causes the text to be underlined.
NONE	This means: apply no effects. Print the text normally in the color specified.

Note: To reset your terminal to the default state, use the following **SCREEN** command:

```
SCREEN ALL DEFAULT
```

5.6 z/VM services

z/VM provides several services in the form of virtual machines. Unlike a traditional operating system, CP does not have the concept of a “process”. Instead, all CP knows is how to run virtual machines. This means that any extra service functionality not built directly into CP must be implemented as a service virtual machine.

A *service virtual machine* is the same as any other virtual machine, except it runs some software (typically on top of CMS) that provides a service to some or all of the other users on the system.

The following examples are services provided by service virtual machines:

- TCP/IP networking stack
- User directory maintenance (Dirmaint)
- Security manager (RACF)
- Performance data collection and reporting (Performance Toolkit)
- Systems management service (SMAPI)

Service virtual machines in z/VM usually have a user name that corresponds with the service it provides. Table 5-12 lists some of the common service virtual machines and brief descriptions of the service provided.

Table 5-12 Common service virtual machines

Guest name	Service provided
TCPIP	TCP/IP networking stack and tools like FTP and telnet.
DIRMAINT	User directory maintenance.
DATAMOVE	Provides disk copying and formatting services for DIRMAINT.
PERSFVM	Performance recording and reporting.
RACFVM	Security management for guests, devices and services.
VSMSERVE	Systems management service.
RSCS	Remote spool device capability.
PVM	Remote communication and system access.
EREP	Error recording.
VMSERVR VMSERVS VMSERVU	Support for SFS.

5.7 Exercises

1. Explain the CP user directory and its purpose.
2. This question has two parts:
   a. What important virtual machine parameter do you need to remember to check (or set) before disconnecting from a production virtual machine?

b. Why is it important to check/set the parameter mentioned in part a?

3. You are a system administrator for a virtual machine that is providing Web hosting services for a very popular company that uses their Web site to process customer order information.

   You are asked to make a few hardware configuration changes to the z/VM guest running the Web server, but you obviously cannot simply shut down this customer's very busy Web site.

   How would you issue the needed commands to CP without shutting down the Web server?

4. What is the difference between a TDISK, a VDISK, and a regular minidisk? In what situations would you prefer one over the other two?

5. Assume that you issue the HELP command, only to discover that HELP does not work. You later learn that HELP failed because you do not have access to the CMS help disk, which is MAINT's 19D disk.

   What CP command (the full command, not just the command name) would you need to use before you could access this disk to use the HELP command within CMS?

6. You need 6 GB of disk space to perform a Linux install, but you do not have disks that are large enough for this task.

   Based on your knowledge of how real DASD devices are typically sized and how they are partitioned and given to users, what type of DASD will your administrator likely give you access to, so you can perform your install?

6

Conversational Monitor System

The Conversational Monitor System (CMS) is the most commonly used point of interaction with z/VM. CMS expresses the vast power of the mainframe through a rich set of commands and utilities that build on the toolset CP provides. This chapter introduces you to the tools needed for day-to-day z/VM use.

Objectives

On completion of this chapter you should be able to:

- Discuss the chief functions of CMS
- Describe the use and benefits of full screen CMS
- Recognize the Window Manager screen
- Use the integrated help system
- Manage files in the file system
- Edit files
- List some CMS facilities

© Copyright IBM Corp. 2007. All rights reserved.

6.1 CMS introduction

Now that we have examined CP, we turn our attention to the Conversational Monitor System (CMS). As you learned earlier, CP is really designed to assign virtual resources to a guest operating system. CMS is the operating system shipped with z/VM to be used as the default z/VM guest operating system.

6.1.1 Overview

CMS is an operating system designed to facilitate mainframe virtual machine administration, by providing users an environment with a higher level of functionality than CP. CMS executes on top of CP just as any other another operating system that can run on CP. You can think of CMS as the user space portion of the Linux distribution.

CMS can help you perform a wide variety of tasks such as writing, testing, and debugging application programs for use on CMS or guest systems; executing application programs developed on CMS or guest systems; creating and editing data files; sharing data between CMS and guest operating systems; and communicating with other system users.

Note: CMS is a powerful operating environment, but be aware that CMS *cannot* be running alongside another guest operating system—only CP can do that. At any time when in a CMS environment, you can issue underlying CP commands, but the converse is not true!

Interestingly, a running instance of CMS ceases to exist when you load Linux or any other operating system from within it. In that respect, CMS can be viewed as a form of high functionality boot loader like the Grand Unified Boot Loader (GRUB) found on most modern Linux systems. Keep in mind, though, that CMS is a full operating system in itself, and in many instances it is not used to load another operating system.

In fact, CMS provides a rich set of commands that are quite useful. The CMS user has facilities similar to those found on basic personal computers. The essential functionality includes a file system, a command-issuing structure, a help system, a file editor, a command procedure language, a session manager, and other useful utilities. Though similar in function, the interface is somewhat less intuitive than modern personal computer systems.

This chapter guides you through many of the common utilities that are shipped with CMS by default. Where possible, we relate CMS concepts to other operating systems you may be more familiar with.

6.1.2 Characteristics of CMS

CMS has certain special characteristics which make it unique among IBM operating systems. It is like a personal computer operating system and has been kept compact because it has been designed as a single user system. This means that users have their own virtual machine entirely to themselves. Additionally, problems and errors that are limited to one virtual machine are limited to one user (and vice versa).

A CMS virtual machine is driven with commands (of which there are many) from a console, as is done with CP. A command will generally consist of the command name (usually a verb of some kind) and a variable number of qualifying details. The common separator in all CMS and CP commands is the blank character. This demands a minimum distortion of natural language processes. In other words, CMS commands tend to look more like written English than the equivalent Linux or DOS commands.

6.1.3 About your CMS environment

Accessing CMS is done through the same mechanisms shown in Chapter 4, “z/VM - job roles and basic concepts” on page 67. Use the usual 3270 terminal emulator and login procedure that you previously used. When you have logged on to the mainframe, you can determine whether you are in CMS by looking for a message such as:

```
NIC 0600 is created; devices 0600-0602 defined
z/VM Version 5 Release 3.0, Service Level 0701 (64-bit),
built on IBM Virtualization Technology
There is no logmsg data
FILES: 0003 RDR, NO PRT, NO PUN
LOGON AT 10:56:42 EST MONDAY 11/19/07
z/VM V5.3.0 2007-06-14 11:51
```

In Chapter 4, “z/VM - job roles and basic concepts” on page 67, you learned how to log on, logoff, disconnect, and begin. If you reconnect after having previously disconnected, you may be in CP Mode, as indicated in the bottom right corner of your screen. Example 6-1 illustrates reconnecting into a CP READ state.

Example 6-1 Reconnecting into a CP READ state

```
LOGON ELI
z/VM Version 5 Release 3.0, Service Level 0701 (64-bit),
built on IBM Virtualization Technology
There is no logmsg data
FILES: 0003 RDR,   NO PRT,   NO PUN
RECONNECTED AT 10:46:43 EDT THURSDAY 06/21/07
```

```
                                                          CP READ   VMLINUX6
```

If you find yourself in this situation, execute the **BEGIN** command (or just **B**). This will usually put you into the normal guest operating environment (Linux, CMS, or something else). For the examples presented in this chapter, make sure you are in RUNNING state rather than CP mode. The message in the bottom right corner of your 3270 session should no longer say CP READ; see Example 6-2.

Example 6-2 Using BEGIN to return to CMS mode from CP READ state

```
LOGON ELI
z/VM Version 5 Release 3.0, Service Level 0701 (64-bit),
built on IBM Virtualization Technology
There is no logmsg data
FILES: 0003 RDR,   NO PRT,   NO PUN
RECONNECTED AT 10:46:43 EDT THURSDAY 06/21/07
begin
                                                          RUNNING   VMLINUX6
```

To determine whether you are definitely executing in a CMS environment as well the precise version of CMS that you are running, enter the command **QUERY CMSLEVEL** (or **q cmslevel**); see Example 6-3.

Example 6-3 Using QUERY CMSLEVEL

```
q cmslevel
CMS Level 23, Service Level 701
Ready; T=0.01/0.01 10:48:02
```

The output in this example shows that CMS level 23 is being used and CMS is running.

Keep in mind that if you drop out of CMS mode, you will revert back to CP mode and only be able to use the tools covered in Chapter 5, “Control Program for new users” on page 103.

As a new CMS user, you might try to execute a CMS command that you found in some manual, only to be told the command is not found. This happens most often when you unwittingly revert to CP mode. To ensure you are in CMS mode, execute the command **IPL CMS**, which will return you to the CMS operating environment (in essence, reboot your CMS operating system).

6.2 Getting help from CMS

This section explains the use of the HELP command in CMS. It covers task, component, and command menus, and discusses formatting options. It mentions other ways to invoke help, and describes how to deal with messages. Finally, the section explains how to exit the help system in CMS.

The first CMS command we cover here is probably the most useful: HELP. The CMS command HELP provides information about many standard system tasks. It also provides the syntax and function of all CMS and CP commands, as well as the meaning of error messages that may be reported to users.

Use the HELP command if you cannot remember the format of a command, or if you need information about what a command does. The output will display a substantial amount of the information contained in official IBM documentation.

The HELP command is similar to the **man** command found on Linux systems, and is simple to use. If you type `help` by itself, you are offered a range of further options, along with a brief description of each option. By default, typing `help` on the command line will bring you to the task-based help menu.

The next few sections of this chapter explain how to access the HELP command more directly. To select an entry from the screen, place your cursor under the appropriate word and press Enter. From there you are taken to another help screen, such as a task menu or a command menu.

6.2.1 Task menus

Task menus are the portion of the HELP system that provide an index and description of tasks that you may want to perform, including creating, modifying or changing files, or customizing the CMS installation. Task menus provide you a way to obtain details about specific actions, without requiring you to know the name of the command in advance.

When you reach the deepest level of navigation relating to a particular topic of interest, the bottom “branch” provides the description and proper format of the required command.

To view the list of tasks and components available to you, type:

```
help task
```

Figure 6-1 on page 152 shows the output of a **help task** command.

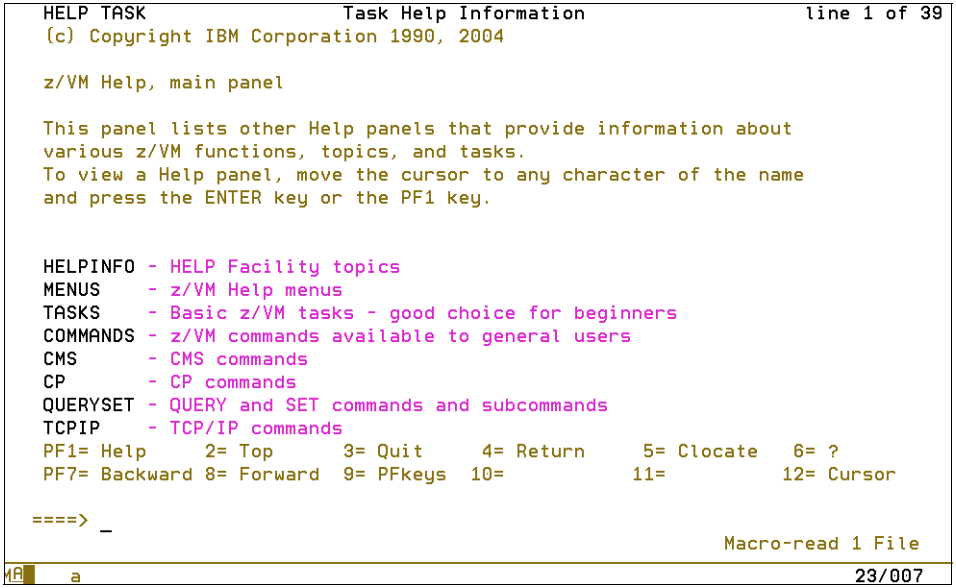


Figure 6-1 Output of help task command

6.2.2 Component menus

Component menus list the names of all the command HELP files available for a specific HELP component.

To display all the command HELP files available for CMS, start at the HELP TASK menu shown in Figure 6-1. Position your cursor anywhere under the word `CMS` and press Enter.

You will reach the Menu Help Information screen shown in Figure 6-2 on page 153, which displays all the command HELP files available for CMS.

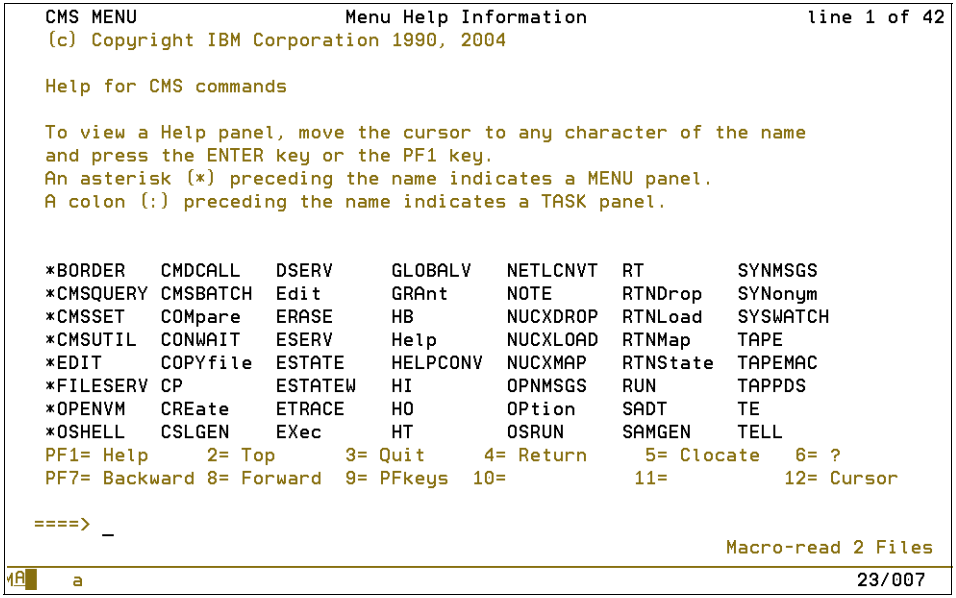


Figure 6-2 CMS component menu

6.2.3 Command menus

In addition to the task menus provided in the help system, there are separate command menus for a number of VM components, including one large menu that covers CP and CMS.

If you select the command menu for CMS from the initial help menu, for example, a panel showing a list of available CMS commands appears. Menu navigation is similar to the task menus; to choose a command, place the cursor over it and press Enter or PF1.

6.2.4 Formatting options

The HELP Facility provides various ways of getting information for a command, depending on your level of expertise and the amount of detail you require for a particular task. Commands can contain three layers of information: BRIEF, DETAIL, and RELATED. Each layer displays a unique level of HELP.

You can display any of the three available layers by specifying the corresponding layering options: BRIEF, DETAIL, or RELATED. You may specify only one layering option at a time. However, after you have requested one layer of HELP

on a specified command, you may toggle (switch) between the other layers available for that command.

BRIEF is the default option, meaning that if you do not specify a layering option, the BRIEF layer of HELP is displayed if it exists. If **BRIEF HELP** is not available for a certain command, **DETAIL HELP** is displayed. The following sections provide more information on the three layers of command HELP.

BRIEF

BRIEF is the first layer of HELP. It is available for many commands. **BRIEF HELP** displays a short description of the requested command, its basic syntax (command without options), an example, and if applicable, a message telling you that either more or related information is available.

If you are in full-screen CMS and request BRIEF HELP, your screen shows the **HELP** command you entered and just below it, displays the BRIEF HELP information in a window that is displayed on your screen. If you are not in full-screen CMS, your entire screen displays the BRIEF HELP information.

The following example shows how your screen would look if you requested BRIEF HELP for the **SENDFILE** command and are not in full-screen CMS.

```
help cms sendfile (brief
```

The output of this command is shown in Figure 6-3.

```
CMS SENDFILE              Brief Help Information              line 1 of 14

SENDFILE

Brief Information

The SENDFILE command lets you send files to other users.  An
abbreviation for SENDFILE is SF.

FORMAT:  SENDFile filename filetype userid (options

EXAMPLE: Greg needs a copy of your file SPEC SCRIPT A.  His user ID is
         Greg.  To send him a copy, enter:
                        sf spec script greg




 PF1= All      2= Top      3= Quit     4= Return     5= Clocate   6= ?
 PF7= Backward 8= Forward  9= PFkeys  10=          11= Related  12= Cursor
DMSHEL241I Press PF11 to get related information.
====> _
                                                        Macro-read 1 File
 A    a                                                          23/007
```

Figure 6-3 Output of help cms sendfile (brief

DETAIL

DETAIL provides a complete description of the command, the command format, an explanation of its parameters and options, usage notes, and error information. For more information regarding DETAIL help, refer to *z/VM: CMS Commands and Utilities Reference*.

This layer of HELP has seven subsetting options: **DESCRIPT**, **FORMAT**, **PARMS**, **OPTIONS**, **NOTES**, **ERRORS**, and **ALL**. By specifying subsetting options, you can display one or more particular sections of the detail help.

ALL is the default option, meaning that the entire detail help is displayed. It is possible to change the default option, but if you do so, you will need to specify **ALL** as the subsetting option to display the entire detail layer. For more information about the subsetting options and the **DEFAULTS** command, refer to *z/VM: CMS Commands and Utilities Reference.*

For example, to display the entire DETAIL layer of the **SENDFILE** command in the CMS environment, type in the following command:

```
help cms sendfile (detail
```

```
 CMS SENDFILE              All Help Information                  line 1 of 807
(c) Copyright IBM Corporation 1990, 2004

 SENDFILE

 >>--.-SENDFile-.--.-! Choice A !-.------------------------------------------><
     '-SFile----'  !-! Choice B !-!
                   '-! Choice C !-'

 Choice A:
        (1)
    .-(------------------------.
 !--+--------------------------+-----------------------------------------------!
    !   (2)                    !
    '-(-----! Options !--.---.-'
                         '-)-'

 Choice B:
              .-*--.          <------------------------------<
 PF1= Brief    2= Top      3= Quit     4= Return     5= Clocate   6= ?
 PF7= Backward 8= Forward  9= PFkeys  10=          11= Related  12= Cursor

====> _
                                                          Macro-read 1 File
A   a                                                              23/007
```

Figure 6-4 Output of help cms sendfile (detail

RELATED

The help information entries for some commands allow you to select help entries for similar commands; this is known as RELATED help.

For example, suppose you want to remove a file from your rdrlist. After reading HELP ERASE, you realize that **ERASE** is not the correct command. Instead, using the RELATED layer of the **ERASE** command will let you easily access the HELP file for the correct command, **DISCARD**.

When you request RELATED HELP on the **SET** or **QUERY** commands, the screen lists and briefly describes all the **SET** and **QUERY** operands available for the system component. You can directly access HELP information on any of the displayed operands from these menu screens by positioning the cursor on a particular operand and pressing Enter.

To display the RELATED layer of the **ERASE** command in the CMS environment, enter the following command; Figure 6-5 displays the output of the command.

```
help cms erase (related
```

```
CMS ERASE              Related Help Information                line 1 of 20
 Related Information

 For RELATED information on removing files or parts of files from
 your virtual machine, place the cursor under the topic of your choice
 and press ENTER or the PF1 key.


 DELETE    - Removes one or more lines from
             a file while using XEDIT.

 ERASE     - Removes files from your minidisk
             or SFS directory.

 PURGE     - Removes spool files from your
             reader, printer, or punch.

 DISCARD   - Removes files from "list-type"
             CMS command environments, such
 PF1= Help     2= Top      3= Quit     4= Return     5= Clocate   6= ?
 PF7= Backward 8= Forward  9= PFkeys  10= Morehelp  11= Brief     12= Cursor

====> _
                                                        Macro-read 1 File
MA    a                                                          23/007
```

Figure 6-5 Output of help cms erase (related

Other help options

There are five other options that affect the display of **HELP**: **SCREEN**/**NOSCREEN**, **TYPE**/**NOTYPE**, and **EXTEND**. Briefly, these other options control the display of files and error messages and the search order of commands. For complete descriptions of these options, refer to *z/VM: CMS Commands and Utilities Reference*.

6.2.5 Other ways to get help

You can also get help for a specific command directly, without going through the various forms of help navigation discussed so far. If you know the command you want to see help for, or want to navigate directly to it, specify the command type as shown here:

`Help CP MENU` - Displays the full screen menu of available CP commands

`Help CP command` - Displays the format of the specified CP command

`Help CP command` - Displays the task menu specified

6.2.6 Dealing with error messages

Sometimes, when you perform a z/VM task, the system responds with a message. You can use HELP for messages to find out why the message was produced, and perform any necessary corrective action.

The HELP files for messages display the message text, an explanation of why the message was displayed, the system action, and a user action. For example, suppose you receive the following error message:

```
DMSERD107S  Disk 'A'(191) is full
```

You wonder whether this message is significant, so you want more detailed information. For this particular error indicator, issue the help command, followed by the initial identifier present in the message `DMSERD107S`, as shown:

```
HELP DMSERD107S
   or
HELP DMS107S
```

Note: The module identifier (characters 4 to 6 of the message identifier) is ignored by **HELP**, so you do not need to enter it.

For example, to display information about message DMSHLP002E, you can enter any of these commands:

```
help msg dmshlp002e
help msg dms002e
help dmshlp002e
help dms002e
```

If you receive a message without a message ID, it could be because you have issued the **CP SET EMSG TEXT** command to display only message text (or an application program might have issued the command).

To obtain information about a message with no message ID, you will need to use help facilities beyond the scope of those built in to CMS (such as the PDF or BookManager® version of the appropriate messages book).

6.2.7 Caution when using HELP

Some commands may not work as expected in HELP mode. For this reason, we recommend that you do *not* enter commands on the command portion of the HELP screen where they are displayed.

In general, use the HELP Menu only for help, and not as a command prompt.

6.2.8 Exiting the HELP system

To exit the help system in CMS at any time, use PF3 or type `quit` on the command line. Note that you may need to issue the **quit** command multiple times if you have descended into the HELP Menu hierarchy and want to back up one level at a time.

As an alternative, you can use PF4 to completely leave the HELP system with one keystroke. The HELP navigation functionality is similar to z/OS.

6.3 Using truncations and abbreviations

To make entering commands on the keyboard more convenient, z/VM allows you to enter many commands and operands in a shortened form of truncations or abbreviations. This section explains truncation and abbreviation formats and usage.

You *truncate* a command or operand name by dropping one or more letters from the end of the name. The syntax box for each command shows the truncations you can use for each command and its operands. For more information about this topic, refer to *Commands and Utilities Reference*, SC24-6081.

Note the following points:

- If a letter is shown in upper case in the syntax box, it means that you *must* enter that letter when you enter the command from your display.

- If a letter is in lower case in the syntax box, you can omit it when you enter the command.

For example, the syntax box for the QUERY command shows the command as:

```
Query
```

This means that you can enter the QUERY command in any of the following forms:

```
query
quer
que
qu
q
```

The minimum acceptable truncation is `q` (you can use upper case or lower case), but any of these forms of the command will be accepted.

Abbreviations are also shorter forms of commands and operands, but they are not formed by simply dropping letters from the end of the command name. Instead, command syntax boxes show the acceptable abbreviations for command names.

The abbreviations appear below the full name of the command. Operand abbreviations are listed in the operand descriptions following the syntax box.

For example, the syntax box for the MESSAGE command shows:

```
Message
Msg
```

This means that you can use any of the following formats:

- Truncate the command to a minimum of `M`
- Use the abbreviation `MSG`
- Truncate the abbreviation to `MS` or `M`

Thus, CP accepts all of the following forms of the MESSAGE command:

```
message
messag
messa
mess
mes
me
m
```

or:

```
msg
ms
m
```

Throughout this book you may see commands presented in full, or in shortened form. In general, when a command is introduced, the minimum acceptable truncation will be shown in upper case letters, with the remainder shown in lower case letters. (This is the same convention that is used in the HELP facility display.)

6.4 Full screen CMS

At this point you have learned the basic concepts of CMS, and you know how to execute commands in the normal CMS shell. This section describes the full screen version of CMS, and discusses how to enable full screen display, how to leave full screen CMS, and receiving messages in full screen mode.

Although it is not necessary to use CMS in full screen mode, doing so offers advantages that may be useful to you in the future. The most useful function in full screen CMS is the ability to scroll back to see previously entered commands and system responses. (Full screen CMS also offers many other features, but they are beyond the scope of this publication.)

Enabling the full screen display

To enable full screen functionality, enter the command **`SET FULLSCREEN ON`** from the command line. The resulting screen will look similar to Figure 6-6 on page 161.

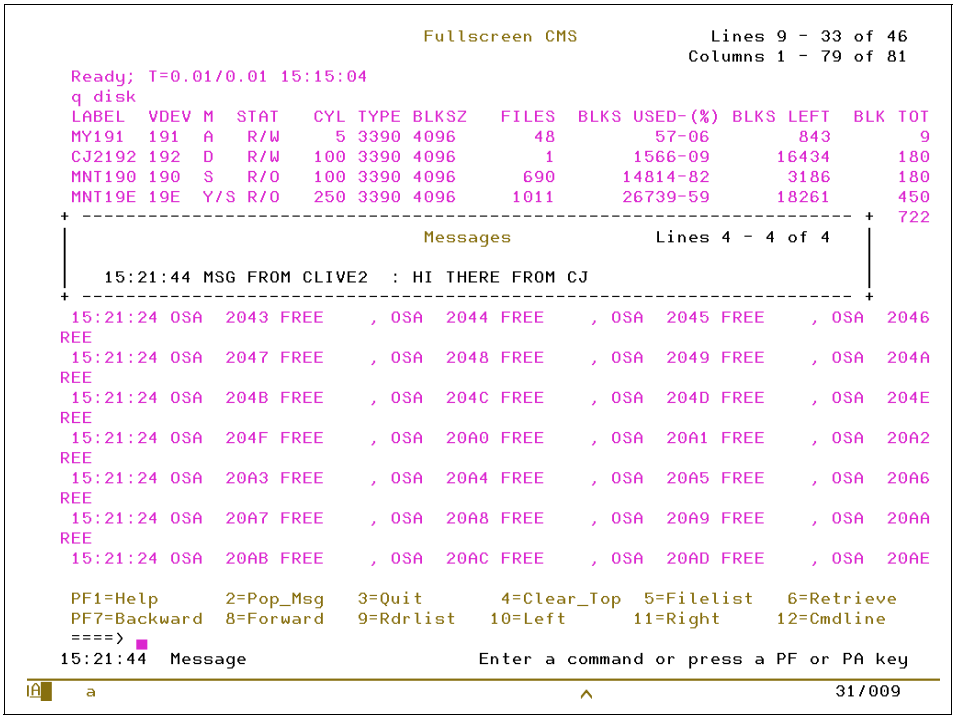


Figure 6-6 Full screen CMS display

Note the following areas in the figure:

- `PF Key Definition Area` displays the CMS PF keys and their functions.
- `Command Line` is the area where you can type commands (as done previously on the normal CMS interface).
- `Status area` may contain indicators regarding outstanding messages or warning states about your virtual machine.

To scroll a window that is connected to the specified virtual screen, use the PF7 and PF8 keys. PF7 scrolls back one window display. PF8 scrolls forward one window display.

Leaving full screen CMS

To leave full screen mode at any time, you have two options. You can issue either the command SET FULLSCREEN SUSPEND or the command SET FULLSCREEN OFF, as explained here:

- **`SET FULLSCREEN SUSPEND`** allows you to resume your full screen session and continue where you left off at a later point in time.

- **SET FULLSCREEN OFF** starts a completely new session when you reenter the full screen (all existing output on the screen becomes immediately occluded).

Messages in full screen mode

In full screen CMS mode, you may see items displayed in the MESSAGE window. During normal, non-full screen CMS operation, messages cause the screen to be cleared. After reading the message, you must press <CLEAR> in order to return to the original screen.

In full screen mode, however, when a message arrives then an alarm sounds and the status area is updated to tell you that a message is waiting. If you press Enter, the message window will pop up on the current screen. This window will remain displayed until you explicitly remove it by using PF4 Clear_top.

6.5 Examining disks

By now you have executed a few commands, been exposed to the CMS working environment, and seen some output. This section discusses the topic of files and file systems used by your guest. It covers linking, CMS formatting, accessing disks, the A disk, and how to deal with running out of disk space.

6.5.1 Your disks

Chapter 5, “Control Program for new users” on page 103 describes DASD and explains how to query the attached devices. DASD refers to disk storage space, a concept that is similar to the physical hard disk drive found in a personal computer.

To display a list of the DASD devices assigned to your guest (and their associated virtual addresses), execute the command **QUERY VIRTUAL DASD** (Example 5-21 on page 121 shows a typical list).

You may recall that attached devices are similar to having plugged in a new hard disk to a desktop computer. That is, the disk is there, but it is not readily useful to the CMS operating system.

DASD is sometimes given as an entire pack (or disk), but it is often given to a guest in the form of a minidisk. A *minidisk* is a section of a DASD pack that is dedicated to a specific guest. In this way, a full DASD pack is conceptually similar to a whole disk drive, and a minidisk is like a partition.

6.5.2 Linking

If your files are stored on minidisks, you will need to link to the minidisks of other users in order to share files. Chapter 5, “Control Program for new users” on page 103 discusses what linking is and how to perform the link operation.

As explained, only one user can *own* a minidisk, but there are many occasions that require users to share data or programs, so z/VM allows you share minidisks, on either a temporary or permanent basis, by linking.

You use the **link** command to link to the minidisks of other users. To verify that the command has been processed, enter the **CP QUERY DASD** command, which will verify whether your devices exist where you think they should.

Linking a disk is like plugging in a hard disk or USB key to a computer. If the disk is truly new and unformatted, it will need to be formatted. More commonly, the disk will simply need to be accessed if it already contains content. Formatting and accessing tasks are explained in more detail in 6.5.3, “CMS formatting disks” on page 163 and 6.5.4, “Accessing disks” on page 165.

Typically when you link to another CMS user’s disk, you will want to access it *read only*. Remember, the disk probably contains their data, which generally you would not change.

If the disk is new and belongs solely to your virtual machine, it is good practice to read the formatting section. To simply view someone else’s data on the linked disk, you can generally skip ahead to the access section.

6.5.3 CMS formatting disks

When you obtain new DASD, it is initially unformatted for CMS. (Remember, formatting is only necessary when you are dealing with “fresh” DASD that has been assigned to you by your system administrator, not DASD that you are linking from someone else.)

For disks that you own, therefore, either you or the system administrator have to format them in order for CMS to be able to store files on them. CMS only needs to format the disks if you intend to use the disks for VM-related tasks.

Note: If you plan on installing Linux on the minidisk, you do not need to format, because Linux uses a different disk format from CMS.

To format, or put a CMS readable file system on the disk, use the command **FORMAT VDEV MODE**. Before you can use any new minidisk, you must format it. This

applies to new minidisks that have been assigned to you, and to temporary minidisks and virtual disks in storage that you have defined with the **CP DEFINE** command.

When you enter the **FORMAT** command, you must use the virtual device number you have defined for the minidisk and assign a file mode letter. The mode letter will be the unique letter name of the formatted disk when the format is complete (mode letter is covered in more detail in 6.5.4, “Accessing disks” on page 165).

Here is an example **FORMAT** command:

```
format 291 c
```

CMS will then prompt with the following message:

```
DMSFOR603R FORMAT will erase all files on disk C(291).
Do you wish to continue? Enter 1 (YES) or 0 (NO).
```

You respond by typing either a number (1 or 0) or a word (YES or NO), as appropriate. If you respond with the Yes option, then CMS asks you to assign a label for the minidisk, which you select.

A valid label is considered to be any combination of one to six alphanumeric (0 to 9 and A to Z) characters. Be aware that the use of *fewer* than six characters will cause blanks to be filled in place of the missing rightmost characters. Conversely, the use of *more* than six characters results in only the first six characters from the left being used.

When the message DMSFOR605R Enter disk label: is displayed, you respond by supplying a minidisk label. For example, if this is a temporary minidisk, you might enter:

```
temp
```

CMS then erases all the files on that minidisk, formats it for your use, and displays the messages shown in Example 6-4.

Example 6-4 Output showing formatting is complete

```
DMSFOR733I Formatting disk C
DMSFOR732I 10 cylinders formatted on C(291)
Ready; T=0.15/1.60 11:26:03
```

After the format of the disk is complete, enter the following command:

```
query disk
```

This will return a display similar to Example 6-5 on page 165, including the newly formatted disk.

Example 6-5 New TEMP disk shown in query disk output

```
LABEL  VDEV M   STAT CYL TYPE BLKSIZE FILES BLKS USED-(%) BLKS LEFT BLK TOTAL
  DL0191 191  A   R/W   20 3380 2048    227        4554-84        846      5400
  TEMP   291  C   R/W   10 3380 4096      0           6-00       1494      1500
  IDTOOL 192  E/A R/O 120 3390 1024 1797          55424-93       3976     59400
  YDISK  19E  Y   R/O 200 3390 4096 1443          28661-80       7339     36000
```

As mentioned, only use the FORMAT command to format CMS minidisks, that is, minidisks you are going to be accessing from CMS. Also note that the FORMAT command gives you a choice of physical disk block size as an option.

For more information about the FORMAT or QUERY DISK commands, refer to *z/VM: CMS Command and Utility Reference*.

6.5.4 Accessing disks

After a new disk is formatted, or after you have been provided link access to someone else's existing disk, the disk must be *accessed*. (Formatting simply makes the disk ready to be read from, and have data written to it.)

The access actually places the disk somewhere in your file system with a name, so that it can be accessed. CMS uses letters, known as *mode letters*, to identify different disks.

Up to 26 disks (one for each letter in the English alphabet) may be accessed at once. Each accessed disk can be thought of as a "mount point", from a Linux point of view.

This is rather similar to how Windows makes formatted drives available to users. On your Windows PC there is typically a C drive which is in actuality some hard disk. There is also typically an A drive, which is a floppy drive. Often a D drive may be present, and is some kind of CD-ROM or DVD drive.

When selecting your access mode, it does not matter which letter you use as long as the letter is not already being used by another disk. To obtain a list of all of your accessed disks, execute the **QUERY DISK** command. Simply pick an available letter that you will remember.

Accessing a disk is done by executing the command **ACC VDEV MODE**. Simply replace VDEV with the virtual device address of the minidisk you want to access, and replace MODE with the file mode you want to use.

To determine the VDEV portion of the command, you need to know the virtual device address of the disk. To find out the virtual device address, use the CP

command **QUERY VIRTUAL DASD**, as previously discussed. That command lists all of the DASD available to a virtual machine.

After you access the disk, issue a new **QUERY DISK** command to display your new entry with the access mode letter you chose. To release a disk (which is the opposite of accessing a disk), use the command **RELEASE MODE**.

Disk access conventions

Now you have learned the basic concepts surrounding disk access, so this section describes some of the conventions related to disk access.

A CMS user will usually have at least two disks (probably minidisks) available. Typically with CMS, a user has a pair of virtual disk with addresses 191 and 190. Note the following points:

- The 191 disk is conventionally accessed with mode A.
- The 190 disk is accessed as S. The 190 minidisk is sometimes known as the “CMS system residence volume”, because it is where the CMS nucleus resides on disk.

As an exercise, enter **IPL 190** from your user ID. Notice that it brings up the `Ready` message, just the same as **IPL CMS** does. The difference now is that you have a non-shared copy of CMS running in your virtual machine.

6.5.5 Your A disk

CMS assumes that you have disks at addresses 190, 191, and 19E. It also looks to see if you have a disk at address 192. When CMS loads, these disks are accessed as follows:

- 191 becomes the A disk
- 192 becomes the D disk (if it exists)
- 190 becomes the S disk
- 19E becomes the Y/S disk

This will only succeed if your guest has DASD with a virtual address of 191. The A disk is a work disk for a user’s permanent file storage. Generally the A disk of CMS users is the only writable disk they have access to. In many ways, the A disk is like a Linux home directory; that is, it is completely owned and operated by a particular virtual machine user.

The A disk is exceptionally important because when CMS is loaded, it looks for a file named PROFILE EXEC on your A disk and attempts to execute that file (the PROFILE EXEC file is discussed in more detail in 6.8, “The PROFILE EXEC” on page 197).

The PROFILE EXEC is similar in concept to a bash profile from a Linux system. That file is often modified to customize the CMS instance for a given user, or even to load another operating system like Linux.

Note: If you have logged on to your guest for the first time, your 191 disk (the A disk) may not have been formatted. Check whether you have a suitably formatted A disk by executing the command **LISTFILE**. If you see a file listing after executing the LISTFILE command, then you can continue with this chapter.

However, if you receive the error message `HCPCMD001E Unknown CP command: LISTFILE` then you may need to format your 191 disk before continuing. In that case, use the command **FORMAT 191 A** to place a VM readable file system on your A disk.

The LISTFILE command is explained in more detail in “LISTFILE command” on page 173.

6.5.6 Running out of space

Each virtual disk is divided into blocks that are usually 4096 bytes in size (although some disks, such as the help disk, use smaller blocks to save wasted disk space from the number of small files on them).

The minimum size of each file is one block. The maximum is limited by the size of the minidisk. Each user normally has read access to a number of these types of disks. But often, users run into trouble when they have no space left to write their own files.

For this reason, it is important to know how much space you have left on one of your accessed disks. You can get this and other useful information by using the **QUERY DISK** command.

Note: **QUERY DISK** is different from the **Q DASD** command. QUERY DISK is a CP command for *devices*. Q DISK is a CMS-only command used for viewing your *accessed volume*s.

It is possible to completely fill up a disk, such as your A disk. In this situation, the system does not give you more space automatically. You must take some action to get more space or erase unnecessary files.

To acquire more space you could attach a new DASD volume, and then format and access it. Another option could be to link to another disk you have write access to. (There is no such thing as a defragment on CMS files.)

6.6 Working with files

At this point you have learned about disks and seen how to access them. In this section, we discuss the files that reside on disks, in particular the PROFILE EXEC file.

6.6.1 The CMS file system

The CMS file system was designed to be fast. CMS file access input/output (I/O) operations were planned to be as efficient as possible. CMS files are created with names, and space allocation is *dynamic* within the minidisk, similar to any modern file system on a personal computer. This is in stark contrast to earlier file systems, in which space had to be *explicitly* allocated. Today, files can be of practically any size (in number of records and characters per record), and are usually accessed sequentially.

A CMS file is similar in function to the files you would use in Microsoft Windows or GNU/Linux. The main difference lies in how the data is organized within the file itself. CMS files are arranged in fixed size rows (known as *records*), instead of being a simple stream of bytes. The size of these rows is given by the “record length” of the file.

For example, a file with a record length of 80 and 24 records is essentially 24 lines of data, where each line is made up of 80 bytes or characters. This distinction is not especially important until you want to transfer file between CMS and another operating system.

Furthermore, the CMS file naming system follows simple conventions. The following section examines each of these conventions.

6.6.2 Filename structure

The FILE is the basic unit of data in the CMS filing system. z/VM files are named with 3-tuple, or three distinct parts. Unlike other operating systems, the unique separator is a space. The first part is known as the *name*, the second part is known as the *extension* (or *type*), and the last part is known as the *mode*. For example, a completely valid file name that exemplifies the naming convention is FILENAME FILEEXTN M.

Next, we explain the file parts in more detail.

Name

The name portion is limited to 8 case-insensitive alphanumeric characters. Filename is also known as fn in some documentation.

File type

The file type (commonly referred to as a “file extension type” in other operating systems) is basically an extension of the filename to indicate the file’s intended purpose. The extension is also limited to 8 case-insensitive alphanumeric characters. The file type is also known as ft in documentation.

File mode

The last component of the z/VM file naming convention is a single letter combined with a single number. The alphabetic character represents the access mode letter of the disk on which the file resides.

In many ways, the concept of the mode is similar to the concept of a “path” in Linux systems. That is, several files may have the same name and extension, but reside on separate accessed disks. It is important to note is that the file mode can be *temporary* within a session.

For example, assume you have some files on a disk accessed as A. Each file name would be something like FOO TXT A. If you access those files with another mode, in effect the file names on that disk have changed! For instance, accessing those files with mode C would make the filenames available as FOO TXT C.

In general, anything that will be accessed frequently should have a mode letter reserved for it that you will remember. The second part of the access mode is a number in the range 0 to 6. (A discussion of the purpose of the numerical portion of the access mode is beyond the scope of this book, and does not need to be specified explicitly during day-to-day operations.)

The fileid

The name, type, and mode letter taken together form a full CMS file identifier or fileid. Each fileid is unique. No two files will ever have the same name, extension, and mode 3-tuple.

Here are some possible fileids:

```
ABCDEFGH IJKLMNOP A1
X        Y        A2
ALL      NOTEBOOK A0
```

```
PROFILE  EXEC     A1
PROFILE  EXEC     B1
PROGABC  COBOL    C1
PAYROLL  ASSEMBLE C1
BF71A    NAMES    A0
XYZZY    123      C1
MY       DATA     A6
BF71     SCRIPT   A1
PROFILE  XEDIT    A2
00000001 DATA     A1
LINK     EXEC     Z1
ELEPHANT STEW     A1
```

As you can see, you do not have to use the filename and filetype in a rigid way. However, certain filetypes have a particular meaning to the system. Also, there are no “reserved” names as such, but certain identifiers are significant to certain programs.

For example, the COBOL compiler program will only compile program source stored in a file with a filetype of COBOL. Likewise, the file editing program XEDIT assumes certain characteristics for files because of the filetype you supply (items such as record length and format).

In addition, the notion of executable files exists. CMS allows the use of command procedures, with file extension EXEC, to simplify standard functions or add new functions. These executable files can be anything from simple lists of frequently executed commands to complex “installation-utility” programs.

A procedure is invoked simply by typing its filename. If it is a normal disk-resident procedure, its filetype must be EXEC, and it must be on a disk accessed by CMS.

Table 6-1 lists and describes CMS filetype conventions.

Table 6-1 Filetype conventions when using CMS

File type	Typically use
ASSEMBLE	Assembler source code
COBOL	COBOL source code
DIRECT	Directory-related files
EXEC	Generic executable program
FORTRAN	FORTRAN program source code

File type	Typically use
LISTING	High level assembler intermediate representation for debugging
MODULE	
NETLOG	
PASCAL	PASCAL program source code
REXX	Scripts written in REXX
TEXT	Object code (as in the text and data segment of a program)
XEDIT	Normal text data files

6.6.3 Listing

The **FILELIST** command is the equivalent of **mc** from the Linux environment. To display a simple list of files that are on an accessed disk (like your A disk), use the command **FILELIST * * A**. This command tells CMS that you want to see a list of all files on the A disk.

The first asterisk (*****) is a wild card that can be used to match anything for the file name. The second * matches any extension. The A in our example matches only items from the disk accessed as A. Note that the wildcard ***** is valid in all three filename fields (name, extension, and mode).

The abbreviation **filel** is the minimum acceptable abbreviation for the FILELIST command. The output of the **FILELIST** command identifies several files and provides pertinent information about them, as listed here.

Size=nnn Total number of files listed
File Identifier Comprised of the filename, filetype and filemode
Format The format of records in the file (**F**ixed or **V**ariable)
Lrecl The maximum length for this file (in characters)
Records The number of records in the file
Blocks The amount of space this file takes on disk
Date and time Date and time stamp of the last change to this file.

When using the **FILELIST** command, there may be more files in the list than can be displayed on the terminal at one time. Table 6-2 on page 172 lists and describes useful commands you can use to work with the screen output.

Table 6-2 Useful list commands

Command	Action
/	Type against any line to bring that line to the top of the display.
PF8	Scrolls the screen down the display.
PF7	Scrolls the screen up the display.
SNAME	Sorts alphabetically by name, type and mode.
STYPE	(PF4) Sorts files by type, name and mode.
SDATE	(PF5) Arranges the files with the newest at the top (this is the default display).
SSIZE	(PF6) Sorts the files in decreasing order of size.

Example 6-6 illustrates a simple execution of the filelist program. Note that the PF keys are displayed at the bottom of the screen for reference.

Example 6-6 Sample execution of the FILEL command to list files interactively

```
filel
 ELI      FILELIST A0  V 169  Trunc=169 Size=3 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      MYFILE   TXT      A1 F         80         20          1  6/07/07 14:23:40
      ELI      NETLOG   A0 V        103          1          1  6/01/07 11:27:44
      PROFILE  EXEC     A1 V         31         14          1  6/01/07 11:26:07









1= Help       2= Refresh  3= Quit   4= Sort(type)  5= Sort(date)  6= Sort(size)
7= Backward   8= Forward  9= FL /n 10=            11= XEDIT/LIST 12= Cursor

====>
                                                          X E D I T  1 File
```

The command **FILELIST** is an interactive program that allows you to search for files interactively. Sometimes, however, using this command is inefficient. For instance, if you only need to see the files listed—without further need for sorting or interaction—then using **FILELIST** might be “overkill”. Instead, it might be more useful to use the **LISTFILE** command.

LISTFILE command

The CMS command **LISTFILE** produces output on screen and returns you to the normal CMS shell immediately; see Example 6-7. The **LISTFILE** command is similar to the commonly used **ls** command found on GNU/Linux systems.

Example 6-7 Sample execution of LISTFILE to statically list files

```
Ready; T=0.01/0.01 11:18:13
listfile
ELI      NETLOG   A0
MYFILE   TXT      A1
PROFILE  EXEC     A1
Ready; T=0.01/0.01 11:18:16
```

Take a moment to use each of these commands to compare how their usage varies. FILELIST and LISTFILE are suitable for different types of task.

6.6.4 CMS search order

When you enter a command in the CMS environment, CMS uses the search order described here to locate the command. When found, CMS stops the search and processes the command.

Note: If you have executables of any sort in storage or on an accessed file mode, CMS treats them as commands. These commands are known as *user-written commands*.

An algorithm used when you enter a command into CMS determines which executable or file you are referring to when the entire command is not explicitly typed out. The basic algorithm is shown here:

1. Search for an executable already in storage.

   Sometimes system administrators have placed commonly used programs in memory so that they are loaded faster by the system users.

2. Search the disk sequentially using standard CMS search order for an EXEC file with the specified command name.

   The standard search order is to begin at disks accessed as A and progress alphabetically towards Z.

If the command or filename was not specified completely, the CMS search algorithm performs the first pass of its search by looking only for files of type exec. If no execs are found, a search for synonym exec files is performed in the same fashion.

If no exec or synonym exec is found the algorithm repeats, but this time looks for files of type “module” (as well as any synonyms with type module, as done with execs). If the algorithm fails to locate a match, the command is deemed unknown to CMS, and is relayed to CP for further processing.

For more information about the CMS command search order, and about searching for a translation or synonym using the SET TRANSLATE or SYNONYM commands, refer to *z/VM: CMS Command and Utility Reference*.

6.6.5 Searching

By default **FILELIST**, as well as **LISTFILE**, will display all files on your A disk. But the **FILELIST** command has additional options which allow you to search other disks, as well as other means to refine searches. The full format of the FILELIST command is shown here:

```
FILEList filename filetype fm ( options
```

Many CMS commands that manipulate files allow you to enter the file name or file type fields or both as an asterisk (*), indicating that *all* files of the specified file name or file type are to be modified. FILELIST is one example of this type of command. An asterisk(*****) can be used to represent any number of any characters. For example, listing all files with a file type of TEST on file mode A is performed as shown here:

```
filel * test a
```

Several commands allow you to perform operations on a group of files that have a file name or file type that begin with the same character string. Making use of the asterisk (*) with these commands is commonplace in z/VM usage.The same commands allow you to use the percent sign (%) as a place holder to mean any single character.

The filemode can also be specified as shown in Example 6-8.

Example 6-8 Sample FILEL command use with pattern-matching functionality

```
FILEL * * D
FILEL FRED * *
```

Letters and asterisks can be combined to refine the search even further, as illustrated in Table 6-3 on page 175.

Table 6-3 Sample file search commands and associated output

Command	Result
`filel * exec *`	Produces a list of all files with type `exec` on any accessed disk
`listfile t* assemble`	Produces a list of all files on file mode A with file names beginning with `t` and having the file type of `assemble`
`listfile tr* a*`	Produces a list of all files on file mode A with file names beginning with `tr` and having file types beginning with `a`
`listfile %%% stock`	Produces a list of all the files on file mode A whose file name is three characters in length and whose file type is `stock`
`listfile t%% cat`	Produces a list of all the files on file mode A with a three-character file name beginning with `t` and having a file type of `cat`, for example: ► top cat ► the cat ► tom cat
`filel p%q*  * g`	Any name starting with `p`, on the `g` disk with name starting `p` something `q`.
`listfile %tr*s *ri%%`	This command would produce a list of files with: ► File name starting with a character, followed by `tr`, and ending with `s` ► File type starting with `ri` or one or more characters preceding the `ri`, followed by two characters. ► File mode A, since unspecified.

6.6.6 File management commands

Just like any operating environment, CMS has commands for working with files. Here is a list of the more common file management commands used by CMS users. For context, and when possible, the command is presented along with an analogous command from other operating systems.

Creating

Typically files are created in CMS by using the text editor when you have new content to save. In some cases copying a file is the best way to create a new file. In general,there is no equivalent CMS command for the “touch” command found on UNIX systems. (The text editor XEDIT is covered in detail in 6.7, “Editing files with XEDIT” on page 183.)

Copying

The **COPY** command copies a file from one location to another. Simply specify the source and destination filenames file extensions and file modes. Though this might sound a bit trivial, it is very important to the CMS user.

Typically, a CMS user has read access allowing them to look at, and list, files that are on disks that are read only. In order to change those files, a CMS user must first copy them to a read-write disk. The command which allows you to do this is the **COPYFILE** command, as illustrated here:

```
COPYfile fileid1 (fileid2 .. fileidn) ( options
```

A simple example is to issue **copyfile duane txt a secret data d**. This makes a copy of the file duane txt on your A disk and stores it as “secret data” on your D disk. This command works the same way as a Linux **cp** command.

However, you do not always have a D disk, or it may not always allow you to write files onto it. In such cases, you will receive an appropriate error message. (Note that the original file will not be *not* erased.)

The **COPYFILE** command is one of the most powerful CMS commands. In addition to the trivial copying operations described, this command allows you to specify multiple input files, copy specific columns to new positions, insert fixed data in the file, select out ranges of records to copy, and perform other actions. The command **COPY MARK TXT S = = A** will copy the file MARK TXT from the system (or S) disk to your A disk.

Another example invocation of the command is **copyfile duck list c finch = = warbler = a birds file a (append**. This example combines the files duck list and finch list on your C disk, and the file warbler list on your A disk, and adds the new combined file to the end of the file birds file a, which already exists. This form of the command is similar to using the **cat** and **>>** combination, for those familiar with Linux operating systems.

The **COPYFILE** command has additional useful options. Here is an example command that copies files from your S disk to your A disk, while ensuring that the time stamp on the system file is preserved:

```
COPY ALL XEDIT S = = A (OLDDATE
FILE 'ALL XEDIT A2' ALREADY EXISTS -- SPECIFY 'REPLACE'.
```

You will note that this example illustrates a common error that occurs when the output fileid already exists (for instance, after a previous COPY operation where the user neglected to keep the time stamps). In such a situation, you must reissue the COPY command with another option, as shown:

```
COPY ALL XEDIT S = = A (OLDDATE REPLACE
```

The open parenthesis (in all these examples is a CMS convention, and things that change the default operation of a command (that is, options) are placed after it. Note that you do not need to add the closing parenthesis).

Renaming

If you want to change the name or type of a file which you have already created, you can use the **RENAME** command as shown:

```
REname fn ft fm  nfn nft nfm ( options
```

For example, the command **rename kyle exec a old exec a** changes the name of a file on your A disk from kyle exec to old exec. This instance of the rename command is similar to the **mv** command in Linux.

Another example is **rename data eli b data = =**. Notice the use of the equal (=) sign. This is a useful convention which applies to all commands where *more* than one fileid must be specified. The equal (=) sign is used as a short way to indicate the same data for filename, filetype, or filemode as already specified earlier in the command.

Note: When using this shorthand, keep in mind that it can sometimes make a command harder to read. So use it judiciously and carefully, and remember that = means part of the fileid is to remain unaltered in the new name.

Comparing

When comparing two disk files, you may want to use the **COMPARE** command. For example, to ask CMS if the file useless data d and the file useless file a are identical, simply invoke the command **compare useless data d = file a.** Upon receiving this command, CMS responds as shown:

```
COMpare fileid1 fileid2 ( options
```

The **COMPARE** command is similar to the **diff** command in Linux environments.

Erasing

The **ERASE** command removes files from disk storage, releasing space for other uses. It gets rid of a file, and the space that the file occupied is immediately reusable. The command works in a similar way to the Linux **rm -f** command. The syntax of the **ERASE** command is shown:

```
ERASE fn ft (fm) ( options
```

An example use of the **ERASE** command is **erase frank tdata a**. This would delete the file frank tdata on your A disk.

Important: This command provides an immediate, *irrevocable* deletion of the file data. **ERASE** will assume that the file is on your A disk unless you specify otherwise.

Printing

The **PRINT** command will send a copy of a disk file to your virtual printer. The syntax of the **PRINT** command is shown:

```
PRint fn ft (fm) ( options
```

However, this assumes that all necessary information has been set by default and that it is a locally attached printer. However, printing today is much more complex than it used to be, and it is now probable that print output will be sent to a network-attached printer on a TCP/IP network.

Here we describe three command options for printing on a z/VM system:

RSCS Remote Spooling Communications Subsystem (RSCS) manages the sending of files to remote printers. Normally you would spool your virtual printer to RSCS, and it would then direct the output to a default printer. It is possible to attach routing information and other information by using the TAG command. RSCS also provides an advanced LPR function.

PSF/VM Print Services Facility™ (PSF/VM) is an optional product on z/VM that can be used to take files from the z/VM spool and format them for advanced function printing.

LPR Those familiar with printing in the TCP/IP environment will be familiar with this command. The z/VM TCP/IP implementation uses a daemon to provide this printer support to users.

6.6.7 CMS Shared File System

Shared File System (SFS) is an additional file system that is shipped with z/VM, and it offers several advantages over the normal CMS file system. Table 6-4 lists and compares these file systems.

Table 6-4 Compare and contrast Shared File System and normal CMS file system

Shared File System	CMS file system
Sharing at file level	Sharing at mindisk level

Shared File System	CMS file system
Minimum allocation is zero 4096 byte blocks.	Minimum allocation 1 cylinder
Hierarchical file system	Flat file system
Allocation can be changed dynamically	Bigger minidisk required
Recovery at file level	Recovery at minidsk level

SMS is essentially a collection of minidisks managed by a server.

As can be seen in Figure 6-7, there are three main areas of interest:

Control data This is where the definitions of the filepool and its users are kept.

Log data Logs are kept so that, in the event of interruption, files can be rolled back.

User data This where the user data exists and the server will allocate data from here to give to users.

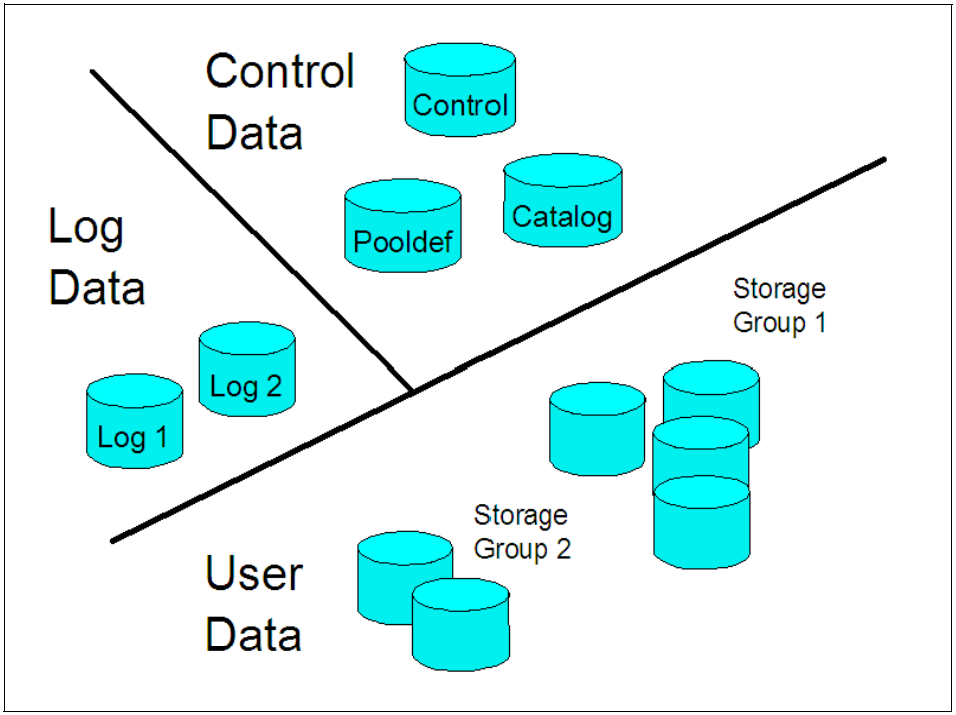


Figure 6-7 Shared File System

With SMS, instead of a minidisk, users are enrolled in a filepool by an administrator and will be given a number of 4096 byte blocks from a storage group that they can use in a similar way to a CMS minidisk.

Users will access the space in the same way that they do a CMS minidisk, but using the filepool name concatenated with their user ID instead of a minidsk device number. For example, they would use VMSYSU:USERID instead of 191 on and the ACCESS command.

However, once accessed, users can use commands to create directories and subdirectories in a similar way as they do on a workstation, creating a hierarchical file structure if needed. Users can then GRANT authority at the file level to allow other users to read or write to data in its file space.

There are several commands, listed and described in Table 6-5, that users should be aware of if they have been allocated some space in the filepool.

Table 6-5 Commands related to SMS

Command	Description
SET FILEPOOL	Use the SET FILEPOOL command to set (or reset) your default file pool. **`set filepool vmsysu`**
QUERY FILEPOOL	Use the QUERY FILEPOOL command to display your current filepool. **`query filepool current`**
QUERY LIMITS	Use the QUERY LIMITS command to see how many 4 K blocks you have available to use. **`query limits`**
DIRLIST	Use the DIRLIST command to list you directories in a similar way that you use FILELIST to list CMS files.
ACCESS	Use ACCESS to assign a mode letter to your file space similar to the way that minidisks are accessed. **`access vmsysu:cmsuser a`**
CREATE	Use the CREATE command to create a directory similar to mkdir on a PC. **`create directory test1`**
GRANT	Use the GRANT command to grant other users access to files or directories.

The entire file system will have an administrator who is responsible for backups and allocation of minidisks to storage groups as required.

z/VM is shipped with three filepool servers:

- VMSERVR and VMSERVS are used by z/VM
- VMSERVU, a sample user file system

For more information about these topics, refer to *z/VM CMS File Pool Planning, Administration and Operation*, SC24-6074, and *z/VM CMS Commands and Utility Reference*, SC24-6073.

6.6.8 Concluding file management

Each of the file operation commands you have learned so far can be entered from the normal CMS command line, or they may be issued directly from the filelist screen. Note that some commands, for instance **RENAME**, require you to manually refresh the screen after performing a file operation when in the FILELIST screen. To do this, press PF2.

If you find a file you are interested in, you can issue commands relating to the file in the section at the left of the display. The command can overlay the information displayed on the screen; no data is lost. A forward slash (/) in a command is understood by CMS to mean the fileid on that line of the screen. Commands not related to files can be typed on the line at the bottom of the screen identified by the large arrow symbol:

```
====>
```

Example 6-9 shows an interactive file listing; ELITEST FOO A is the line to interact with.

Example 6-9 Interactive file listing

```
ELI      FILELIST A0  V 169  Trunc=169 Size=6 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      MESSAGE  LOGFILE  A0 V         71          2          1  6/25/07 16:51:38
      ELITEST  FOO      A1 F         80         21          1  6/25/07 11:38:19
      ELI      NETLOG   A0 V        104          2          1  6/21/07 12:12:53
      PROFILE  XEDIT    A1 V         72         29          1  6/21/07 12:11:29
      ELIFILE  TXT      A1 F         80         20          1  6/07/07 14:23:40
      PROFILE  EXEC     A1 V         31         14          1  6/01/07 11:26:07




1= Help       2= Refresh  3= Quit   4= Sort(type)  5= Sort(date)  6= Sort(size)
7= Backward   8= Forward  9= FL /n 10=            11= XEDIT/LIST 12= Cursor

====>
```

```
                                                              X E D I T  1 File
```

Typing over the existing file line entry for ELITEST FOO A allows you to copy (or perform any other similar file management command) from the filelist interactive screen; see Example 6-10.

Example 6-10 Typing commands into the interactive file listing

```
ELI      FILELIST A0  V 169  Trunc=169 Size=6 Line=1 Col=1 Alt=29
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      MESSAGE  LOGFILE  A0 V         71          2          1  6/25/07 16:51:38
COPY / ELITEST BAR A    A1 F         80         21          1  6/25/07 11:38:19
      ELI      NETLOG   A0 V        104          2          1  6/21/07 12:12:53
      PROFILE  XEDIT    A1 V         72         29          1  6/21/07 12:11:29
      ELIFILE  TXT      A1 F         80         20          1  6/07/07 14:23:40
      PROFILE  EXEC     A1 V         31         14          1  6/01/07 11:26:07




1= Help       2= Refresh  3= Quit   4= Sort(type)  5= Sort(date)  6= Sort(size)
7= Backward   8= Forward  9= FL /n 10=            11= XEDIT/LIST 12= Cursor

====>
                                                              X E D I T  1 File
```

When you enter the command, an asterisk (*) indicating a change occurred will show up in the left margin. To see the updated file listing, execute a refresh operation; see Example 6-11.

*Example 6-11 The * indicates a refresh is required*

```
ELI      FILELIST A0  V 169  Trunc=169 Size=6 Line=1 Col=1 Alt=9
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      MESSAGE  LOGFILE  A0 V         71          2          1  6/25/07 16:51:38
*1    ELITEST  FOO      A1 F         80         21          1  6/25/07 11:38:19
      ELI      NETLOG   A0 V        104          2          1  6/21/07 12:12:53
      PROFILE  XEDIT    A1 V         72         29          1  6/21/07 12:11:29
      ELIFILE  TXT      A1 F         80         20          1  6/07/07 14:23:40
      PROFILE  EXEC     A1 V         31         14          1  6/01/07 11:26:07




1= Help       2= Refresh  3= Quit   4= Sort(type)  5= Sort(date)  6= Sort(size)
7= Backward   8= Forward  9= FL /n 10=            11= XEDIT/LIST 12= Cursor
```

```
====>
                                                              X E D I T  1 File
```

Press F2 to refresh the screen. Your output will now show the updated file listing with the new copy of ELITEST FOO A as ELITEST BAR A; see Example 6-12.

Example 6-12 The refreshed screen showing the newly copied file

```
ELI      FILELIST A0  V 169  Trunc=169 Size=7 Line=1 Col=1 Alt=47
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      ELITEST  BAR      A1 F         80         21          1  6/29/07 12:08:58
      MESSAGE  LOGFILE  A0 V         71          2          1  6/25/07 16:51:38
      ELITEST  FOO      A1 F         80         21          1  6/25/07 11:38:19
      ELI      NETLOG   A0 V        104          2          1  6/21/07 12:12:53
      PROFILE  XEDIT    A1 V         72         29          1  6/21/07 12:11:29
      ELIFILE  TXT      A1 F         80         20          1  6/07/07 14:23:40
      PROFILE  EXEC     A1 V         31         14          1  6/01/07 11:26:07




1= Help       2= Refresh  3= Quit   4= Sort(type)  5= Sort(date)  6= Sort(size)
7= Backward   8= Forward  9= FL /n 10=            11= XEDIT/LIST 12= Cursor

====>
                                                              X E D I T  1 File
```

6.7 Editing files with XEDIT

You have now learned how to examine your accessed disks, and work with files at the file system level. But what about editing at the file level? Creating and changing content is an important part of the CMS users skill set. This section will introduce you to the utility most commonly suited for editing files on a CMS system.

XEDIT is the name of the CMS editor, and is one of the most flexible and powerful utilities provided with CMS. The shortest abbreviated command used to edit a file with xedit is **X** followed by the file name, for example **X PROFILE EXEC**. The full command invocation of the same example would be **XEDIT PROFILE EXEC**. If you are not used to working with command line text editors, the interface to XEDIT may seem somewhat limited.

The basic XEDIT screen displays several areas: a command area, an edit area. a prefix area, and a line at the top of the screen that gives information about the file being edited.

The command region is a single line at the bottom of the terminal. Here you can issue commands such as **SAVE**, **FILE**, and **QUIT**. If you type anything into the edit region (most of the rest of the screen), you will be modifying the loaded file. This can frustrate new CMS users who are not used to command and data being manipulated the same way.

Important: Pressing Enter at any time will bring your cursor directly to the command line, but any text already on the command line will be executed!

Editing means changing, adding, or deleting data in a CMS file. You make these changes interactively; that is, you instruct the editor to make a change, the editor makes it, and then you request another change.

6.7.1 The XEDIT window layout

This sections describes the various areas of the XEDIT window, and explains what each are is used for.

File identification line

The file identification line (which is the first line on the screen) identifies the file you are editing. The display shows information about the file name, file type, file mode of the current file.

If you do not specify a file mode, the editor assigns a file mode of A1. The file mode identifies an accessed minidisk or SFS, for Shared File System, directory where the file resides.

The record format and record length (V 132) indicate the maximum length of line (the integer portion) and the V indicates lines may be variable length. Note that it is possible for a file line to be longer than a screen line.

The truncation column is the same as the record length (132). Because a file line can be only 132 characters long, any data you enter beyond 132 characters (in total) can be truncated. Additionally, you can see information about the current number of lines in the file. Finally, notice the alteration count shown. This value indicates the number of lines modified since the last AUTOSAVE. For more information about AUTOSAVE, refer to “AUTO SAVE” on page 194.

Message line

The editor communicates with you by displaying messages on the second and third lines (the message lines) of the screen. These messages tell you if you have made an error, or they provide information.

File area

The file area part of the screen is available to display the file. You can make changes to the file by moving the cursor to any line and typing over the characters, or by using special keys to insert or delete characters.

You can make as many changes as you want on the displayed lines before pressing Enter. When you press Enter, the changes are made to the copy of the file that is kept in virtual storage.

At the end of the editing session, a **FILE** subcommand permanently records those changes on the copy of the file that resides on disk or directory. Because a file can be too long to fit on one screen, various subcommands scroll the screen so you can move forward and backward in a file. Scrolling the screen is like turning the pages of a book.

Prefix area

The prefix area consists of the five left-most columns on the screen. Note that each line in the file has a prefix area.

The area may display five equal signs (=====), or sometimes line numbers, depending on the particular configuration of XEDIT being used. In some cases the prefix area has no special marker at all.

You can perform various editing tasks, such as deleting a line, by entering short commands called *prefix subcommands* in the prefix area of a line.

The current line

The current line is the file line in the middle of the screen (above the scale). It is highlighted, appearing brighter than the other file lines. The current line is an important concept, because most subcommands perform their functions starting with the current line.

The line that is current changes during an editing session as you scroll the screen, move up and down, and so forth. When the current line changes, the line pointer (not visible on the screen) has moved. Many XEDIT subcommands perform their functions starting with the current line and move the line pointer when they are finished.

Note: Sometimes XEDIT is configured to show a scale that appears under the current line to help you edit. It is like the margin scale on a graphical text editors.

Command line

The large arrow (`====>`) displayed at the bottom of the screen points to the command input area. One way you communicate with the editor is to enter XEDIT subcommands on this line. You can type subcommands in upper case or lower case or a combination of both, and many can be abbreviated. For example, `BOTTOM`, `Bottom`, and `b` are all valid formats for entering the BOTTOM subcommand.

After typing a subcommand on the command line, press Enter to execute the subcommand. To move the cursor from any place on the screen to the command line, simply press Enter or PF12)

Status area

The lower right corner, the status area, displays the current status of your editing session (for example, edit mode or input mode), and the number of files you are editing.

6.7.2 XEDIT and full screen CMS

If you invoke XEDIT from full screen CMS, the way you see messages that other users send you is not the same as when full screen CMS is off.

- When full screen CMS is off, the message appears on a cleared screen with a `HOLDING` status displayed at the bottom. You can press Clear to return to the XEDIT screen.
- If full screen CMS is on, then any message you receive appears in the message window, which automatically pops up on top of your XEDIT screen.

  To scroll forward in the message window, type `f` (forward) in one of the border corners (indicated by plus (+) signs) and press Enter.

  Continue to use the f border command until you have seen all the information in the message window. When there is no more information to display, the window is automatically removed from your screen.

6.7.3 Data manipulation with prefix subcommands

After you enter the XEDIT command, you are in typically in edit mode. You must be in edit mode to enter XEDIT subcommands. You can enter data into the file using input mode or power typing mode, which are discussed in the following sections.

If you are editing an existing file, you can simply place the cursor in the file area and type to replace the underlying text. As previously mentioned, XEDIT refers to

its “shortcuts” as prefix subcommands. Prefix subcommands are one- or two-character commands that perform basic editing tasks on a particular line.

You enter prefix subcommands by typing over any position of the five-character prefix area on one or more lines. When you press Enter, all of the prefix subcommands that have been typed on the screen are executed.

Setting the current line

Many subcommands begin their operations starting with the current line. For example, the **INPUT** subcommand makes room for you to enter data after the current line. You have already seen the **INPUT** subcommand that inserts lines after the Top of File line.

You can type the forward slash (/) prefix subcommand in the prefix area of any line on the screen. When you press Enter, that line becomes the current line. Then, if you enter an **INPUT** subcommand, the new lines entered in input mode are inserted between the current line and the line that followed it.

Adding lines

To add a line, type the single character `A` (or the character `I` for insert) in the prefix area to append blank lines. When you press Enter, a blank line is immediately inserted following the line containing the A. A number can precede or follow the A to indicate adding more than one line. For example, A5 adds five blank lines.

Here are valid ways to type the A prefix subcommand:

```
====A     Adds   one blank line after this line.
a====     Adds   one blank line after this line.
10a==     Adds   ten blank lines after this line.
===A5     Adds   five blank lines after this line.
```

You can then type information in the added lines. If no information is typed, the blank lines remain in the file throughout the editing session and after the file is written to disk or directory.

Deleting lines

To delete a line, enter the single character `D` in the prefix area of a line. A number can precede or follow the D to indicate deleting more than one line.

To delete a group of consecutive lines (that is, a block of lines) enter the double character `DD` in the prefix area of both the first and last lines to be deleted. This method makes it unnecessary for you to count the number of lines to be deleted; see Example 6-13.

Example 6-13 The consecutive line delete functionality

```
==dd=  This   is  the first line I want to remove.
=====  This   is  the second.
=====  This   is  the third.
=====  This   is  the fourth.
===dd  This   is  the fifth.
```

When you press Enter, the block of lines is deleted. The first and last lines of the block do not need to be on the same screen; you can scroll the screen before entering the second DD.

When you have typed one `DD` and pressed Enter, the status area of the screen displays `DD pending...`. You can use the PF7 or PF8 keys to scroll the screen until you find the last line of the block, and then type `DD` in its prefix area. When you press Enter, the entire block of lines is deleted.

Recovering deleted lines

If you delete one or more lines, you can recover them anytime during an editing session by using the **`RECOVER`** subcommand. The following subcommand returns lines deleted in an editing session:

```
RECover n
```

Here, n represents the number of lines you wish to recover. Recovered lines are inserted starting at the current line. The last lines deleted are the first lines recovered.

If the lines were deleted from different places in the file, you put them back where they belong by using the **`M`** prefix subcommand (refer to “Moving lines” on page 189 for more information about this subcommand). To recover all lines that you deleted during an editing session, enter:

```
====> recover
```

In the previous example of the `A` and `D` prefix subcommands, six lines were deleted. To recover only the last 2 lines, use the following command:

```
====> recover 2
```

Adding indented lines

To continuously add lines of indented text, type the characters `SI` in the prefix area. When you press Enter, a line is immediately added following the line that contains SI. The cursor is positioned at the same column where the text on the previous line begins, thus making it easier for you to enter indented text.

If you do not want to add more lines, press Enter one more time without typing anything on the new line. To add a blank line in a file while using SI, make at least one change (such as pressing the spacebar once) on the line that contains `. . . . .` in the prefix area. Note that simply using the cursor position keys to move the cursor over a line does *not* change the line.

You can leave the line you are adding and make corrections elsewhere in the file if you type something on the new line first. When you press Enter while the cursor is away from the new line, another new line is added following the last line that was added. SI is canceled only if you press Enter and have typed no text on the new line.

Duplicating lines

To duplicate a line, enter a double quote (") in the prefix area of a line. A number can precede or follow the double quote to duplicate the line more than one time; see Example 6-14.

Example 6-14 Inserting three duplicate lines

```
=3"== I want three more copies of this line.
===== They will appear before this line.
```

When you press Enter, the file displays output similar to Example 6-15.

Example 6-15 Output from the previous example

```
=====    I want three more copies of this line.
=====    I want three more copies of this line.
=====    I want three more copies of this line.
=====    I want three more copies of this line.
=====    They will appear before this line.
```

To duplicate a block of lines, either one time or a specified number of times, type two double quotes ("") in the *first* and *last* lines of the block. A number can precede or follow the first double quotes (for example, 5"") to duplicate the block more than one time.

When you type one double quote ("") and press Enter, the status area of the screen displays `"" pending...`. This allows you to scroll the screen before completing the block and pressing Enter.

Moving lines

To move one line, enter the single character `M` in the prefix area of the line to move. Indicate its destination by entering either the character `F` (following) or `P` (preceding) in the prefix area of another line. When you press Enter, the line

containing the M is removed from its original location and is inserted in one of the locations (either immediately following the line containing the F, or immediately preceding the line containing the P, as appropriate).

A number can precede or follow the M to indicate moving more than one line (for example, 5M or M5) in the prefix area. The line to move and the destination line can be on different screens.

After you enter M, F, or P, the status area of the screen displays a `Pending` status. This pending status allows you to scroll the screen before entering the other prefix subcommand.

To move a block of lines, enter the double character `MM` in the prefix area of both the *first* and *last* lines to be moved. The first and last lines to be moved, and the destination line, can all be on different screens. You can use PF keys to scroll the screen before pressing Enter.

Copying lines

The procedure for copying lines is the same as for moving lines, except that you enter a `C` or `CC` prefix subcommand instead of M or MM. The copy operation leaves the original lines in place but makes a copy at the destination line, which is indicated by F or P.

Canceling prefix subcommands

If you have entered one or more prefix subcommands that create a pending status, you can cancel all these prefix subcommands by entering the following subcommand on the command line:

```
====> reset
```

When you press Enter, all prefix subcommands disappear from the display and the prefix areas are restored with equal signs, numbers, or blank characters, depending on your particular XEDIT configuration.

If you have typed any prefix subcommands (even those that do not cause a pending status) but have not yet pressed Enter, you can press Clear to remove them. If you have only typed a few characters, it may also be sufficient to just type over them with blank characters.

6.7.4 Moving through a file

XEDIT lets you move backward, forward, to the top and bottom, and up and down in a file. You have already seen that the PF7 and PF8 keys are set to the BACKWARD and FORWARD subcommands, which scroll one full screen backward or forward.

You can also enter the BACKWARD and FORWARD subcommands in the command line. The format of these subcommands is:

```
BAckwardn
FOrwardn
```

Here n is the number of screen displays you want to scroll backward or forward. (This is like pressing PF7 or PF8 n times.) If you omit n, the editor scrolls one screen backward or forward.

If you enter a BACKWARD subcommand when the current line is the Top of File line, the editor wraps around the file, making the last line of the file the new current line. Similarly, if you enter a FORWARD subcommand when the current line is the End of File line, the editor makes the first line of the file the new current line.

Suppose the file is many screens long and the current screen display is somewhere in the middle of the file. To return to the beginning of the file, you could enter multiple BACKWARD subcommands, or you can enter the **TOP** subcommand.

The TOP subcommand makes the Top of File line the new current line. Enter the TOP subcommand as shown:

```
====> top
```

The BOTTOM subcommand makes the last line of the file the new current line. Enter the BOTTOM subcommand as shown:

```
====> bottom
```

These subcommands are useful when you want to insert new lines either at the beginning or end of a file.

Suppose that you want to move the file up or down a few lines instead of a whole screen. The **DOWN** subcommand advances the line pointer one or more lines toward the end of a file. The line pointed to becomes the new current line; for example:

```
====> down 5
```

This command makes the fifth line down from the current line the new current line. If you omit the number, then 1 is assumed. The **UP** subcommand moves the line pointer toward the beginning of the file. The line pointed to becomes the new current line; for example:

```
====> up 5
```

This command makes the fifth line up from the current line the new current line. If you omit the number, then 1 is assumed.

6.7.5 Searching within a file

Searching for a particular word or phrase is a very important part of text editing on any platform, and is especially important when using CMS to edit various configuration files. When using XEDIT, you can search by using the forward slash (/) command on the command line, as shown:

```
====> /target_search_string
```

After pressing return, XEDIT will move to the first line it has found that matches your search phrase. XEDIT searches by default, moving from the current line toward the end of the file.

This can be a very powerful way to move through files, but you can also search in reverse. To search in reverse, prefix the search command with the minus (-) symbol. The search will be performed from the current line moving toward the top of the file. Here is an example of a reverse search:

```
====> -/target (reverse search)
```

Lastly, in some cases highlighting all occurrences of a phrase within a file can be useful. To accomplish this task, XEDIT provides the **all** command:

```
====> all /target
```

From the XEDIT command line, the **all** command will show a listing of all lines that contain instances of the target search phrase. To return to normal editing mode, issue the all command with no search phrase as shown:

```
====> all
```

Note: When using the **all** command or the search / command, you can set your current line to be one of the lines of output. This will enable you to exit the search results and be immediately editing the file at your position of interest.

6.7.6 Setting tabs

You may want to place information in specific columns. The PF4 key functions like a tab key on a typewriter. Each time you press the PF4 key, the cursor is positioned under the next tab column, where you can enter data. The editor defines initial tab settings according to file type; you can display them with the following subcommand:

```
====> query tabs
```

You can change these settings one or more times during an editing session with the SET TABS subcommand; here is an example:

```
 ====> set tabs 10   20   30
```

The first time you press PF4, the cursor moves to column 10 on the screen. The second time, it moves to column 20, and so forth. You can use PF4 for tabbing in regular XEDIT input mode, but not all other advanced xedit modes. You can change the tab settings by entering another **SET TABS** subcommand. If you want to see the current tab settings before changing them, use the following subcommand:

```
====> modify tabs
```

This displays the current SET TABS subcommand in the command line; you can type over the numbers and press Enter to define new tabs.

6.7.7 Inserting from external files

The **GET** subcommand inserts all or part of another file into the file you are editing after the current line. (A file that you “get” is not destroyed; a copy of that file is inserted.)

Before you enter the **GET** subcommand, make the current line the line *preceding* where you want to insert data. That way, the file will be inserted immediately under the current line.

To insert another file at the end of your file, use the **BOTTOM** subcommand to make the last line current. To insert another file somewhere in the middle of your file, use the **UP** or **DOWN** subcommands to make the desired line current.

Inserting a whole file

To insert all of another file into the file you are editing, use the command:

```
====> get filename filetype
```

For example, to insert all of FILE2 SCRIPT A at the end of FILE1, then while you are editing FILE1, move the line pointer to the end of the file by issuing the command:

```
====> bottom
```

With the line pointer at the end of our current file, import the entire external file with the command:

```
====> get file2 script
```

When the entire second file has been inserted, the editor displays the message:

```
EOF reached
```

Inserting a portion of another file

Sometimes importing an entire file is unnecessary; instead, you only need a portion of an external file. The **GET** command takes additional optional arguments that will help you to insert a portion of another file.

The first additional parameter for this invocation specifies the line number of the first line to import. The second additional parameter indicates the number of suasive lines to insert. For example, the following **GET** subcommand inserts the first 10 lines of a second file:

```
====> get file2 data 1 10
```

Powertyping

If you need to type a significant amount of text continuously, without worrying about line numbers or word length, enter the command **POWERINP**. This places XEDIT in a mode called power input or powertyping mode.The shortest command abbreviation is **POW**.

When you are in powertyping mode, a `*** Power Typing***` banner is displayed across the top of the screen. Any time you want to leave powertyping mode, simply press the return key. Your text will be formatted to fit the correct line width in a normal XEDIT session.

Combining the **TOP** or **BOTTOM** commands with **POWERINP** provides an easy method for prepending or appending to a file.

AUTO SAVE

To minimize the risk of losing your data, XEDIT provides the **SET AUTOSAVE** subcommand. This subcommand causes your file to be automatically written to disk after you have typed in or changed a certain number of lines. Its format is:

```
SETAUtosave n
```

Here n is the number of typed-in or changed lines. For example, to write the file to disk or SFS directory every time you have changed 10 lines, enter:

```
====> set autosave 10
```

The number of alterations you have made to your file since the last AUTOSAVE is displayed in the alteration count (Alt=n) in the file identification line. When the alteration count is equal to the AUTOSAVE setting, and the file contains at least one record, the file is saved on disk or SFS directory and the alteration count is reset to zero (0).

You can enter the **SET AUTOSAVE** subcommand at any time during an editing session, but it is good practice to enter it right after you enter an XEDIT command to create a new file or to call an existing file from disk or SFS directory.

When a file is automatically saved, it is written into a new file whose file name is a number and whose file type is AUTOSAVE. If the system malfunctions during an editing session, you can recover all changes made up to the time of the last automatic save.

To do this, replace the original file with the AUTOSAVE file using the CMS **COPYFILE** command with the **REPLACE** option. If you enter a **SET AUTOSAVE** subcommand while you are creating a new file or revising an existing file, and then enter a **QUIT** subcommand, the new or revised file is not saved, but the AUTOSAVE file is available from disk.

6.7.8 Ending an editing session

You can end an editing session by using **FILE** or **QUIT**. When you use the **XEDIT** command to create a new file, the file is created in virtual storage. When you make changes to an existing file, those changes are made to a copy of the file that is brought into virtual storage (when the **XEDIT** command is entered). However, virtual storage is temporary. To write a new or modified file to disk, enter the following subcommand:

```
====> file
```

When the **FILE** subcommand is executed, the file is written to disk or directory and control is returned to CMS. You must use **FFILE** to file an empty file.

The **QUIT** subcommand ends an editing session and leaves the permanent copy of the file intact on the disk or directory. You can execute the **QUIT** subcommand either by pressing the PF3 key or by entering it on the command line, like this:

```
====> quit
```

Use the **QUIT** subcommand to quit XEDIT without saving changes to the file. This method of exiting XEDIT may be useful when you edit a file just to examine it but not change its contents, or if you have made an error when altering a file.

If the file is new and you have not input any data, the file is not written to disk. If a file is new or has been changed, the editor gives you a warning message to prevent your inadvertently using **QUIT** instead of **FILE**. The message is:

```
File has been changed; type QQUIT to quit anyway
```

If you really do not want to save the file, enter **QQUIT** (abbreviated as **QQ**):

```
====> qquit
```

6.7.9 Customizing xedit

You can improve the X interface by creating a PROFILE XEDIT file on your A disk. XEDIT starts with the basic requirements of any data editing program (that is, it should allow you to add, delete, or change records in a file interactively from your screen). It also provides a significant range of additional and more specialized functions. Some of the advanced features are:

- Sophisticated data location commands
- Program controllable changes
- Display layout “tailoring”
- Selective data display
- Multiple files handled simultaneously
- Multiple logical displays per physical display

Recognizing the variety of different users who will need an editor, and their different expectations and requirements, XEDIT allows each user to define how file data should be displayed and how commands should take effect. Discussing these advanced features, and many more, are beyond the scope of this book.

A sample PROFILE XEDIT is shown in Example 6-16.

Example 6-16 Sample PROFILE XEDIT A with basic XEDIT customization

```
* * * Top of File * * *
/****************************/
/*   Sample PROFILE XEDIt   */
/****************************/

/* Set up function keys to do something useful */
'SET PF01 HELP MENU'    /* XEDIT help                                  */
'SET PF02 SOS LINEADD'  /* Add a line at cursor position               */
'SET PF03 QUIT'         /* Quit XEDIt                                  */
'SET PF07 BACKWARD'     /* Scroll backward                             */
'SET PF08 FORWARD'      /* Scroll forward                              */
'SET PF09 ='            /* Re-execute last subcommand entered          */
'SET PF10 RIGHT 10'     /* Scroll document to right 10 columns         */
'SET PF11 LEFT 10'      /* Scroll document to left 10 columns          */
'SET PF12 ?'            /* Retrieve last command issued.               */

/* Set up colors */
'SET COLOR PREFIX BLUE'
'SET COLOR ARROW WHITE'
'SET COLOR FILEAREA GREEN'

/* Set up display of editing window */
'SET NUM ON'         /* Show line numbers                              */
'SET CMDLINE BOTTOM' /* Command line at bottom of screen               */
```

```
'SET SCALE OFF'      /* No "scale" bar across window                  */
'SET NULLS ON'       /* Allows you to insert text in middle of a line */
'SET CASE M I'       /* Allows uppercase and lowercase characters     */
'SET CURLINE ON 3'   /* Current line is always the 3rd visible line.  */
'SET FULLREAD ON'    /* move cursor to any spot and char stays put    */
'SET STAY ON'        /* stay at loc of last find vs bottom of file    */
* * * End of File * * *
```

Experiment with the various option in the configuration file. For example, saving the file to PROFILE XEDIT A in your CMS system will put the configuration overrides into production. It is recommended that you start by adding one line at a time to determine what you want. The configuration shown in Example 6-16 on page 196 may not meet your needs, but it is presented to illustrate the customization available to XEDIT users.

6.7.10 Getting help with XEDIT

If you forget how to use a subcommand, or want to view information about subcommands not covered in this subset, press PF1, which is set to the **HELP MENU** subcommand. PF1 lists all subcommands and macros available with the editor.

If PF1 does not display a help menu for XEDIT, you can manually enter the command **HELP XEDIT MENU**. Move the cursor to the desired subcommand and press Enter. The subcommand description appears on the screen, replacing the Full-Screen Text Processing HELP Menu.

To return to the previous screen, press PF3. To leave the HELP display and restore your file on the screen, press PF4.

6.8 The PROFILE EXEC

Many users may not want to learn about CMS facilities at all. All they may want to know is how to use the application of their choice (for example, AS, APL, or PROFS).

The functions of CP and CMS allow you to set up a virtual machine in which the only indication of the presence of CP or CMS comes at logon and logoff. At logon, the SYSPROF EXEC or PROFILE EXEC can be used to automatically start the chosen application. A more recent “application” automatically launched by CMS is the Linux operating systems that execute on VM.

6.8.1 PROFILE EXEC capabilities

A PROFILE EXEC is different from other execs. It has the special file name, PROFILE, and it is automatically processed whenever you enter `IPL CMS` (or if you have an automatic IPL mechanism in place).

Your PROFILE EXEC contains the CP and CMS commands that you enter at the start of every terminal session. It can be used to set up special characteristics for each CMS user beyond those defined in that users directory entry, such as:

- Describing your terminal and printer
- Making any non-standard minidisks or shared file systems in your virtual machine configuration known to CMS
- Changing the colors used to display data at a terminal (color terminals only)
- Running your synonym table
- Making frequently used execs storage-resident
- Setting up to 24 program function keys for commonly used commands
- Changing the Ready message sent by CMS
- Automatically checking your virtual card reader (your “in-tray”) for files and messages
- Taking you directly to an application instance

6.8.2 Creating a PROFILE EXEC

You can write your PROFILE EXEC for any of the exec interpreters, but REXX is the most common choice. A PROFILE EXEC written with REXX statements might appear as shown in Example 6-17 on page 198.

Example 6-17 PROFILE EXEC A file with convenience functionality added

```
/***************************/
/*   Sample PROFILE EXEC   */
/***************************/

/* Set up function keys to do useful things */
'SET PF1 IMMED HELP'
'SET PF11 RETRIEVE FORWARD'
'SET PF12 RETRIEVE BACKWARD'

/* Access TCPIP tools */
'LINK TCPMAINT 592 592 RR'
'ACCESS 592 F'

/* Ensure SET RUN is ON - Useful when we disconnect */
'SET RUN ON'

/* Highlight user input */
```

```
'TERM HILIGHT ON'

/* Display CP commands in RED so we know when cp is executing. */
'SCREEN CPOUT RED NONE'
* * * End of File ***
```

Using Example 6-17 along with the knowledge you gained in this chapter, you are able to customize your profile exec as you want. For new users, the commands **SET PF11 RETRIEVE FORWARD** and **SET PF12 RETREIVE BACKWARD** are particularly useful because they remove the need to retype commands you are experimenting with. Setting those options enables functionality like the bash shell on Linux systems and using PF11 and PF12 to scroll through previous command history.

If you make changes to your PROFILE EXEC during your terminal session, the changes will not be in effect until you run your profile again. You can enter the following command at any time to run your PROFILE EXEC:

```
profile
```

Note: Do not use the CP DEFINE STORAGE command in your PROFILE EXEC. It resets your virtual machine and you would have to IPL CMS again. Unfortunately the first thing your CMS guest does is load your PROFILE EXEC. This leads to a cyclic situation that leaves your system broken.

Often newer users make a few mistakes when customizing their PROFILE EXECs. To suppress the processing of your PROFILE EXEC for any reason, enter the **IPL** command, and then enter command **CMS ACCESS** with the **NOPROF** option specified.

This can help if you intentionally want to avoid the loading of your PROFILE EXEC in order to make alterations or corrections to it. From the login screen, enter your user ID and password on the appropriate line and then enter the command **ACCESS NOPROF** on the command line, as shown in Example 6-18.

Example 6-18 Logging on to a CMS guest without accessing the profile

```
   z/VM ONLINE


                          / VV          VVV MM       MM
                         /  VV        VVV  MMM     MMM
                ZZZZZZ  /   VV      VVV   MMMM   MMMM
                   ZZ  /    VV    VVV    MM MM MM MM
                  ZZ  /     VV  VVV     MM  MMM  MM
                 ZZ  /      VVVVV      MM   M   MM
```

```
                  ZZ     /       VVV       MM       MM
                 ZZZZZZ /        V        MM       MM

                   built on IBM Virtualization Technology
z/VM 5.3.0 IESP


Fill in your USERID and PASSWORD and press ENTER
(Your password will not appear when you type it)
USERID   ===>  ELI
PASSWORD ===>

COMMAND  ===>  ACCESS (NOPROF
                                                RUNNING   VMLINUX6
```

The system will respond with:

```
Ready;
```

This means that you have successfully loaded CMS and accessed file mode A without running your PROFILE EXEC. For more information about the CMS ACCESS command, refer to *z/VM: CMS Command and Utility Reference*.

6.8.3 Synonyms, abbreviations and parsing

At its simplest, enhancement of the system might take the form of providing synonyms for commonly used commands and procedures, or allowing abbreviations of their names.

All CMS commands may be abbreviated according to a supplied table. Each user may add further abbreviations or synonyms as necessary, for both CMS commands and EXEC procedure files. For example, you can use the command **del** (or **delete**) for erase if you invoke the synonym function with a synonym-table file containing this line:

```
ERASE DELETE 3
```

You can use the system-supplied synonym file on the 190 or 19E disk, or you can have your own on your 191 disk. Create a file MYSYN SYNONYM on your A disk. With the synonym file on disk, add the line SYNONYM MYSYN SYNONYM A to your PROFILE EXEC to complete the synonym process.

6.9 Distributing files

At this point, you have learned about the filesystem and ways of editing files. This section describes how you can distribute files among other users, including both those running on your z/VM system and those running remotely.

6.9.1 SEND and RECEIVE

Sending and receiving files uses the CP SPOOL subsystem to allow users to distribute files freely around the z/VM system. It uses the virtual reader, print and punch facilities described in 5.4.7, “Spool devices” on page 135 to distribute locally. It uses the remote spooling facilities of RSCS to distribute remotely.

Sending files can be done either from the command line or in a utility such as FILELIST. The command format is:

```
SENDFILE filename filetype filemode userid
```

Example 6-19 illustrates the command line execution; it shows using sendfile to transmit the file “testing file” on the A disk to another user eli that is on the same system. The file will be placed in the recipient’s reader.

Example 6-19 Sendfile sample

```
Ready; T=0.01/0.01 15:07:36
sf testing file A eli
File TESTING FILE A1 sent to ELI at VMLINUX6 on 06/04/07 15:07:44
Ready; T=0.01/0.01 15:07:44
```

Example 6-20 illustrates the **SENDFILE** command being executed from within the context of a filelist session. Because FILELIST is interactive, you can use CMS commands like **SENDFILE** without using the standard command line.

Example 6-20 Executing sendfile from a filelist session

```
CLIVE    FILELIST A0  V 169  Trunc=169 Size=6 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Fm Format Lrecl     Records     Blocks   Date     Time
      CLIVE    NETLOG   A0 V         103           3          1  6/04/07 15:07:44
SF / ELI TING  FILE     A1 F          80           2          1  6/04/07 15:04:13
      FRED     EXEC     A1 V          38           6          1  6/04/07 10:00:35
      PROFILE  XEDIT    A1 V          23           8          1  6/04/07  9:52:56
      LASTING  GLOBALV  A1 V          38           2          1  6/01/07 11:41:08
      PROFILE  EXEC     A1 V          31          14          1  6/01/07 11:26:07
```

Note: The command is only **SF / ELI** because the slash causes the file name to be implicitly entered.

If your system is running RSCS, it may be possible to send files to users on remote systems. To do this all you have to do is add AT NODEID to any of the user names that we have seen in the previous examples (for example, ELI AT VMLINUX7). Similarly, you can receive files using the **RECEIVE** command after you determine spool information by using the **Q RDR** command, as shown in Example .

Example 6-21 Examining the contents of a reader using the short form of the query rdr command

```
q rdr
ORIGINID FILE CLASS RECORDS  CPY HOLD FORM     DEST     KEEP MSG
DIRMAINT 0003 A PUN 00000027 001 NONE STANDARD OFF      OFF  OFF
CLIVE    0002 A PUN 00000141 001 NONE STANDARD OFF      OFF  OFF
DIRMAINT 0004 A PUN 00000010 001 NONE STANDARD OFF      OFF  OFF
Ready; T=0.01/0.01 15:17:14
receive 002 clives direct a
File CLIVES DIRECT A2 created from DIRMAINT NEWMAIL B2 received from CLIVE
at VMLINUX6
Ready; T=0.01/0.01 15:18:02
```

Note: You can rename files as you receive them. In Example 6-21, file DIRMAINT NEWMAIL was received as CLIVES DIRECT A.

You can also use a utility such as **RDRLIST**, as shown in Example 6-22.

Example 6-22 Using the RECEIVE command from the interactive reader listing

```
CLIVE    RDRLIST  A0  V 164  Trunc=164 Size=3 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Class User  at Node     Hold  Records  Date      Time
receive / = = a EWMAIL  PUN A CLIVE    VMLINUX6 NONE      141  6/01    11:41:08
      CLIVE    DIRECT   PUN A DIRMAINT VMLINUX6 NONE       27  6/01    11:41:09
      CLIVE    VMLINUX6 PUN A DIRMAINT VMLINUX6 NONE       10  6/01    12:01:02
```

Note: Specifying `= = a` after the slash / will retain the filename and filetype of the file that was sent.

Simply typing `receive /` is sufficient to automatically place the file, with file name intact, on your A disk. You could have chosen to save the file with another filename using the format receive / `newname newextension newmode`.

6.9.2 LINK and GRANT

5.4.4, “DASD (disk devices)” on page 126“Accessing someone else's DASD” on page 129 shows you how to use the **CP LINK** command to access the minidisks of other users. Similarly, you can allow other users to link to your minidisks in

either read or read/write mode, and they can then access your files and copy or update them as they want.

If you are using the Shared File System, you can use the **GRANT** command to allow other users to access specific files in your filespace.

6.9.3 FTP

There are several requirements that need to be in place before you can use FTP from your CMS user ID:

- You must have TCP/IP up, running, and connected to the network.
- You must have a FTPSERVE user ID configured and available. Typically this is configured by a systems administrator for you. You can check to see if the service is running by using the **Q NAMES** command previously discussed.
- You must have access to the FTP command, normally by issuing a LINK to TCPMAINT 592 minidisk and then using the **ACCESS** command previously discussed.

After these tasks are done, you can start to use FTP. The steps are fairly simple, and are the same as those performed on any other command line FTP program.

1. Issue the FTP command.
2. Enter your user ID and password.
3. List your files.
4. Select ASCII for text files and binary for other files; for example, a CMS COPYFILE (PACK file.
5. Issue **GET** or **PUT** as required, delimiting the filename, filetype and filemode with periods.
6. When finished, issue the **QUIT** command.

Example 6-23 demonstrates a complete ftp session on z/VM. (Notice the switch to ASCII mode for the file transfer.)

Example 6-23 Using CMS ftp program

```
Ready; T=0.01/0.01 16:30:19
ftp 9.12.4.200
VM TCP/IP FTP Level 530
Connecting to 9.12.4.200, port 21
220-FTPSERVE IBM VM Level 530 at VMLINUX8.ITSO.IBM.COM, 16:30:33 EDT MONDAY
200-06-04
220 Connection will close if idle for more than 5 minutes.
USER (identify yourself to the host):
```

```
KYLE
>>>USER kyle
331 Send password please.
Password:
>>>PASS ********
230 KYLE logged in; working directory = TCPMAINT 191
Command:
ls
>>>PORT 9,12,4,89,4,3
200 Port request OK.
>>>NLST
125 List started OK
FTP.TCPIP
PROFILE.EXEC
250 List completed successfully.
Command:
ascii
>>>TYPE a
200 Representation type is ASCII.
Command:
get ftp.tcpip ftp1.tcpip.a
>>>PORT 9,12,4,89,4,4
200 Port request OK.
>>>RETR ftp.tcpip
150 Sending file 'ftp.tcpip' FIXrecfm           80
250 Transfer completed successfully.
164 bytes transferred in 0.007 seconds. Transfer rate 23.43 Kbytes/sec.
Command:
quit
>>>QUIT
221 Quit command received. Goodbye.
```

6.10 Exercises

1. Which virtual address is typically accessed as A in a CMS environment?
2. What is the maximum number of minidisks that can be accessed concurrently by a single CMS user? What command would you use to show how many minidisks are accessed right now by your CMS guest?
3. Describe which command you would use to list files interactively, and which command you would use to list files statically. Explain a situation in which you would prefer each.
4. Provide a command that would list all the files with name PROFILE in them on your disk accessed with file mode A.

5. List two or more ways of creating a file using CMS tools. Provide a use case for each.
6. Review your understanding of XEDIT's file-closing mechanisms. Make a distinction between the **`qquit`**, **`quit`**, and **`file`** commands used in an XEDIT session, and their use cases.
7. Describe the role of your PROFILE EXEC A, and why it is important.
8. Examine your PROFILE EXEC A. How would you set up your CMS environment to automatically list files interacitvely upon each IPL?
9. As an optional experiment, customize your PROFILE EXEC as you deem appropriate. Be sure to make a backup before you begin editing.

7

The REXX programming language

This chapter introduces the REXX programming language (also known as the REstructured eXtended eXecutor language) on z/VM.

Objectives

After completing this chapter, you will be able to:

- Understand basic REXX/VM programming concepts
- Use data and control structures
- Explain data manipulation
- Describe functions and subroutines
- Use system interfaces
- Execute and debug REXX EXEC

© Copyright IBM Corp. 2007. All rights reserved.

7.1 What is REXX

The Restructured eXtended eXecutor language (REXX) is a versatile, easy-to-use structured programming language that is an integral part of z/VM. Although REXX is a general purpose language that resembles PL/I, there are major differences that make REXX more powerful.

REXX instructions are quite different from, and easier to use, than PL/I. For example, REXX programs are “interpreted” (that is, the language processor operates on the program directly as it runs). The advantage of using an interpreter lies in its superior security and error-handling abilities. If a program fails with a syntax error of some kind, for instance, the point of error is clearly indicated and can usually be understood and corrected quickly.

7.2 Features of REXX

REXX has many features, including the ability to include CP and CMS commands, a wide variety of functions including extensive arithmetic, character data parsing capabilities, and debugging features. Here, we examine REXX features in more detail.

- Ease of use

  REXX is a relatively easy language to read and write. Many of the instructions are English-like words or phrases that have similar meanings outside the computing world as well. Unlike many lower-level languages, REXX does not use abbreviations.

- Dynamic Variable allocation

  All variables in REXX are “typeless”. Instead, they contain variable-length strings with contents that define their data type. If you assign a variable a string that looks like a numeric value, REXX will allow you to perform calculations on it.

  If you assign a variable a character string, REXX will allow you to perform string manipulation operations on it (such as parsing, pattern-matching, or concatenation).

- Free format

  REXX has few formatting rules. For example, a single instruction can span many lines, or multiple instructions can be entered on a single line. Instructions to not need to begin in a particular column; you can skip spaces in a line or skip entire lines. You can type instructions in upper case, lower case, or mixed case. And there is no line numbering.

- Interpreted

  Each language statement in a REXX EXEC is processed directly by the interpreter. This means that you do not have to compile or link edit your program before you execute it. Simply run the EXEC.

  Languages that are not interpreted must be compiled into machine language (inseparable files) before they can be run.

- Built-in functions

  REXX supplies built-in functions that perform various processing, searching, and comparison operations for both text and numbers. Formatting, conversion and arithmetic functions are also available to the REXX programmer.

- Parsing capabilities

  The REXX language provides extensive character manipulation capabilities. The parsing capabilities of the language allow you to set up patterns or templates for separate characters, numbers or mixed input.

- Debugging

  A number of tools are available to assist you in debugging your REXX programs. When REXX encounters an error, a descriptive message is displayed at your terminal. Additionally, the trace instruction and the interactive debug facility are available.

Apart from all these features, the best aspect of REXX is the cross-system consistency that it provides. A program that is written in REXX will run in a variety of environments with little or no re-coding required. This means that you can write a REXX program to run under CMS and port it immediately to run in a Multiple Virtual Storage (MVS) operating system.

7.3 REXX and VM

The most vital role REXX plays is as a language that can be used to build complex applications for z/VM. By using REXX, you can reduce lon, complex or repetitious tasks to a single command or program that can be run from CMS. REXX is a built-in feature of z/VM, so there is no installation process or separate environment.

z/VM provides you with a broad range of programming interfaces. These interfaces are available using the following facilities:

- EXEC statements (REXX instructions, for example)
- CMS assembler macros and functions

- Callable service library (CSL) routines
- OS/MVS and DOS/VSE simulation interfaces
- CMS and CP commands
- Data record formats, such as accounting records, intended to be processed by application programs
- CP system services (such as MSG)

In general, z/VM programming interfaces are designed to be used exclusively from programs (often using binary or other machine-level formats as parameter values), and they are usually supported in a compatible manner from release to release.

7.4 REXX overview

REXX consists of a small core of approximately 20 instructions. This core is then surrounded by a rich function set of about 70 built-in functions; see Figure 7-1.

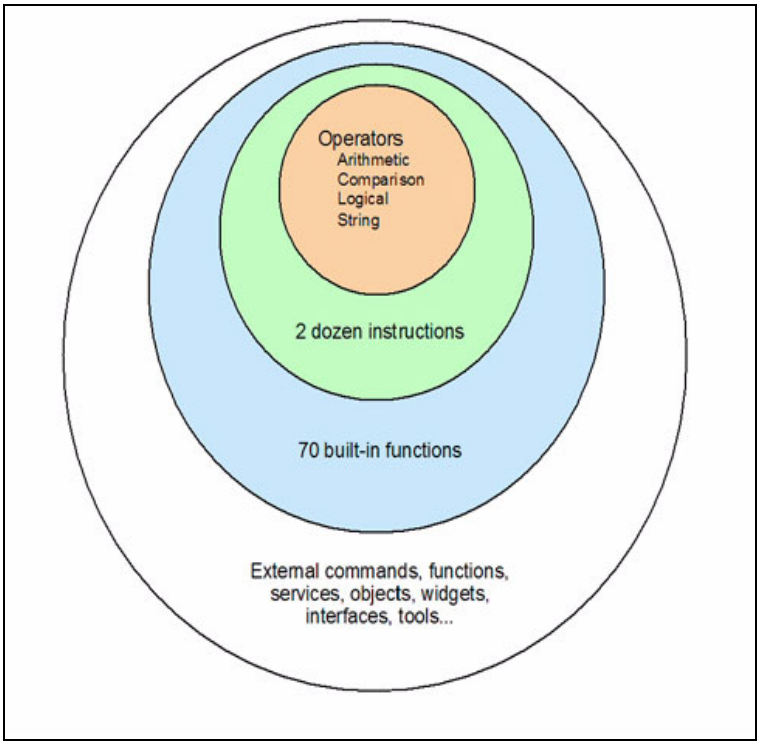


Figure 7-1 Elements of REXX

This structure makes it easy to learn the basics while adding to your knowledge of the functions over time. Beyond the built-in functions, REXX scripts can easily access dozens of free external function libraries, tools, and interfaces.

7.4.1 REXX components

The REXX language is made up of the following components:

- Instructions
- REXX built-in functions
- System-supplied functions
- Program stack functions

We explain these components in more detail in the following section.

Instructions

REXX has the following types of instructions: keywords, assignments, labels, and commands.

Keywords

Keyword instructions consist of one or more clauses beginning with a REXX keyword; see Figure 7-1 on page 210. They are used to control external interfaces (printers, disks), logic flow and to handle some data manipulation tasks.

Example 7-1 Keyword instructions

```
DO, SAY, PARSE
```

Assignments

Assignment instructions are used to give variables new values. They consist of a single clause in the form VARIABLE = EXPERESSION; see Figure 7-2 on page 219.

Example 7-2 Assignment instructions

```
SUM = ((NO1 * NO2) + NO3)
```

Labels

Label instructions consists of a single symbol followed directly by a colon (:), as shown in Figure 7-3 on page 235.

A label is used to identify the target of **CALL** or **SIGNAL** instructions to any function and internal function calls.

Example 7-3 Label instructions

```
GO:
SUBRTN1:
```

Note: **CALL** and **SIGNAL** are discussed in more detail in 7.11, “Functions and subroutines” on page 235.

Commands

Commands are not processed by the REXX language processor, but by some external environment. To issue a command, you must enclose it in double quotes (“ ”) or in single quotes (‘ ’); see Figure 7-4 on page 244.

Example 7-4 Command instructions

```
“QUERY NAMES”
“LISTFILE * * A”
```

REXX built-in functions

Functions that are supplied by the REXX language provide a wide variety of components and powerful processing options; see Example 7-5.

Example 7-5 Built-in functions

```
DATATYPE - returns the type of data in a specified string.
MAX - returns the largest number from a specified list
REVERSE - returns a string swapped end-for-end.
```

System-supplied functions

These functions interact with the system to accomplish specific, system-related tasks; see Example 7-6.

Example 7-6 System-supplied function

```
STORAGE - used to display or alter the main storage (RAM) of your virtual
machine.
CMSFLAG - gives the setting of certain indicators, for example to find in which
environment the virtual machine is.
```

Program stack functions

These functions allow you to manipulate the program stack structure. A *program stack* is used to store data for I/O and other types of processing; see Example 7-7 on page 213.

Example 7-7 Program stack functions

```
PUSH - Adds elements to a program stack.
PULL - Removes elements from a program stack.
```

7.4.2 General structures and syntax

In this section we explain the general structures and syntax that make up REXX.

Comments

A *comment* is a sequence of characters (on one or more lines) delimited at the beginning and end by a forward slash and asterisk (/*); see Example 7-8.

Within these delimiters, any characters are allowed. Comments have no effect on the program, but they do act as separators. (Two tokens with only a comment in between are not treated as a single token.)

Example 7-8 Null instructions

```
/* this is an example of a valid REXX comment */
```

Note: REXX EXECs must start with a comment statement.

Naming variables

We can create any variable name by using any combination of the following characters:

- A . . . Z upper case alphabetic
- a . . . z lower case alphabetic
- 0 . . . 9 numbers
- # $ _ ... special characters

Example 7-9 lists variable names that are valid.

Example 7-9 Valid variable names

```
ANITA
#82C
A9
```

There are restrictions regarding naming the variables:

- The first character cannot be a period (.) and cannot be 0 through 9.
- The variable name cannot exceed 250 characters.
- Do not use RC, SIGL or RESULT (these are reserved).

Assigning values

Like most high level languages, REXX variables can be initialized and allocated as needed. If we are editing a REXX exec for an existing program, and we need to introduce a new variable called “alpha”, then the variable will automatically be given the initial value ALPHA (upper case) and will be kept until we decide to give it another value.

A variable value can be any of the following:

- A constant (for example, A = 4)
- A string (for example, A = ‘hello’)
- The value from another variable (for example, A = B)
- An expression (for example, A = B+C)

Expressions

An expression can consist of numbers, variables, strings and one or more operators. It returns a single unique value. To create or evaluate an expression, we use four types of operators:

- Arithmetic
- Concatenation
- Comparison
- Logical

Next, we explain these operators in more detail.

Arithmetic

Using the operators listed in Table 7-1, you can write any arithmetic operators.

Table 7-1 Arithmetic operators

Operator	Description
+	ADD
-	SUBTRACT
*	MULTIPLY
/	DIVIDE
%	DIVIDE and return a whole number without a remainder
//	DIVIDE and return the remainder only
**	EXPONENTIATE raise to the whole number power

Operator	Description
-NUMBER	NEGATE the number

Example 7-10 illustrates sample arithmetic operators.

Example 7-10 Arithmetic operators

```
6 + 2 : result would be ‘8’
6 ** 2 : result would be ‘36’
6 / 2 : result would be ‘3’
7 % 2 : result would be ‘3’
7 // 2 : result would be ‘1’
```

Order of evaluation

The following is the order followed during evaluation of an expression.

1. Expressions within brackets or Parentheses are evaluated first.
2. Then, from left to right, the order is:
   a. ** exponentiation
   b. * / % // multiplication and division
   c. + - addition and subtraction

Tip: An easy way to remember the order of evaluation is by using the acronym BEDMAS: Brackets Exponents Division Multiplication Addition Subtraction.

Concatenation

Concatenation operators combine two terms into one. The terms can be strings, variables, expressions or constants. Concatenation gives you significant control over the format of your output. Table 7-2 lists and describes concatenation operators.

Table 7-2 Concatenation operators

Operator	Description
blank	This operator concatenates terms, and places one blank in between. Terms that are separated by more than one blank default to one blank.
||	This operator concatenates terms, and places *no* blanks in between.

Example 7-11 on page 216 illustrates concatenation operators.

Example 7-11 Concatenation operators

```
/* REXX Concatenation Example */
climb='steel'
tools= 'hammer'
column= ' '
cost = 100
SAY clim||tools column '$'cost
```

When executed, the program would display: `steelhammer $100`.

Comparison operators

By using comparison operators, you can compare two expressions and get an response expressed as either "true or false" or "1 or 0". If the code is 'variable-name = expression', REXX automatically knows that you are doing a ASSIGNMENT, but in all other cases the operators would be used for comparing to expressions. Table 7-3 lists and describes common comparison operators.

Table 7-3 Common comparison operators

Operator	Description
= =	Strictly equal
=	equal
\ = =	not strictly equal
\ =	not equal
>	greater than
<	less than
><	greater than or less than

Example 7-12 illustrates a sample comparison operator.

Example 7-12 Comparison operator sample

```
if 5 >= 4 then say ...
if sales = "SALES" then say ..
if 2E2 \== 200 then say ...
if 5 >< 5 then say ...
```

Logical operators

Logical (or Boolean) operators combine two comparisons and return a true (1) or false (0) value when processed. Using the common logical operators listed in

Table 7-4, you can built new operators by combining one or more logical operators.

Table 7-4 Common logical operators

Operator	Description	Result
&	AND	Returns 1 if both comparisons are true.
|	inclusive OR	Returns 1 if at least one comparison is true
&&	exclusive OR	Returns 1 if only one comparison (but not both) is true.
prefix \	logical NOT	Returns the opposite response.

Example 7-13 illustrates a logical operators sample.

Example 7-13 Logical operators

```
(9 > 8) & (A = A) Result would be 1 (because both are true)
(5 > 4) | (4 = 3) Result would be 1 (because atleast one is true)
\ (5>4) Result would be 0 (since opposite of true)
```

The evaluation of operators overall proceeds as listed in Table 7-5.

Table 7-5 Order of evaluation for operators

ARITHMETIC	PREFIX EXPONENTIAL MULTIPLY and DIVIDE ADD and SUBTRACT
CONCATENATION	CONCATENATION
COMPARISON	COMPARISON
LOGICAL	LOGICAL AND INCLUSIVE and EXCLUSIVE OR

7.5 Creating an EXEC

At this point, we have explained the general structures and syntax of REXX and as a result, you are now able to write your first REXX EXEC.

You can start with a short REXX program that accepts an input and then displays it on your console or monitor. To create the program, follow these steps:

1. Log on to z/VM and IPL CMS. Then type the command:

   ```
   xedit welcome exec
   ```

2. Type in the program exactly as it is shown in Example 7-14.
3. Save the file by using either `save` or file `XEDIT` command.

Example 7-14 WELCOME EXEC

```
1. /* WELCOME EXEC  - First REXX Program */
2. say 'Type in your name:'
3. pull name
4. if name <> "" then say 'Welcome ' name 'to the REXX World'
5. else say 'No Inputs '
```

Example 7-14 has five statements, called *clauses*. These clauses have the following meanings:

1. The first clause is a comment explaining what the program is about. All REXX programs must begin with a begin-comment delimiter (/*). In this example, the comment is `/* WELCOME EXEC - First REXX Program */`
2. The second clause is the keyword instruction `say` which displays text on the screen.

   After the `say` command, the literal string is placed between the quotation marks, so that it is displayed just as it is. In this example, the literal string is `'Type in your name:'`.
3. The third clause is the keyword instruction `pull` which reads and stores the response entered by the program's user.

   In the third clause we also define a variable; in this example the variable is `name`. This is basically a name given to the place in storage where the user's response is stored.
4. The fourth clause begins with the `if` instruction; it tests a given condition.
   - If the condition is true, the `then` part of the `if` instruction is executed.
5. If the condition is false, the `else` part of the `if` instruction is executed.

7.6 Executing an EXEC

Because the program you created is of type EXEC, you need to type in only the file name from the `Ready` prompt. From the filelist view, you can also invoke or execute the REXX EXEC simply by issuing `EXEC` from the command column of the filelist view; see Example 7-15.

Example 7-15 Executing REXX Program from the filelist view

```
Cmd   Filename Filetype Fm Format Lrecl     Records     Blocks   Date     Time
EXEC WELCOME   EXEC     A1 V         58           5          1  6/01/07 15:16:59
     LASTING   GLOBALV   A1 V         38           3          1  6/01/07 14:02:42
```

7.7 Stopping an EXEC

If you need to stop the REXX program at any time from processing further, issue either the CMS command HI (HALT INTERPRETATION) or the CMS command HX (HALT EXECUTION) from the `Ready` prompt. This will cause REXX to stop running the program and return to the CMS prompt.

7.8 Terminal I/O and control structures

Programs generally require some form of input. This data may come from a number of different sources; see Figure 7-2.

When you want the user to supply some input to the program, it is necessary to prompt the user. The REXX instruction **`SAY`** is used to display information to the terminal screen.

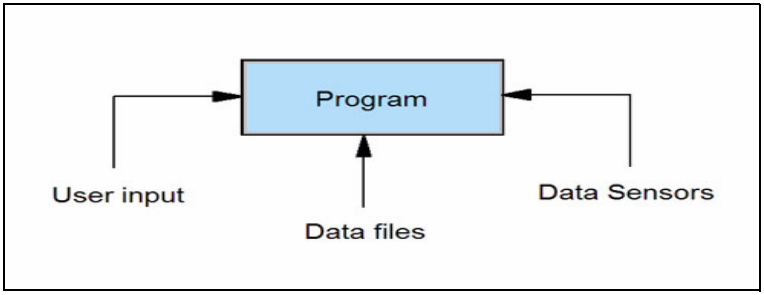


Figure 7-2 Input/output

There are two REXX instructions that you can use to accept input from the user: PULL, and ARG. We explain PULL in this section. We explain ARG in 7.8.1, “The ARG statement” on page 221.

Note these points regarding the PULL instruction:

- PULL is used to accept input from users.
- PULL automatically translates all input to upper case.

Example 7-16 illustrates the use of the PULL instruction.

Example 7-16 WELCOME EXEC1

```
/* Accepting the user name using PULL - REXX exec */
SAY ‘Enter your name below’
PULL name
SAY ‘Welcome ‘ name
```

The PULL instruction can also accept more than one piece of data. In Example 7-17, notice that we altered the WELCOME EXEC to accept two inputs. Here the PULL instruction splits the input into separate arguments at the spaces that are in the input.

Example 7-17 WELCOME EXEC2

```
/* Accepting the user name using PULL - REXX exec */
SAY ‘Enter your first and last name below’
PULL fname lname
SAY ‘Welcome ‘fname lname
```

The DESKTOP EXEC

Using the REXX knowledge you have gained at this point, you can now start to build an exec that will provide a few basic functions. In this case, we use the SAY and PULL instructions to create a menu of options for the user, as shown in Example 7-18. We name this EXEC the DESKTOP EXEC.

Example 7-18 DESKTOP EXEC

```
/* DESKTOP - REXX EXEC  */

   SAY '        DESKTOP'
   SAY ' '
   SAY ' 1. DISPLAY CURRENT DATE & TIME'
   SAY ' 2. ADD TWO NUMBERS (CALCULATOR)'
   SAY ' 3. QUIT'
   SAY ' '
   PULL option
```

In Example 7-18 on page 220, the SAY instructions are used to create the menu and list the options. The PULL instruction is then used to accept the user's choice into the variable `option`.

This is the beginning of our DESKTOP EXEC example. We will keep building on this as you learn other concepts in the following sections.

7.8.1 The ARG statement

The ARG (for argument) instruction is used to pass data to an exec when it is invoked. (Note that "statements" are also known as "instructions".) Here, we alter the WELCOME EXEC to use the ARG statement (rather than PULL) to accept the user's name; see Example 7-19.

Example 7-19 WELCOME EXEC3

```
/* Accepting the user name using ARG- REXX exec */
ARG fname lname
SAY ‘Welcome ‘fname lname
```

To invoke this exec, we need to supply the `fname` and `lname`, as shown in Example 7-20.

Example 7-20 invoking WELCOME EXEC3

```
WELCOME SAM NICK
```

Upon execution you will notice that the string has been converted to upper case, exactly as it was when we used the PULL instruction.

Note these points regarding the ARG instruction:

- The tokens in the template become variable names.
- The last token is assigned all remaining arguments.
- The full stop or period (.) is a placeholder.
- ARG is a shorthand form of the instruction PARSE UPPER ARG.
- Use the form PARSE ARG to preserve lower-case values from the terminal.
- When there are no arguments, tokens in the template are assigned the null string (nothing). (Keep in mind that this is different from the value of an unassigned token.)

7.8.2 Parsing data

To change the format of the user's input, the REXX language has a powerful parsing capability. *Parsing* involves taking a string of characters and breaking it up into specified parts.

You may recall, from the discussion of PULL and ARG instructions, that data is translated into upper case. The PULL statement in this exec is really a short form of the instruction shown in Example 7-21.

Example 7-21 PARSE syntax

```
PARSE UPPER PULL fname lname
```

PULL is actually a part of the PARSE syntax. The PARSE instruction is used to assign data from various sources to one or more variables. For example, strings can be passed to the PARSE statement, as shown in Example 7-22.

Example 7-22 Passing string to PARSE instruction

```
/* VALUE .. WITH Clause : PARSE Exampler */
PARSE VALUE 'WELCOME TO REXX' WITH w1 w2 w3
SAY w1 w2 w3
```

The template

The template is the part of the PARSE statement that makes it so powerful. A template is simply a pattern that tells REXX how to break up the specified strings. The simplest form of the template is a list of variable names. The string being parsed is split into words; each word or group of words is assigned to the specified variables. In Example 7-22, the parse statement would break up the string and store into the variables w1, w2, and w3.

Note these points regarding templates:

- All variables listed in the template will be assigned a new value.
- The case of the words will be identical to the strings.
- If there is no value in the string that can be assigned to variable in the template, then the variable is set to a NULL string.
- In most cases, the leading or trailing blanks would be truncated.

Patterns

The real power of the template is that you can specify parsing patterns in it. As explained here, there are two types of patterns: patterns that search for a matching string, and patterns that specify position.

Matching strings - literal patterns

In Example 7-23, the PARSE statement would put the entire string in m1, and m1 through m6 would be set to NULL strings. This is because the PARSE statement looks for blanks as delimiters between the words.

Example 7-23 Literal pattern 1

```
string = 'jan,feb,mar,apr,may,june'
PARSE VAR string m1 m2 m3 m4 m5 m6
```

In Example 7-24 we specify a comma (,) as a delimiter, instead of a blank. This PARSE statement will scan the string to find a sequence that matches the value of the literal, in this case a comma. When specifying literal patterns, they must be enclosed in single (' ') quotes.

Example 7-24 Literal pattern 2

```
string = 'jan,feb,mar,apr,may,june'
PARSE VAR string m1 ',' m2 ',' m3 ',' m4 ',' m5 ',' m6
```

Note: The literal pattern, in our case the comma (,) is removed from the data being parsed.

Matching string - variable pattern

Variable patterns are similar to literal patterns. The only difference is that the variable pattern is enclosed in parentheses () and not in single quotes; see Example 7-25.

Example 7-25 Syntax for variable pattern

```
date = '06/30/2007'
PARSE VAR DATE mm (/) dd (/) yy
```

Positional - absolute patterns

Positional patterns can be used to cause parsing to occur on the basis of position within the string, rather than on its contents. You can specify the absolute position in the string with an unsigned integer; see Example 7-26.

Example 7-26 Absolute patterns

```
Using DATE = '06/30/90'
PARSE VAR DATE 7 yy
```

The string will be scanned and, beginning in the position specified (7, in this case), it will assign the remainder of the string to the variable yy.

Positional - relative patterns

Relative positioning allows you to specify signed integers that indicate movement relative to the first character position at which the previous match occurred; see Example 7-27.

Example 7-27 Relative patterns

```
PARSE VALUE '1234567890' WITH 4 n1 +2 n2 -4 n3 8
```

In this case the parsing beings at position 4, traverses for a length of 2 characters, and assigns them to string n1. Then it adds 2 to the first character position of the last match (that is, 4 + 2 = 6).

Next it begins at position 6 for a length of - 4. That is illogical, however, so the remaining characters from position 6 to the end of the string are assigned to n2.

Then it subtracts 4 from the first character position of the last match (6 - 4 = 2). And beginning at position 2 through position 8 (but not including the contents of position 8), it assigns the contents of position 2 through 7 to the string n3.

7.9 Conditional branching structures

Assignments inside an EXEC can be very useful if you want to provide a default value. For example, what if the user of the program development EXEC fails to provide arguments at all? The program must be able to detect this and take appropriate action, including setting up default values.

The way to do this in REXX is to test for conditions. REXX has two conditional branching structures:

- IF-THEN-ELSE
- SELECT-WHEN-OTHERWISE-END

In the following sections, we explain these branching structures in more detail.

7.9.1 The IF instruction

The IF-THEN-ELSE structure is used to conditionally execute one of two options. Generally, at least one instruction should follow THEN and ELSE clauses; see Example 7-28 on page 225.

Example 7-28 Syntax IF-THEN-ELSE

```
IF expression THEN instruction
              ELSE instruction
```

There may be times when either the THEN or ELSE clause requires that no instruction is executed. For this reason, it is good programming practice to include a NOP instruction next to the clause, as shown in Example 7-29.

Example 7-29 Syntax IF-THEN-ELSE with NOP

```
IF expression THEN instruction
              ELSE NOP
```

If you have more than one instruction for a condition, you must begin the set of instructions with a DO and end them with an END; see Example 7-30.

Example 7-30 RETIRE EXEC - IF-THEN-ELSE

```
/* RETIRE EXEC - IF-THEN-ELSE */
SAY 'Enter your AGE'
PULL age
IF age >= 58 THEN
   DO
      SAY 'Are you retired (y/n)?'
      PULL retired
   END
ELSE
  DO
     SAY 'At what age do you wish to retire?'
     PULL age_retire
END
```

Now, add a few more items to Example 7-30. If users are retired, you want to know at what age they retired. The program already asks if they are retired, so you need to add another IF-THEN-ELSE structure. This is called a “nested” IF-THEN-ELSE structure.

You can have a structure that is nested as many times as you need; see Example 7-31 on page 226.

Note: Keep in mind that, when nesting IF-THEN-ELSE structures, each DO must be matched with a corresponding END.

Example 7-31 RETIRE EXEC Nested IF-THEN-ELSE

```
/* RETIRE EXEC - IF-THEN-ELSE */
SAY 'Enter your AGE'
PULL age
IF age >= 58 THEN
    DO
      SAY 'Are you retired(y/n)?'
      PULL retired_yn
      IF retired_yn = 'Y' THEN
         DO
            SAY 'At what age did you retire?'
            PULL retire_age
         END
      ELSE NOP
    END
 ELSE
    DO
      SAY 'At what age do you wish to retire?'
      PULL age_retire
    END
```

Rules for the IF instruction:

- The expression must result in a 1 or a zero (0); that is the condition is true or false.
- You must have a THEN, followed by whatever you want to be done only if the expression is true (1).
- You may have an ELSE, followed by whatever you want to be done only if the expression is false (0).
- You can have multiple conditions (IF A and B THEN or IF A or B or (C and D...) and nested conditions.

7.9.2 The SELECT instruction

As mentioned, REXX has two conditional branching structures. The IF-THEN-ELSE structure allows you to conditionally execute one of two choices, or even more, if you nest them.

The SELECT-WHEN-OTHERWISE-END instructions, as shown in Example 7-32 on page 227, gives you the option of multiple choices.

Example 7-32 SELECT syntax

```
SELECT
    WHEN expression THEN instruction
    WHEN expression THEN instruction
    WHEN expression THEN instruction
      .
      .
      .
    OTHERWISE
       instruction(s)
 END
```

This is how the process works. The language processor scans the WHEN clauses until it finds a true expression. It this point, it processes the instructions. After it has found a true expression it ignores all other WHEN clauses, even if the expressions are true. If none of the expressions are true, it processes the instructions in the OTHERWISE clause.

As shown in Example 7-32, you get the month from the user by using the PULL instructions and then use conditional structures to find out the number of days in that particular month.

The SELECT instruction is used in this case, because you will need multiple choices for months.

Note: As with the IF-THEN-ELSE structure, if there is more than one instruction it must be enclosed in a DO-END structure.

However, this does *not* apply to the instructions in the OTHERWISE clause.

Example 7-33 LEAP EXEC - SELECT statement

```
/* LEAP EXEC - SELECT Instruction */
 SAY ‘Enter the Month : ‘
 PULL Month
 SELECT
     WHEN month = 2 THEN
        DO
           SAY "Is it a leap year (Y/N)?’
           PULL leapyear
           IF leapyear = 'Y' THEN days = 29
                             ELSE days = 28
        END
     WHEN month = 4  THEN days = 30
     WHEN month = 6  THEN days = 30
     WHEN month = 9  THEN days = 30
```

```
    WHEN month = 11 THEN days = 30
    OTHERWISE days = 31
END
SAY 'Number of days is : ' days
```

7.10 Looping structures

REXX has a number of looping structures that help you in repeating instructions. We group these looping structures into three types, as explained here.

- Iterative
- Infinite
- Conditional

7.10.1 Iterative looping

Iterative loops are used to repeat a set of instructions a *specified* number of times. Use iterative loops when you know how many times you want the loop to execute.

In Example 7-34, both of these loops would execute exactly the same way, and `count` would equal 100 at the conclusion.

Example 7-34 DO clause

```
Do Statement mode 1
DO number = 1 to 10
  count = count + 10
END

Do Statement mode 2
DO number = 1 to 10 by 1
  count = count + 10
END
```

Now, as shown in Example 7-35 on page 229, suppose you want to display the odd numbers between 1 and 10. You could add a conditional branch that checked for “oddness” and displayed the number only if the condition was met.

However, if you use the BY clause to specify the increment, you will get the desired result. If you run this loop, the odd numbers are displayed and the loop quits after `number` exceeds 10.

Example 7-35 ODD EXEC - DO loop

```
/* ODD EXEC - odd number generation using do loop */
DO number = 1 to 10 by 2
  SAY number
END
SAY 'Dropped out when'
SAY 'loop reached ' number
```

7.10.2 Infinite looping

Example 7-36 shows a crude example of infinite looping. Whenever the counter is incremented with the BY clause, inside the loop you decrement it.

Example 7-36 INFINITE EXEC

```
/* INFINITE EXEC - Crude way
DO counter = 10 to 29 by 5
    SAY counter
    counter = counter -5
 END
 SAY ‘counter = ‘ counter
```

Because this is an infinite loop, it will go on indefinitely. This type of loop will occur when the loop cannot obtain the last number. In this case the loop will never exceed 29. There may be occasions when you want an infinite loop, for example:

- In an exec that reads records from a data set until it reaches the end of file.
- In an exec that interacts with the user until a specific symbol is entered.

For these occasions, REXX has a loop structure to create an infinite loop; see Example 7-37.

Example 7-37 MARKLIST EXEC - EXIT instruction

```
/* MARKLIST EXEC - Do Infinite     */
DO FOREVER
    SAY 'Enter a name of student: '
    PULL name
    SAY 'Enter Science Marks : '
    PULL sci_mark
    SAY 'Enter Maths Marks : '
    PULL mat_mark
    IF name = '' THEN
       EXIT
    ELSE NOP
END
```

```
SAY ‘Still in the EXEC’
```

In Example 7-37 on page 229, the DO FOREVER loop will only cease when the user enters a blank for the name. There are three instructions for interrupting a loop: the instructions **EXIT**, **LEAVE**, and **ITERATE**.

In this instance, we used the **EXIT** instruction. When the user enters a blank name, the conditional branch will be true and the **EXIT** statement will be executed.

Loop interrupts

In this section, we explain the loop interrupt instructions EXIT, LEAVE, and ITERATE.

EXIT instruction

The EXIT instruction causes an exec to unconditionally end and return to where the exec was invoked. For example, if you invoked an exec from an ISPF panel, then when the EXIT instruction was executed, you would be returned to that panel.

Returning to Example 7-37 on page 229, you can see that the EXIT instruction halts the execution of the entire exec and returns control to the location from which the exec was invoked (in this case, CMS). Also, the last SAY instruction would not be executed.

LEAVE instruction

The LEAVE instruction causes an immediate exit from a loop, and control goes to the instruction following the END keyword. As shown in Example 7-38, we replaced EXIT with the LEAVE instruction.

Example 7-38 MARKLIST EXEC - LEAVE instruction

```
/* MARKLIST EXEC - Do Infinite     */
DO FOREVER
    SAY 'Enter a name of student: '
    PULL name
    SAY 'Enter Science Marks : '
    PULL sci_mark
    SAY 'Enter Maths Marks : '
    PULL mat_mark
    IF name = '' THEN
       LEAVE
    ELSE NOP
END
SAY ‘Still in the EXEC’
```

In this case, the control would return to the END statement and the last SAY instruction would be executed.

The LEAVE instruction can be used to exit and terminate any type of loop; that is, iterative, infinite, and conditional.

ITERATE instruction

The ITERATE instruction stops execution within the loop and returns control to the DO instruction at the top of the loop.

In Example 7-39, the REXX exec would go into an infinite loop. After the user wants to end the execution by keying a blank name, we issue the ITERATE instruction to give control to the DO FOREVER statement.

Example 7-39 MARKLIST EXEC - ITERATE instruction

```
/* MARKLIST EXEC - Do Infinite     */
DO FOREVER
    SAY 'Enter a name of student: '
    PULL name
    SAY 'Enter Science Marks : '
    PULL sci_mark
    SAY 'Enter Maths Marks : '
    PULL mat_mark
    IF name = '' THEN
       ITERATE
    ELSE NOP
END
SAY ‘Still in the EXEC’
```

Note: The ITERATE instruction only makes sense in an iterative or conditional loop. You will never get out of an infinite loop using this interrupt instruction.

7.10.3 Conditional looping

The CONDITIONAL loops are controlled by one or more expressions, and they loop until the expression is either true or false, depending on the type of loop structure.

There are two types of conditional loops: the DO WHILE structure, and the DO UNTIL structure. The main difference lies in *when* the looping condition is tested, as explained here:

- With DO WHILE, the condition is tested at the top of the loop.
- With DO UNTIL, the condition is tested at the bottom of the loop.

Example 7-40 illustrates DO WHILE syntax.

DO WHILE

Example 7-40 DO WHILE syntax

```
DO WHILE expression
   instruction(s)
END
```

The DO WHILE structure checks the expression at the *top* of the loop; see Example 7-41. This expression must initially be true in order for the instructions to be executed.

Example 7-41 PASSWORD EXEC - DO WHILE looping

```
/* PASSWORD EXEC - DO WHILE LOOP */
time=0
 psswrd='abra'
 SAY "ENTER PASSWORD: "
 PULL ans
 DO WHILE (ans <> psswrd) & (time <= 3)
    SAY "INVALID PASSWORD - TRY AGAIN"
    SAY "ENTER PASSWORD: "
    PULL ans
    time = time + 1
 END
```

This is what happens when the EXEC is executed:

- Assume that the first time a user is prompted for password, the user provides a wrong password. Both conditions are true, so the expression is evaluated to true and the loop instructions are executed.
- Inside the loop, the user provides the correct password. The expression is evaluated to false this time through and the DO-WHILE loop is now terminated.
- As soon as the expression becomes false, the loop structure is ended and the instruction that immediately follows the END clause is evaluated.

DO UNTIL

The DO UNTIL structure will execute until the loop expression is true (that is, as long as it is false).

Example 7-42 on page 233 illustrates the DO UNTIL syntax.

Example 7-42 DO UNTIL syntax

```
DO UNTIL expression
    instruction(s)
 END
```

With DO UNTIL, as mentioned, the expression is checked at the *bottom* of the loop. As a result, the instructions will always be executed at least once; see Example 7-43.

Example 7-43 PASSWORD EXEC - DO UNTIL

```
/* PASSWORD EXEC - DO UNTIL LOOP */
time   = 0
 psswrd = 'ABRA'
 DO UNTIL (ans = psswrd) ¦ (time = 3)
    SAY "ENTER PASSWORD: "
    PULL ans
    IF ans <> psswrd THEN DO
       SAY "INVALID PASSWORD - TRY AGAIN"
       time = time + 1;  END
    ELSE NOP
 END
```

This is what happens when the EXEC is executed:

- No evaluation takes place on the first time past this statement. Evaluation will occur after the END statement.
- Assume that the first time a user is prompted for password, the user provides a wrong password. The ELSE clause is skipped because the expression was evaluated to true and the THEN instructions were executed.
- The expression is false, so the instructions are executed again.
- Inside the loop for the second time, the user provides the correct password, so the expression is evaluated to true. The No OPeration clause, or NOP, was previously discussed, so if the password is equalled, then nothing gets done.
- The expression is now resolved to be true, and the DO UNTIL loop is terminated.
- As soon as the expression becomes true, the loop is ended and the instruction that immediately follows the END clause is executed.

DESKTOP EXEC - adding functionality

Next, we explain how to incorporate the functionality that you have learned into the DESKTOP EXEC; see Example 7-44. At this point, you will concentrate on writing the code that will display the current date and time.

Note: Because the topic of built-in functions has not been discussed yet, simply assume at this point that the current date and time are stored in curdate and curtime in the format mm/dd/yy.

Example 7-44 DESKTOP EXEC

```
/* DESKTOP - REXX EXEC  */
DO FOREVER

   SAY '        DESKTOP'
   SAY ' '
   SAY ' 1. DISPLAY CURRENT DATE & TIME'
   SAY ' 2. ADD TWO NUMBERS'
   SAY ' 3. QUIT'
   SAY ' '
   PULL option
   SELECT
       WHEN option = '1' THEN
       DO
           PARSE VAR curdate mm '/' dd '/' yy
         SELECT
            WHEN mm = '01' THEN month = 'January'
            WHEN mm = '02' THEN month = 'February'
            WHEN mm = '02' THEN month = 'March'
            ::::::::::::
               WHEN mm = '12' THEN month = 'December'
            SAY 'DATE: ' month dd', 19'yy
            SAY 'TIME: ' curtime
       END
   WHEN option = '2' THEN
         ::::::::::::::::
       WHEN option = '3' THEN  LEAVE
          OTHERWISE
             SAY 'INVALID OPTION - REENTER'

END
EXIT      /* return to calling environment */
```

In this example, the SELECT clause is used to branch the menu according to the user's menu choice. Using the SELECT claus is most appropriate when there are more than two alternatives.

The PARSE instruction to parse the contents of curdate into variables `mm`, `dd`, and `yy`. With the month data in variable `mm`, the SELECT-WHEN-OTHERWISE construct is used to determine the name of the month.

Now that we know the name of the month, we can display the date in the form month dd, 19yy. The first function of the DESKTOP EXEC is almost complete. As we proceed through the next topics, we add to the exec and improve its efficiency.

7.11 Functions and subroutines

A function or a subroutine is a series of instructions that a REXX exec calls on to perform a specific task. A function or a subroutine may receive data, process data, and return a value, as illustrated in Figure 7-3.

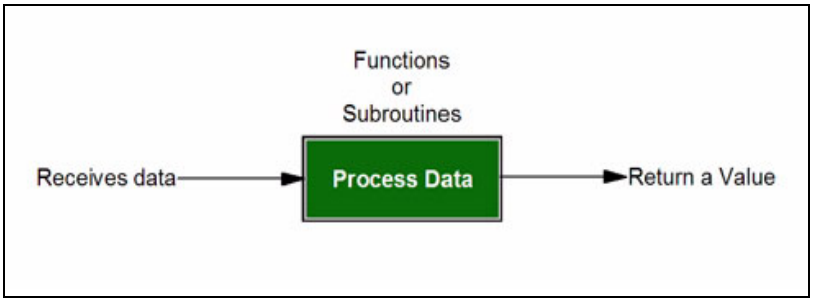


Figure 7-3 Functions and subroutines

The major difference between a function and a subroutine is that a function *must* return a value to the calling routine. In contrast, a subroutine *might* return a value to the calling routine.

7.11.1 Control instructions

As your EXECs are refined, they become longer. And as they become longer, it becomes more difficult to keep the essential logic in view as you read through them. Fortunately, REXX contains additional control instructions that allow flexibility in the structure or layout of an EXEC.

For example, it is often possible to give EXECs a modular structure; some parts will be repeated sequences of instructions, or can be treated as logically independent of the main body. You can regard such sections as subroutines.

CALL instruction

The CALL instruction allows you to pass control to a different point for a subfunction of your EXEC; see Example 7-45. When that function or routine has finished, control will return to the next instruction after the CALL.

Example 7-45 CALL instruction syntax

```
CALL    name [expression[,expression[, ...]]] ;
```

The data passed with the CALL instruction, after the routine name, is accessible to the called routine as a new set of arguments. Each expression separated by a comma (“,”) becomes a separate argument string for the subroutine.

RETURN instruction

A CALL is ended by the first RETURN instruction found, or by the end of a routine; illustrates the syntax of the RETURN instruction.

Example 7-46 RETURN instruction syntax

```
RETURN        [expression]  ;
```

RESULT special variable

The optional “expression” on the RETURN instruction sets a special variable called RESULT. REXX built-in functions always give a RESULT, if called. The RESULT variable will be uninitialized after a CALL if it is not implicitly set by the RETURN instruction, as shown in Example 7-47.

Example 7-47 Checking for the return value using RESULT variable

```
/* USER WRITTEN EXEC */
Call  FRED
Say  result
/* Do something */
FRED:  /*subroutine*/
Say ‘ I am returning the result’
Return /*..And give nothing back*/
```

The instruction `Say result` will produce the response `RESULT`.

Note: The RESULT special variable is different from the way that ARG or PULL instructions treat variables named templates.

CALLing routines

CALLs can be made to internal functions or subroutines, and to external functions and subroutines, as explained here.

Internal functions or subroutines

CALLs made to internal functions or subroutines means that control will pass to a statement that has a label matching the name specified on the CALL, as illustrated in Example 7-48.

Example 7-48 Using CALL to call internal routines

```
/* CALL Instruction - Internal Call */
Call FRED 'ceramic camels'
    ...
FRED: /* Special subroutine */
Arg values /* What data have we been passed?*/
    say ‘Value passed is :’ value
Return
```

External functions or subroutines

REXX may also look for external routines with names that satisfy the CALL name, as illustrated in Example 7-49. This means that you could use the CALL statement to invoke another EXEC or MODULE (CMS program).

Example 7-49 Calling another program using CALL instruction

```
call JANE 'some parameter(s)'
             |
             --------------- |JANE EXEC B|
                             /* another exec */
                             Arg something /*anything passed?*/
                                           ...
                             Return 'interesting value'
              -------------------------
              |
say result /* what have we got back?*/
```

PROCEDURE instruction

The PROCEDURE instruction can also be used with internal subroutines. PROCEDURE will protect all existing variables by making them unknown to following instructions. Using PROCEDURE, when a RETURN instruction is executed, the original variable environment is restored and any variables used by the routine are dropped.

If you want some variables to be available to the following subroutine, use the EXPOSE option. Any variables not mentioned in the EXPOSE will still be protected; refer to Example 7-50 for illustrated usage.

Example 7-50 Protecting using PROCEDURE

```
/*This is the main program*/
j=1; x.1='a'
call protect
say j k m                                  /* Would display "1 7 M" */
exit
protect:procedure expose j k x.j
          say j k x.j                      /* Would display "1 K a" */
          k=7; m=3                         /* Note "M" is not exposed*/
          return
```

The PROCEDURE instruction can be very useful when you are building a program with subroutines that are standard packages, because you can “bolt” these subroutines on to the main program without worrying about the variable names that have already been used.

7.11.2 Functions

A function is a sequence of instructions that can receive data, process data, and return a value. An exec activates a function by naming the function (either built-in or user-written), immediately followed by parentheses () with no blanks in between.

A function can be built-in or user-written, as explained here.

Built-in functions

The REXX interpreter has a number of functions built into it that are always available. REXX provides more than 60 standard functions that you may require on a regular basis. These 60+ functions fall into the categories listed and described in Table 7-6.

Table 7-6 Built-in functions and categories

Functions	Description
Comparison functions	Compare numbers and strings, or numbers or strings, and return a value
Conversion functions	Convert one type of data representation into another type

Functions	Description
Formatting	Manipulate the characters and spacing
String manipulating	Analyze a string (or a character representing a string) and return a value.

Example 7-51 illustrates a common built-in function.

Example 7-51 Finding the largest of the three numbers

```
/* Arithmetic Built-in - REXX Example*/
 a=10; b=5; c=20; d=6
 bignum = max(a,b,c,d,4)
 SAY 'the largest number is ' bignum
 EXIT
```

As illustrated in Example 7-51, by using the MAX built-in function, you need only three lines of instruction to return the largest number from your input list of numbers.

Example 7-52 REXX built-in functions

```
If substr(name,1,1)='?' then do
 . . .
name=strip(name,'B')    /* ensure clean name */
 . . .
lfile=userid() 'PACKAGES A'
 . . .
call diag 8,'CLOSE' pun 'NAME' subword(lfile,1,2)
 . . .
log=' 'date('O') time() ' 'pak'    ' who
 . . .
indent=right(' ',depth*2-1)
 . . .
Do i=1 to LENGTH(Alphabet) by 1
```

These examples are taken from real EXECs, and they illustrate some of the available functions and the different places where you can use them. There are approximately sixty different functions supplied with CMS. Table 7-7 on page 240 lists and describes some of these functions.

Table 7-7 Common built-in REXX functions

Function	Description
Abbrev(information,info,length)	Validate a string as an abbreviation of another string; for example: *pull ans.* */* 'Yes' entered? */* If abbrev('YES',ans,1) then...
Copies(string,n)	Returns n copies of string; for example: say copies('ABBA',2) ---> ABBAABBA
Datatype(string,type)	Returns simple description of type of data in string, or allows you to check for specific types (Alphanumeric, Bits, Lowercase, Number, Symbol, Uppercase, Whole number, heXadecimal); for example: say datatype('ABC') ---> 'CHAR' /* is input hex */ If datatype(input,'X') then
Date(form)	Returns current date in various forms; for example: say date() ----> '13 Oct 1990' say date('M') ----> 'October' say date('W') ----> 'Friday'
Left(string,length)	Returns the leftmost characters of a string; for example: say left('ABCD',2) ----> 'AB'
Length(string)	Returns the length of a character string; for example: say length('ABCD.....XYZ') ----> '26' /* process each character */ do i = 1 to length(input)
Queued	Gives the number of lines in the data stack
Random(min,max,seed)	Gives a (pseudo-) random number between 0 and 999

Right(string,length)	Returns the rightmost characters of a string; for example: say right('ABCD',2) ----> 'CD'
Strip(string,option,char)	Removes leading and trailing characters, or removes leading or trailing characters; for example: say strip(' ABCD ') ----> 'ABCD' say strip('001230','L',0) ----> '1230'
Substr(string,start,length)	Returns part of a larger string; for example: say substr('ABCDEFGH',3,2)----> 'CD'
Time(option)	Returns current time in various forms (Civil, Elapsed,Hours,Long, Minutes, Normal, Reset, Seconds); for example: say time() ---> '16:45:01' say time('H') ---> '16'
Translate()	Translates a string to upper case or according to specified table; for example: say translate('abcd') --->'ABCD' say translate('abba','B','b') --->'aBBa'
Userid()	Returns the VM user ID
Word()	Returns the nth word of a string; for example: say word('just an example',2) ---> an
Words()	Returns the number of words in a string; for example: say words(date()) ---> 3

For further details about REXX built-in functions, refer to *z/VM REXX/VM Reference*, SC24-5770; *REXX/VM Reference*, SC24-6113; and *CMS and REXX/VM Messages and Codes*, GC24-6118.

User-written functions

Repetitive tasks that you must do within your EXEC can be called by using CALL. User-written functions are functions written by you, the user. Sometimes it can be useful to think of functions as extensions to the REXX language. Example 7-53 on page 242 shows the syntax for a user-written function.

Functions can contain up to ten arguments, or no arguments at all. When a function ends, it will use the RETURN instruction to send a value back to the calling routine.

Example 7-53 User-written function syntax

```
function(argument1,argument2...)
```

Example 7-54 shows a REXX EXEC that uses a user-written function to calculate the factorial of two numbers.

Example 7-54 FACT EXEC -

```
/* Factorial program */

  do n=1 to 5
    say 'The factorial of' n 'is:' factorial( n )
  end
  return

  factorial : procedure
    n = arg(1)
    if n = 1 then
      return 1
    return n * factorial( n - 1 )
```

DESKTOP EXEC - using built-in functions

Returning to our DESKTOP EXEC example, in order to complete the DATE and TIME functions, you need to use built-in functions to retrieve the system date and time; see Example 7-55.

Example 7-55 Using built-in functions to retrieve system date and time

```
/* DESKTOP - REXX EXEC  */
CALL INIT

DO FOREVER

   SAY '        DESKTOP'
   SAY ' '
   SAY ' 1. DISPLAY CURRENT DATE & TIME'
   SAY ' 2. Add Two Numbers'
   SAY ' 3. QUIT'
   SAY ' '
   PULL option
   SELECT
     WHEN option = '1' THEN  DO
       curdate =  DATE(u)
       curtime =  TIME()
       PARSE VAR curdate mm '/' dd '/' yy
       SELECT
            WHEN mm = '01' THEN month = 'January'
```

```
            WHEN mm = '02' THEN month = 'February'
            WHEN mm = '02' THEN month = 'March'
            ::::::::::::
            :::::::
            WHEN mm = '12' THEN month = 'December'
            SAY 'DATE: ' month dd', 19'yy
            SAY 'TIME: ' curtime
      END
    WHEN option = '2' THEN DO
          SAY ‘ENTER THE FIRST NUMBER : ‘
          PULL FNUMBER
          SAY ‘ENTER THE SECOND NUMBER : ‘
          PULL SNUMBER
          res = CALC()
          SAY ‘THE ADDITION of ‘ FNUMBER ‘and’  SNUMBER ‘ : ‘ res
          ::::::::::::::::
    END
    WHEN option = '3' THEN  LEAVE
           OTHERWISE
              SAY 'INVALID OPTION - REENTER'
END
EXIT      /* return to calling environment */
/* CALCULATE Procedure */
 CALC:
 a = 2
 b = 5
 sum = a+b
 RETURN sum
```

7.11.3 Program stack

A program stack is a useful hybrid that combines conventional stack and queue structures with a number of other unique characteristics.

- A program stack can contain a virtually limitless number of data items of unlimited size. This makes it a very flexible data structure with many applications.
- The data items may be commands to be executed when the exec ends.
- The program stack can be used to pass information between REXX execs and other types of programs in VM and non-VM environments.

The flexibility of the program stack gives it many potential uses, including the following tasks:

- Storing a large number of data items for a single exec's use

- Passing a large number of arguments or an unknown number of arguments between a routine and the main exec
- Passing responses to an interactive command that can run after an exec ends

The program stack is used to pass data to CMS commands, or to obtain data from them. In computing terminology, a *stack* is a list of items that you can work with from only one end, which is the top. You can PUSH an item onto the stack, or you can PULL an item off the stack, as illustrated in Figure 7-4.

The item you PULL off will always be the last item that you (or someone else) pushed on. This method is known as LIFO, which is an acronym for “last in, first out.”

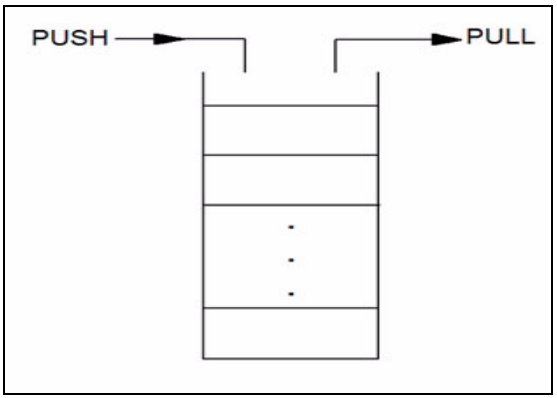


Figure 7-4 A stack using PUSH and PULL

Queues

A *queue*, in contrast, is a list of items which you can work with from *both* ends. You can decide to QUEUE (or add) items only at the back, and you can decide to PULL items only off the front, as illustrated in Figure 7-5 on page 245. This method is known as FIFO, which is an acronym for “first in, first out.”

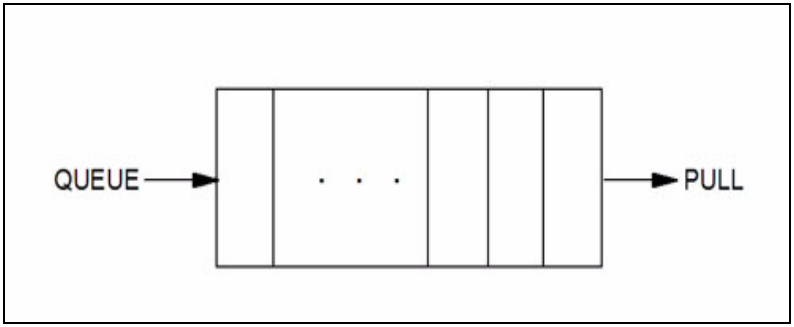


Figure 7-5 A stack using QUEUE and PULL

Buffers

A *buffer* is a general term for a part of the computer's storage that is used for input or output. You can build buffers as extensions to the program stack. Usually there is only one buffer in the program stack. You can create new buffers by using the **MAKEBUF** command.

Using a program stack

Follow these steps to use a program stack:

1. Begin the stack-processing portion of your program with the CMS command **MAKEBUF**. This will set up your own newly created buffer in the program stack.
2. Find out how many entries are already on the stack by using the **QUEUED( )** function; for example, using theirs as a variable name: "**theirs= queued()**"
3. Use the **QUEUE** instruction to put data onto the program stack. The **PUSH** instruction can also be used to put data on the top of the program stack.
4. Use the **PULL** instruction to take data off the stack.
5. Be sure that you have removed all your data from the program stack before you return to CMS.You can use the CMS command DROPBUF to do this.

Note: It is important to avoid removing items that your program did not place on the program stack. This would only occur after you have placed a number of items on the stack. Remove the items one at a time, first checking that what you are about to remove is yours.

This is has to be done programatically by keeping track of what you place into the stack.

Adding elements to the program stack

There are three instructions that are used to manipulate program stacks: two instructions that add items, and one instruction that removes items.

We begin by looking at the instructions (PUSH and QUEUE) that add elements to the program stack.

PUSH Puts one item of data on the top of a program stack.
QUEUE Puts one item of data on the bottom of a program stack.

Example 7-56 illustrates a program stack exec.

Example 7-56 Program stack exec

```
/*REXX - Program Stack (PUSH/QUEUE)
 item1 = 'THIS'
 item2 = 'IS'
 item3 = 'THE'
 item4 = 'END'
 PUSH  item1
 QUEUE item2
 PUSH  item3
 QUEUE item4
```

After the execution of the REXX EXEC, the program stack would look as shown here.

THE
THIS
IS
END

Note: To avoid confusion, most programmers will consistently use either PUSH or QUEUE.

Removing elements from a program stack

Next, we look at the PULL instruction, which removes elements from a program stack.

PULL Removes one item at a time from the top of a program stack.

Example 7-57 on page 247 illustrates the use of the PULL command.

Example 7-57 PULL command

```
/* REXX - PULL Command
item1 = 'THIS'
PUSH  item1
PULL Stack1
SAY STACK1
```

Sample program stack

The simple program illustrated in Example 7-58 issues a warning message when your primary minidisk, file mode A, is more than 80% full.

Example 7-58 MDCHECK EXEC

```
/* MDCHECK EXEC - check for minidisk */
/* Gives a warning when the user's primary minidisk (file mode a) is */
/* more that eighty percent full. */

"MAKEBUF"
"QUERY DISK A (STACK"
if rc = 0 then do
   pull                 /* Discard the header */
   parse pull "=" percentage .
   if percentage > 80
      then say "Warning : Your disk is " percentage "% full"
end
else say "MDCHECK EXEC - unexpected error " rc
"DROPBUF"
```

The REXX program shown in the example (MDCHECK EXEC) gives a warning when the user's primary minidisk (file mode A) is more than 80% full. This EXEC utilizes the MAKEBUF buffers and the program stack.

After the buffer is created and the stack used to hold the output, the program uses the IF statement to check whether the return code (RC) is equal to zero (0). It then discards the header line of the QUERY DISK output and pulls in the data line of the QUERY DISK output.

It finds the portion that contains the percentage and uses it to check whether the percentage is greater than 80. If it is, a warning is displayed. If the percentage is not greater than 80, then no action is taken and the buffer is released using the DROPBUF command.

7.11.4 Compound variables and stems

A compound variable consists of at least one period (.) with at least one character on either side of it.

Example 7-59 Compound variable

```
ANYTHING.6
PHONE.NAME
NAME.FIRST.LAST
ARRAY.I.J
```

These are the compound variable stems. A *stem* consists of the first variable name and the period (.) as shown in Example 7-59.

As discussed, REXX variables are dynamically allocated and initialized at the time of first use. The same is true of compound variables. For example, if we make the assignments shown in Example 7-60, they display the value of the compound variable.

Example 7-60 Compound variables declaration

```
/* COMP EXEC - Compound variable exec */
country = 'USA'
state = 'California'
SAY home.country.state.city
```

The example illustrates the first use of this compound variable. The following rules apply to initializing its value:

- The stem (home.) is always the upper case equivalent of itself.
- The remaining variables are substituted with their values.

Notice that the city is being used for the first time and so is initialized to CITY. So you can see, before the compound variable is used, the value of any simple variables after the stem (for example, country, state, city) are substituted into the variable, thus generating a new derived name. This derived name is then used just like a simple variable; see Example 7-61.

Example 7-61 Derived name

```
HOME.USA.California.CITY=address
```

Arrays using compound variables

You can use compound variables to create and access arrays, as previously mentioned. Note that there are a few differences between typical array

processing and the use of compound variables. For example, array subscripts do not need to be numeric, which can be of interest to the creative programmer.

Example 7-62 illustrates how to implement an array.

Example 7-62 Implementing an array

```
/* ARRAY Manipulation - Compound variables */
DO i = 1 TO 7
   SAY 'Enter Name: '
   PULL name.i
END
```

Assume the first DO loop portion of a REXX exec was executed; the resulting array of names would be created as shown here.

name.1 = cheryl
name.2 = clive
name.3 = fred
name.4 = bobby
name.5 = lopez
name.6 = jason
name.7 = frank

In the second DO loop portion of the exec, REXX departs from traditional array processing. Because you do not require numeric subscripts, you can use an array element to subscript another array.

Stems

When working with compound variables, it may be useful to be able to initialize an entire collection of variables, perhaps an array, to the same value. You can accomplish this easily by a stem.

You can refer to all the variables in an array by using its stem. It is often convenient to set all variables in an array to zero (0) by using their stem. Example 7-63 illustrates an array (named TOTAL) being initialized through the use of stems.

Example 7-63 Array initialization using stems

```
/* TOTAL EXEC - STEM */
/* This initializes all array elements to 0. */
```

```
total. = 0
 do forever   /* do loop that runs forever */
    /* Asking the user a question */
    say "Enter an amount and a name:"
    /* To retrieve data from the user (2 items ret) */
    pull amount name
    /* Make sure amount is # */
    if datatype(amount) = 'CHAR' then leave
         total.name = total.name + amount
         /* In the total array there is a name segment that */
         /* stores the total amount entered for a specified name */
 end
```

The value that has been assigned to the whole collection of variables can always be obtained by using the stem. As illustrated in Example 7-64, an array is defined in the form `arrayname.datatyp`.

Example 7-64 Array variables

```
total.dan = 200
total.kelly = 345
total.stephanie = 800000
```

7.11.5 Host environment commands

In this section we discuss the various operating (or host) environments that are available to REXX, and explain how you can perform a task in one environment and then easily move to another environment within the same exec.

REXX host environment

The host environments available in REXX VM are CMS, Command, and XEDIT (and optionally, ISPEXECs) as explained here:

CMS This is an environment in which VM/CMS commands and VM REXX commands execute.

Command The CMS Command environment presents commands directly to CMS, and makes execs faster and more predictable.

XEDIT This is the VM editor, which accepts subcommands from REXX execs.

ISPEXEC This is the environment in which ISPF commands execute. (ISPF is an optional user interface.)

The CMS, Command and XEDIT environments are readily available on an VM installation. The default command environment is CMS for REXX execs. Otherwise, the default environment is the one where the program was called from.

Changing the host environment

A single exec may issue commands in more than one environment. To do this, you must be able to change host environments at any time. Use the **ADDRESS** instruction, followed by the name of the new environment, as shown in Example 7-65.

Example 7-65 Changing the environment to XEDIT

```
/*ADDRESS EXEC */
ADDRESS xedit
"SET P7 UP 10" /* xedit command follows the ADDRESS Statement */
"SET P8 NEXT 10"
EXIT
```

The EXEC would continue to be in XEDIT, until you issue a new host environment command.

Determining the host environment

It may be important to have your exec be aware of what the current host environment is before attempting a specific task. To determine the active environment, use the **ADDRESS** built-in function as shown in Example 7-66.

Example 7-66 Finding the host environment

```
/* HOST ENVIRONMENT EXEC */
  x = ADDRESS()
  SAY x
  EXIT
```

By including the **SAY** instruction, as shown in the example, you can display the current environment on your screen.

Issuing HOST commands

VM/CMS commands are enclosed in single (' ') or double (" ") quotation marks; see Example 7-67.

Example 7-67 Issuing host commands

```
 /* Creating a Temporary Disk using REXX */
arg size addrs
```

```
if size='' | size='?' then
      do
        say 'this EXEC gets a temporary disk'
        say 'It requires two parameters          '
        say 'The size of the disk required    '
        say 'and an address to use for the disk '
        say 'the format of the call is:-'
        say 'TDSK size address    '
        exit 99
    end
if addrs='' then do
        say 'Please enter an address to use '
        pull addrs .
  end
'DEFINE T3380 AS 'addrs 'CYL' size

if rc¬=0 then
      do
        say 'define failed please check the address'
        exit 99
      end
'FORMAT' ADDRS 'K'
```

Other host environment commands are also enclosed in single or double quotation marks. Example 7-68 illustrates how to run the ADD EXEC in A disk from within another REXX exec.

Example 7-68 Examples for calling a EXEC

```
‘EXEC Add Exec A’
'EXEC Add 54 49 78'
```

The host command and arguments are all enclosed in single or double quotes. However, there is an exception: when you pass variables to an exec, the variables must be *outside* the quotation marks, as shown in Example 7-69.

Example 7-69 Passing variables to an EXEC

```
/* PASSING EXEC */
 num1 = 54; num2 = 49; num3 = 78
 "EXEC Add" num1 num2 num3
```

Issuing CMS and CP commands

Example 7-70 on page 253 shows a program that allows you to use files that are on another user's disk. The CP command LINK makes another user's disk available to you.

Example 7-70 Linking a minidisk using CMS and CP commands

```
/* LINKDISK EXEC */
/* For linking to disk 196 belonging to SRCWEB */
"LINK SRCWEB 196 200"    /* CP Command */
"ACCESS 200 B"           /* CMS Command */
```

After linking to the other user's disk, the EXEC issues the CMS command ACCESS to make the files on that user's disk accessible to you.

7.11.6 Detecting and correcting errors

Because they are interpreted, it is often relatively easy to recognize when things go wrong in a REXX EXEC. The EXEC will stop and REXX/VM will issue an error message if you do not conform to the rules of REXX.

Sometimes, however, the underlying cause is not readily apparent; for example, if the EXEC has been written to display the minimum amount of irrelevant information to its user (as is normally the case). Also, after an EXEC begins to exceed about 50 lines, it becomes much more difficult to follow what is going on.

Tracing can be achieved in a number of ways:

- By instructions within the program
- From CMS before the EXEC starts
- Dynamically, while the EXEC is running

You can use the REXX TRACE instruction to pinpoint problems quickly and easily, as explained here.

REXX TRACE instruction - options

You might want to examine different areas when testing and tracing an EXEC, and the TRACE instruction offers various options to handle those tasks, including those listed and described in Table 7-8.

Table 7-8 Useful TRACE options

Option	Description
A	(All) trace all clauses; that is, show the flow through every REXX clause.
C	(Commands) Trace all commands; that is, system commands, but not assignments, comments, etc.
I	(Intermediate) The most detailed of all the traces; shows how every variable and literal is interpreted.

Option	Description
A	(All) trace all clauses; that is, show the flow through every REXX clause.
N	(Negative or Normal) the default trace. Show only commands with negative return codes.
O	(Off) no trace.
R	(Results) trace all clauses and expressions. This is an example of a statement traced with TRACE R: 20 *-* 'LISTFILE' fn ft fm '(FORMAT LIFO' >>> "LISTFILE TEST FILE A (FORMAT LIFO"
S	(Scan) all remaining clauses in the data will be traced without being executed. Basic checking (for missing ENDS, and so on) is carried out and the trace is formatted as usual.

The most obvious way of using the TRACE instruction is to place it in your EXEC (for example, when writing a new EXEC); see Example 7-71.

Example 7-71 Tracing an EXEC

```
/* A NEW EXEC */
TRACE ‘ALL’
```

For more detailed information about REXX TRACE, refer to *z/VM V4 R2: REXX/VM Reference*, SC24-6035, and *z/VM V3 R1: REXX/VM User's Guide*, SC24-5962.

CMS commands

There are controls external to your EXEC that you can also use to detect errors, as listed here.

EXECTRAC setting

The CMS setting EXECTRAC will automatically set up a TRACE for the next EXEC that runs. To use it, enter the following command before running your EXEC:

```
set EXECTRAC on
```

You then find yourself in an interactive trace, which means that you can issue REXX clauses at the terminal, including a new TRACE instruction.

TS, TE, HI commands

There are also three CMS immediate commands that relate to running and tracing EXECs. These are called *immediate* commands because CMS acts on them immediately, rather than queuing them up in the normal way. These commands function in the following ways.

TS

TS starts REXX tracing in the middle of an EXEC, if necessary. This is similar to a dynamic SET EXECTRAC ON. (Note, however, that TS will *not* halt an interactive trace.)

TE

TE stops REXX tracing in the middle of an EXEC, if necessary (the opposite of what TS does). This is similar to a dynamic SET EXECTRAC OFF.

HI

HI offers a way to stop an EXEC before the next clause. It operates like inserting an EXIT instruction in the middle of the EXEC, and provides an exit from unruly or looping EXECs without doing anything too drastic. (A drastic way to stop an EXEC is to re-IPL CMS or enter a halt execution (HX) command.)

Note: For information about various commands, refer to *z/VM HELP Facility*, z/VM V4 R2: REXX/VM Reference, SC24-6035, and *z/VM V3 R1: REXX/VM User's Guide*, SC24-5962.

SIGNAL instruction

The SIGNAL instruction may be used for trapping errors or branching to another part of the EXEC.

The format of the SIGNAL instruction is shown in Example 7-72.

Example 7-72 Syntax of SIGNAL

```
SIGNAL ON condition ;
              OFF
```

This form of the instruction is designed to allow you to set a trap for special conditions (or turn it off again). Control passes to the first label with a name which matches the condition-name. So, for example, if you want to use the following instruction, you need to have a label somewhere in the EXEC file.

```
Signal on Syntax;
```

7.11.7 EXERCISE

Question 1

Indicate whether the following statements are accurate. (Y/N)

- REXX ignores spacing and is not case sensitive. _____.
- REXX is an interpreted language, but compiled versions are now available. _____.
- A REXX EXEC will run in any operating environment. _____.
- The REXX language is suitable only for advanced application developers. _____.
- REXX command syntax is “English-like” and does not use command abbreviations. _____.

Question 2

Given what you know so far about positional parsing, what would be the result of the following PARSE statement?

```
PARSE VALUE '12/25/90' WITH mm 4 dd 7 yy
```

Options are:

1. mm = 12/25/90 dd = 25/90 yy = 90
2. mm = 12 dd = 25 yy = 90
3. mm = 12/ dd = 25/ yy = 90
4. mm = 1 dd = 2 yy = 9

Question 3

Given the following example, what values would be placed in each variable by the PARSE statement?

```
string = 'jan,feb,mar,apr,may,june'
PARSE VAR string m1 m2 m3 m4 m5 m6
```

Question 4

Given the string A = 'THIS IS A TEST STRING' and the following relative parse statement:

```
PARSE VAR A 9 w1 +1 w2 -4 w3 +10 w4
```

What would be the contents of the variables w1, w2, w3, w4?

Question 5

Given the following REXX EXEC, what would be displayed on the screen?

```
climbing= 'steel'
tools= 'pegs'
column=  '    '
cost = 100
SAY climbing ¦¦ tools column '$'cost
```

Question 6

Complete these sentences by choosing from the list of options:

1. If it is not necessary to return a value, you would chose to write a _____?
2. A value must be returned to the calling routine. In this case, you would choose a _____?
3. A function or a subroutine is a series of instructions that will perform a specific _____?

Options are:

- ROUTINE
- SUBROUTINE
- FUNCTION
- TASK

Question 7

When the following REXX EXEC is executed, what would be displayed in the console?

```
/* STEM EXEC */

names. = 'NOBODY'
a = 3
names.a = 'ANITA'
newname = names.a
marks.newname = 56
street = names.street
b = names.a
student = names.b

SAY marks.fred
```

Question 8

What instruction can be used in a REXX EXEC to change the host environment to CMS?

Question 9

What instruction can be used to determine the Active Environment in which the REXX EXEC is executed?

Question 10

Write a REXX EXEC to read the first 6 lines of the file “test exec a” starting at line 50. The file also needs to be opened in a read only mode.

8

CMS pipelines

This chapter provides an introduction to CMS pipelines for those who already use REXX as a programming language and would like to extend their knowledge by using pipelines.

Objectives

After completing this chapter, you will be able to:

- Describe the concept of a pipeline
- Execute some pipeline commands from the CMS command line
- Write REXX EXECs using pipeline commands
- Redirect CP/CMS output
- Manipulate and reformat data
- Perform I/O functions
- Write simple pipelines with multiple data streams

© Copyright IBM Corp. 2007. All rights reserved.

8.1 Pipeline concepts

We are all familiar with the concept of a *pipeline*, whether through pictures that we have seen of oil pipelines or even through the plumbing in our own home. The purpose of a pipeline is to carry something from one place to another.

A CMS pipeline can be considered similar to this. A *CMS pipeline* is one or more pieces of pipe (stages) joined by connectors (the solid vertical bar character); see Figure 8-1. There are many types of stages, which can be used to perform many types of data manipulation.

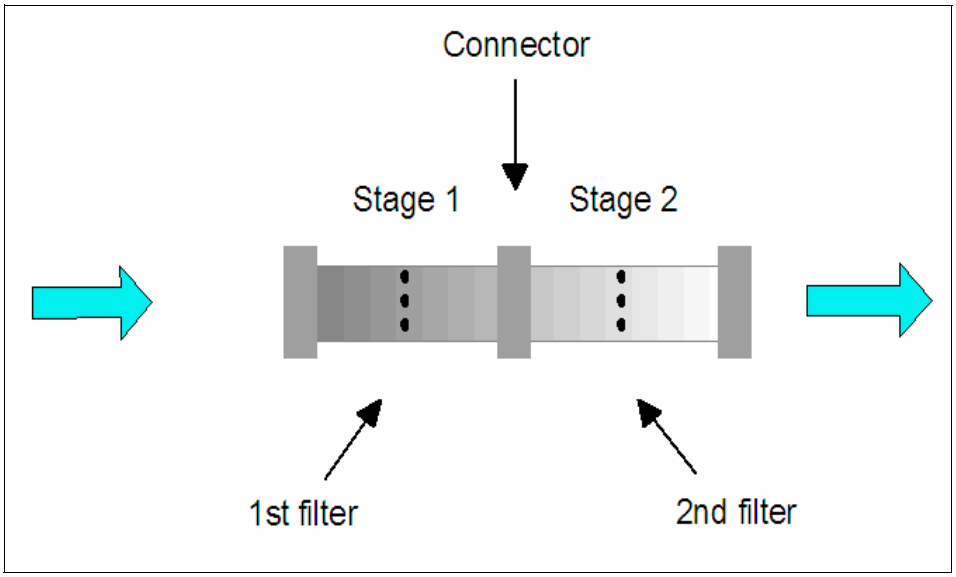


Figure 8-1 A CMS pipeline

This figure shows a very simple pipeline that can be entered from the command line, as shown here, and which will display the contents of a file on the virtual machine's console.

```
PIPE FILE cookies file a | console
```

This pipeline has two stages: the first is the input stage and the second is the output stage, and data flows from left to right. This could easily have been done using the **CMS TYPE** command, but it does demonstrate the basic function of a pipeline: getting data in, processing it in some way, and then getting data out. Each stage runs independently of any other, so each stage can be used as many times as necessary in the pipeline.

As we progress through this chapter we will add many more stages to your toolkit. We will also look at the device driver stages that are used to handle some of the potential input and output devices.

Keep in mind that you can get help by typing: `HELP PIPE MENU`. You can find a significant amount of information at that menu, including usage tips and examples that you can use in your pipelines.

When you have become acquainted with some of the stages and want more help when coding, type: `HELP PIPE stage` to go directly to the help that you require.

8.2 Developing pipelines

At this point, you have seen a simple pipeline to type the contents of a file. Now we develop this further in order to process the data in the file.

For the rest of this chapter it will be useful for you to duplicate some of the examples if you have a CMS session available. You can start by creating a file as follows:

```
XEDIT COOKIES FILE A
```

Next, cut and paste the following data into that file and file it on your A-disk. Then, when you see "cookies" in the example, you can try it out.

```
Who is bringing cookies to the party?
Pablo   chocolate chip cookies
Lisa   lemon drop cookies
```

The first pipe that we saw was:

```
PIPE file cookies file a | console
```

This lists the file on the console:

```
Who is bringing cookies to the party?
Pablo   chocolate chip cookies
Lisa   lemon drop cookies
```

Now, add a stage to split the file after each word:

```
PIPE file cookies file a | split | console
```

The following text will be displayed:

```
Who
is
bringing
cookies
to
the
party?
Pablo
chocolate
chip
cookies
Lisa
lemon
drop
cookies
```

Now, add a stage that sorts the list alphabetically:

```
PIPE file cookies file a | split | sort | console
```

This results in the following alphabetically sorted list:

```
bringing
chip
chocolate
cookies
cookies
cookies
drop
is
lemon
party?
the
to
Lisa
Pablo
Who
```

Finally, add a stage to eliminate duplicates:

```
PIPE file cookies file a | split | sort | unique | console
```

This results in two fewer cookies being displayed, as shown:

```
bringing
chip
chocolate
cookies
drop
is
lemon
party?
the
to
Lisa
Pablo
Who
```

Later in this chapter, we examine in more detail at some of the stages that were used in this example.

8.2.1 Device driver stages

Normally, the first and last stages of a pipe are *device driver stages*. Do not confuse these with the concept of device drivers on other platforms. A pipeline device driver stage reads from a device (for instance, a disk file, the stack, or the virtual machine console), or writes to a device. In some cases it can both read and write to the device, as shown in the following example:

```
pipe file cookies file a|change /Lisa/Tom/|file cook1 file a:
```

A pipeline can take data from one type of input device and write it to another type of device. Within the pipeline, data can be modified in almost any imaginable way.

The inherent characteristic of pipelines is that any stage can be connected to any other stage any number of times, because each obtains data and sends data through a device-independent standard interface. The pipeline usually processes one record or line at a time. The pipeline reads a record from input, processes it, and writes it to output. It continues until all the records from the input have been processed.

Table 8-1 on page 264 lists common device driver stages.

Table 8-1 Common pipeline device driver stages

Driver	Purpose
<	Reads a CMS file. `pipe **<** cookies file a | count words | console` This line displays the number of words in the file. **Note:** This must be a first stage, because it can only be used for input.
>	Replaces or creates a CMS file. `pipe < cookies file a | sort | **>** cook2 file a` This line places the sorted contents of a file into another. **Note:** This must *not* be a first stage, because it can only be used for output.
>>	Appends to or creates a CMS file. `pipe < cookies file a | **>>** cook2 file a` This line appends the contents of one file to another. **Note:** This must *not* be a first stage because it can only be used for output.
FILE	Reads, creates or appends to a CMS file. `pipe **file** cook2 file a | console` This line reads when a first stage writes or appends when not a first stage. It creates a file if it does not exist. It differs from < because it does not have to be a first stage.
CMS	Executes a CMS command and places the response in the pipeline. `pipe **cms** query cmslevel | chop , | console` This line discards the service level in the CMS version message. **Note:** CMS processes the commands in the same way that ADDRESS CMS does (that is, the command is converted to upper case).

Driver	Purpose
COMMAND	Issues CMS commands and writes response to pipeline. `pipe` **`command`** `Q DISK | > my disks a` This line issues the command and directs the output to a file. **Note:** COMMAND does not preprocess commands as CMS does. It works much like ADDRESS COMMAND. Synonyms are respected.
CP	Executes a CP command and places the response in the pipeline. `pipe` **`cp`** `screen inarea red rev` This line discards the output from the CP SCREEN command. Try adding an output stage `| console` to see the difference.
CONSOLE	Reads or writes to the terminal in line mode. `pipe < cookies file a |` **`console`** This line displays the contents of a file on the screen. `pipe` **`console`** `| > my output a` This line writes the keyboard input to a file.
STACK	Reads from or writes to the program stack, depending on whether or not it is a first stage. `pipe` **`stack`**`|drop 1|specs 1-6 1| console` This line would normally be used only in REXX EXECs, so do *not* try this one.
LITERAL	Writes an argument string into the pipe. `pipe` **`literal`** `hello there | console`
XMSG	Prefixes the contents of the pipeline with MSG and a blank so that it displayed in the XEDIT screen. First enter XEDIT COOKIES FILE A, then try this from the command line while you are in XEDIT: `pipe literal hello there! |` **`xmsg`** This line places the message in the XEDIT message area.

Driver	Purpose
READER	Reads from a virtual card reader. **Note:** READER must be a first stage.

PUNCH	Writes to a virtual punch.
	`pipe < cookies file a| punch`
	`cp close punch name cookies file`
	These lines punch a file without carriage control and no header record. If you issue QUERY PUNCH, you will see this file.

TAPE	Reads to or writes from tape. This example will (unless you really do have a tape) tell you that you do not have one available. `pipe file cookies file a | sort | tape tap2 wtm`

8.2.2 Pipelines in REXX

The examples of pipelines shown so far would easily fit on the command line and thus, pose no problem. However, pipelines can get much more complicated; take a look at the pipe in Example 8-1.

Example 8-1 A difficult pipe

```
/*EXEC to reformat a reader file*/

'pipe reader file 003 hold |find' '41'X||'|spec 2-* 1.80
|deblock netdata |find' 'C0'X||'|spec 2-* 1| console'
```

As you see, it is a bit difficult to follow the flow. So in Example 8-2 on page 267, it is formatted differently.

Example 8-2 A reformatted, easier pipe

```
/*EXEC to reformat a reader file*/
'pipe reader file 003 hold',
'|find' '41'X||'',  /* only data records               */
'|spec 2-* 1.80',   /* discard channel command and pad */
'|deblock netdata', /* deblock                         */
'|find' 'C0'X||'',  /* only data records               */
'|spec 2-* 1',      /* remove control character        */
'|console'
```

As shown in Figure 8-2 on page 272, each stage has been put on a separate line and we have added comments to help others read and understand what we are trying to achieve. Each stage is delimited by using a single quote at the start and end of a stage, and a comma at the end of the line is the REXX continuation character. This is the preferred way to write pipelines in a REXX EXEC.

8.2.3 More device drivers

Now that you have seen pipelines in REXX, in Table 8-2 we list other device drivers that you may find useful.

Table 8-2 More pipeline device drivers

Driver	Purpose

REXXVARS	Retrieves a REXX variable from within a REXX program
	
	`/* SHOWVARS EXEC */`
	`A=1`
	`B='Fred'`
	`C=A+3`
	`'PIPE` **`rexxvars`** `| console'`
	`exit`

Driver	Purpose

VAR	Retrieves or sets a variable. In the example, the variable `maxnum` is put into a file.
	
	`/* MAXFIND EXEC */`
	`parse arg x y`
	`maxnum=max(x,y)`
	`'pipe var maxnum | > max file a'`

STEM	Retrieves or sets an array of variables. The example will place each record of the COOKIES FILE A into an array of variables where each record will be referenced by the rec.n, where n is the record number.
	
	`/*LINENUM EXEC*/`
	`'pipe < cookies file a | stem recs.'`
	`do i=1 to recs.0`
	`say 'This is line 'i':' recs.i`
	`end`

REXX	Enables you to call a REXX program as a pipe stage. In the example, maxi is a REXX exec that will perform like a pipe stage. `pipe < number form a|split|rexx maxi|> output file`

8.2.4 Selective filters

A *filter* is a stage in a pipeline that takes its input from the left and then passes its output to the next stage on the right. A filter does not reference CMS files or virtual devices directly. Instead, it takes input from device drivers or other pipe stages.

A data filter is comparable to a water filter that you might place in a water pipeline. If you connect a water filter that has the function of removing sediment or large particles from the water, then clean water will be passed along the pipe after that filter.

When dealing with data pipelines, we want a data filter to perform a function, too. This function could be anything imaginable. For instance, data can be added to or deleted from a file. Alternatively, the contents of the records can be rearranged, based on columns of records.

Pipeline provides many built-in filters, some of which are outlined in Table 8-3 on page 269. In most cases, we direct the output to the console only as a simple

demonstration of what a data filter can do. Typically, though, you might direct output to another pipe or to some other device.

Table 8-3 Selective pipeline filters

Stage	Purpose
SPECS	Arranges the contents of input records. It can rearrange input records on a column basis, add literal strings or record numbers to the output, and convert fields. This can be a very complex stage, but you can try the following example to give an idea of what can be done. `pipe literal hi |spec 1-* 1 / there/ next| console`
LOCATE	Selects records that contain a specified target string of characters. `pipe < cookies file a | locate /Lisa/ | console` **Note:** This stage is case-dependent.
NLOCATE	Does not select records that contain a specified target string of characters. `pipe < cookies file a | nlocate /Lisa/ | console` **Note:** This stage is case-dependent.
FIND	Selects records that begin with a specified text. `pipe < cookies file a| find Pablo | console`
TOLABEL	Selects records from its primary input stream. The records selected are determined by the target string you specify. The specified target string must begin in the first column of an input record. `pipe < cookies file a | tolabel Pablo | console` (There are several other variations on the label concept.)
CHOP	Truncates each record after a specified column or string. `pipe < cookies file a | chop 12 | console`

Stage	Purpose

PAD	Extends records with one or more specified characters or blanks. You can extend a record on the left or the right.
	
	`/* PADQSRCH EXEC */`
	`'pipe cms QUERY SEARCH',`
	`'| pad left 35 .',`
	`'| pad right 50 .',`
	`'| > MYPAD OUTPUT A'`

STRIP	Removes leading or trailing characters from records. `pipe < mypad output a|strip .|console`
SPLIT	Splits records into multiple records. `pipe < mypad output a | split | console`
DROP	Drops records at the beginning or end of the input stream. `pipe < mypad output a | drop 2 | console`
SORT	Arranges records in ascending or descending order determined by additional criteria. `pipe < mypad output a | sort 22-24 | console`
UNIQUE	Compares each input record with the next one. By default, it discards a record that has the same contents as the following one. `pipe < mypad output a | unique 34-35 | console`
COUNT	Counts bytes, blank-delimited character strings, or records. `pipe < mypad output a | count lines | console`
PACK	Compresses records in a similar way to CMS COPYFILE (PACK. `pipe < mypad output a | pack | > mypack file a`
JOIN	Concatenates one or more input records into a single output record. `pipe < mypad output a | join 1 /--/ | console`

Stage	Purpose
XLATE	Translates data passing through the pipeline on a character by character basis. There are several types of translation that can be done using xlate; enter **HELP PIPE XLATE** for more information. `pipe < mypad output a | xlate lower | console`

8.2.5 Multistream pipelines

Many stage commands can use multiple input streams and multiple output streams. These stage commands are like houses that have several front doors and several back doors. So far we have been admitting records only through one front door (the primary input stream), and have been pushing them out of the corresponding back door (the primary output stream).

One of the simplest multistream pipelines is the capability to combine one or more pipelines into a single pipe. Example 8-3 shows two separate pipes that simply have an input stage and an output stage.

Example 8-3 Two separate pipes

```
pipe < file1 text a | > file1 save a

pipe < file2 text a | > file2 save a
```

These can be combined into one pipe which will also demonstrate the use of `(enchar ?)`, as shown in Example 8-4:

Example 8-4 Two pipes combined

```
/* two pipes one pipe command */
'pipe (endchar ?)',  /* start and define endchar  */
'< file1 text a | > file1 save a', /* first pipe  */
'?',                               /* endchar     */
'< file2 text a | > file2 save a'  /* second pipe */
```

The main thing to notice in Example 8-4 is `(endchar ?)`. This marks the end of a pipe and the start of the next pipe.

Some pipeline stages such as LOCATE have by default more than one output. LOCATE has one output for the records that it finds, and one output for records that do not match the locate criteria and that it does not find.

This can be demonstrated by using the pipe in Example 8-5. This pipe scans the PROFILE EXEC looking for the characters “Address”. Then it puts this in the file named ADDRESS FILE. The records that were discarded (that is, those without “Address”) are put into the file named REST FILE.

Example 8-5 LOCATE pipe

```
/*  Two outputs from stage */
'pipe (endchar ?)',
'< profile exec a | a: locate /Address/', /* first pipe*/
'| > ADDRESS FILE a',
'?',                                          /* endchar */
'a: | > REST FILE a'                        /* second pipe */
```

The main thing to notice in Example 8-5 is the use of `a:` which is a label that identifies where streams enter and leave a stage that has multiple streams. The diagram in Figure 8-2 illustrates this concept.

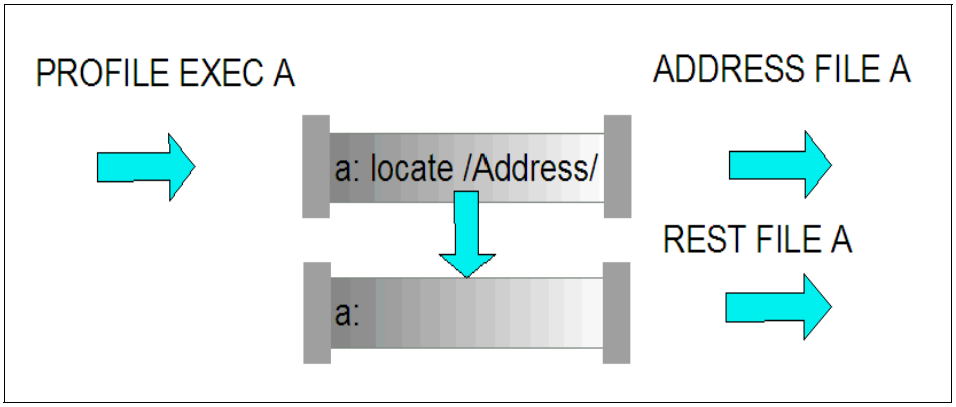


Figure 8-2 Using labels to identify the entrance and exit of stages

It may be necessary to join multiple input files together. There are special pipe stages for this task, as explained here:

- FANIN reads the records from the input files in order.
- FANINANY reads input records as and when they arrive at the stage.

As an example, you might have data coming in from a disk file, a tape file, and the console. FANIN would read in the order that input stages are specified.

Example 8-6 on page 273 shows a simple FANIN that merges two files and puts the output on the console. To try the example, to create a two files and put some text into them: FILE1 TEXT and FILE2 TEXT on the A-disk.

Example 8-6 A simple FANIN stage

```
/*       Fanin example           */
'PIPE (endchar ? ) < file1 text a', /* input file 1 */
'| f: fanin                      ', /* label */
'| console                       ', /* output */
' ?                               ', /* end of pipe */
' < file2 text a                  ', /* input file 2 */
'| f:' /* send input file 2 to label */
```

It may also be necessary to create more than one output data stream from a single input stream. Use FANOUT for this task.

In Example 8-7, FANOUT reads the PROFILE EXEC and then creates two files, one with any record that has `SET` in it, and another with any record that has `Address` in it.

Example 8-7 A simple FANOUT stage

```
/* fanout example */
'pipe (endchar ?) < profile exec a', /* input file */
 '| a: fanout',                     /* label */
 '| locate /SET/',                 /* 1st pipe stage */
 '| > SET data a ',                /* 1st output */
 '?',                              /* end of pipe */
 'a:',                             /* label input */
 '| locate /Address/',             /* 2nd pipe stage */
 '| > Address data a' /* 2nd output */
```

8.2.6 Reference

In this publication, we covered the basics of pipelines to get you started in your career as a CMS pipelines “plumber”. You can find more information by searching the Web with an argument that contains `CMS PIPELINES PLUMBER`. Also refer to the following publications:

- *z/VM CMS Pipelines User Guide*, SC24-6077
- *z/VM CMS Pipelines Reference*, SC24-6076

8.3 Exercises

1. Using a pipe, copy your PROFILE EXEC A to `profile save A`. It should be in lower case.

2. Perform the same operation as in exercise 1, but in a REXX exec.
3. Write an exec that will display an alphabetically sorted list of the users logged on your system. (Hint: Use Q NAMES.)
4. Write an exec that first lists the users who are logged on, and then lists the disconnected users (DSC) using FANOUT.
5. Display the HELP information for PIPE and find a pipe that allows you to take an input stream and then add the output of another input stream to the beginning of the first stream.

9

System administration tasks

Chapter 5, “Control Program for new users” on page 103, presents tasks that allow you to display your virtual machine information, and explains how to change various properties of your virtual machine.

In this chapter, we present the tasks that allow you to execute powerful CP commands and utilities, and to configure the properties of the entire z/VM system.

Objectives

After completing this chapter, you will be able to:

- Use privileged CP commands and utilities
- Configure your system
- Manage users
- Manage resources
- Communicate with service virtual machines
- Gain a high-level understanding of system installation and maintenance

© Copyright IBM Corp. 2007. All rights reserved.

9.1 Overview of system administration tasks

If you are a system administrator, you will be responsible for many areas that pertain to real resources and their allocation to users. You may even have responsibility for ensuring the integrity and security of the system.

System administration tasks are of several types:

Dynamic changes These changes are performed using CP commands to dynamically change both the system and the user environment.

Static changes These changes involve tailoring files and then making them available for system use. The two main files are the SYSTEM CONFIG file and the USER DIRECT file; their names describe their functions.

Backup and restore This task involves using z/VM functions, external products, or vendor products to ensure the integrity of the system.

Startup and shutdown The IPL and controlled shutdown of z/VM and its guests involves a knowledge of the hardware in addition to the CP commands necessary to achieve a successful shutdown.

This chapter provides an overview of these tasks. For detailed information about these topics, consult the referenced documents.

9.2 CP commands

In Chapter 5, “Control Program for new users” on page 103, you learn the CP commands that can be used by normal users to view and change their virtual environment. However, another level of commands exist that allow a user to control real system resources and to make changes that affect other users (this is similar to the root user in Linux).

The authorization for CP commands is based on the *privilege class*. The CP commands are assigned a privilege class based on the function they perform, such as system operation, spool processing, real device operation and general user.

In the CP Directory entry, users have their privilege classes assigned based on the tasks they need to perform; for example, primary operator, spool operator, I/O operator, performance specialist, or end user.

Table 9-1 lists some of the more useful commands that you may need to use. For more detailed information about all the privileged CP commands and utilities discussed later in this chapter, refer to *z/VM CP Commands and Utilities Reference*, SC24-6081, and to the **HELP** information.

Table 9-1 Some CP commands

Command	Description
ATTACH	Use ATTACH to logically connect a real device to a virtual machine. This appears to the virtual machine as if you plugged the device in.
AUTOLOG	Use AUTOLOG or XAUTOLOG to log on another virtual machine. This can be useful if a server virtual machine needs starting or restarting for any reason (for example, DIRMAINT).
CHANGE	Use CHANGE to change some characteristics of the reader, punch or printer spool files. This can be useful when you need to modify its classes or to hold files so that they cannot be printed and so on.
DETACH	Use DETACH to detach a real device from a virtual machine. Use this command with caution, because if the virtual machine is using the device, this may introduce problems.
CPACCESS	Use CPACCESS to grant access to special disks called PARM disks, which are used by CP.
CPLISTFILE	Use CPLISTFILE to list the content of special disks called PARM disks, which are used by CP.
CPRELEASE	Use CPRELEASE to release the access to special disks called PARM disks, which are used by CP.
DISABLE	Use DISABLE to disable access to terminal devices.
DEFINE	You can use DEFINE for many things. The more useful ones for an administrator may be: ► CPOWNED ► DEVICE ► TIMEZONE ► VSWITCHs and LANs
ENABLE	Use ENABLE to enable access to terminal devices. If users complain that their sessions will not start and they do not have a logo displayed, try using this before anything else.
FORCE	Use FORCE to log off a user to terminate immediately because it is stuck and cannot issue any command.

Command	Description
INDICATE	Use INDICATE to see a snapshot of resource utilization. This command can be used by system operators, system analysts, system programmers, and general users (classes B, C, E, G). The more useful INDICATE commands include: ► ACTIVE ► IO ► LOAD ► NSS ► PAGING ► QUEUES ► SPACES ► USER More detailed information about this command, refer to Chapter 10, “Performance” on page 321.
ORDER	Use ORDER to change the order of the reader, punch or printer spool files. This can be useful when you need to generate some z/VM components.
QUERY	Use QUERY to display many aspects of the system. There are many QUERY commands but the more useful ones for an administrator may be: ► CPDISKS ► CPOWNED ► NSS ► DASD
PURGE	Use PURGE to purge the reader, punch or printer spool files. This can be useful when you need to clean up some spool files.
SET	Use SET to set or change various properties of the system. There are many SET commands but the more useful ones for an administrator to use for performance actions are: ► CACHE ► MAXUSERS ► MDC ► QIOASSIST ► QUICKDSP ► RESERVED ► SHARE ► SRM ► THROTTLE More detailed information about this command, refer to Chapter 10, “Performance” on page 321.

Command	Description
SPXTAPE	Use SPXTAPE to back up or restore your spool files. In spool space there are reader, punch and printer files, and also system files called System Data Files. We will discuss some of them as NSS and DCSS in 9.13.1, “SPXTAPE” on page 303.
VARY	Use VARY to make real devices available or unavailable to the system and consequently to any virtual machine that runs under z/VM. ► VARY ON ► VARY OFF
WARNING	Use WARNING to alert users that some action will be taken (for example, communicating to users that the system will be IPLed within certain period of the time).

9.3 CP utilities

z/VM provides not only commands that can be executed by the control program, but also utilities that can be useful to administering the system. For more detailed information about privileged CP commands and utilities, refer to *z/VM CP Commands and Utilities Reference,* SC24-6081.

In this section, we explain the use of a few of the more commonly used ones in this topic to give you an insight into what they can do.

CPFMTXA

Use CPFMTXA to format, label, and allocate DASD volumes for CP usage such as paging, spooling, temporary disk, and directory. All CP volumes must be formatted in this way. The CPFMTXA program invokes the Device Support Facilities (ICKDSF) program to do the work.

CPSYNTAX

Use CPSYNTAX to check the syntax of the system configuration file. Use this utility with care, because it only checks the *syntax* of the file and not the actual configuration. If the configuration is configured improperly, you may have problems at your next IPL.

DDR

Use the DASD Dump Restore (DDR) utility to dump, copy, or restore data that resides on user minidisks or dedicated DASDs. This utility may also be used to restore or copy DASD data that resides on z/VM user tapes. There are two

versions: a CMS command, and an IPLable standalone program. For more information about this utility, refer to “DDR” on page 302.

DIRECTXA

Use DIRECTXA to create a user directory. DIRECTXA loads a CP readable copy of the USER DIRECT file that you have edited onto the disk area reserved for it. These areas, known as DRCT, will have been allocated by using CPFMTXA.

DISKMAP

Use DISKMAP to summarize the MDISK statements in the user directory. The output produced by DISKMAP shows gaps and overlaps between minidisk assignments. This is especially important when you are assigning new areas of disk to users. For more information about this utility, refer to “DISKMAP” on page 280.

DUMPLOAD

Use DUMPLOAD to process system abend dumps, standalone dumps, and virtual machine dumps. Sometimes a component of z/VM may suffer a failure and, to help in diagnosing the failure, z/VM will produce a dump of storage associated with the failure to specified dump areas or reader files. For more information about this utility, refer to “DUMPLOAD” on page 280.

HCPSADMP

Use the HCPSADMP EXEC to create a load module of the standalone dump utility, and write the module on a tape or disk. If you have a catastrophic failure of z/VM, it is useful for IBM support personnel to have as much information as possible to determine the cause of the error, and you can use a standalone dump to provide this information.

A standalone dump will allow you to dump storage to either tape or disk. Note that disk is much faster and will reduce the outage that you will experience if z/VM takes a hit.

SALIPL

Use the SALIPL module to install a copy of the Stand-Alone Program Loader (SAPL) in cylinder zero (0) in CKD DASD, or in blocks 5 to 207 of a FBA DASD. The SALIPL utility can run under CMS, or it can be loaded to run standalone. You need to write SAPL in the z/VM system resident (SYSRES) volume DASD in order to be able to IPL on it. For more information about this utility, refer to *zVM System Operation*, SC24-6121.

UTILITY

Use UTILITY to provide occasionally-used installation utility functions. You can create a standalone service utility tape for either (or both) ICKDSF and DDRXA. This is especially useful when creating tapes for a disaster recovery (DR) exercise, because you can create backups using these tapes which can be IPLed at your DR site to restore your z/VM system.

9.4 CP messages and codes

In this section we explain the logic and format of CP messages and codes.

System messages

In general, messages are issued to alert you to a problem, to request that you perform some action, or to provide information. Messages consist of a message identifier (for example, DMSACC017E) and message text; see Example 9-1. The identifier distinguishes messages from each other. The text is a phrase or sentence describing a condition that has occurred, or requesting a response from the user.

Example 9-1 Format of most message identifiers

```
xxxmmm###s or xxxmmm####s
```

The message format consists of four fields:

xxx The 3-character prefix indicates which z/VM component, facility, or feature, or which other product, contains the module that generated the message.

mmm The 3-character module code indicates which module generated the message. This field is usually an abbreviation of the name of the module in which the error occurred.

or **####** The numeric message number consists of three or four digits that are associated with the condition that caused the message to be generated.

s The 1-character severity code is a letter that indicates what kind of condition caused the message. The severity codes used by z/VM and their meanings are:

A - Immediate action required
D - Decision
E - Error
I - Information only

R - Response
S - Severe error
T - Terminating error
W - System wait (CP only), warning (all others)

System codes

Codes are generated by the system in response to either an action or lack of action that has been detected.

CP abend codes

Soft abends occur when an error condition can be isolated to a single virtual machine, or when system integrity is not endangered. CP abends and tries to automatically recover (auto-restart).

Hard abends occur when CP detects an error condition that it cannot isolate to a single virtual machine, or when system integrity is endangered.

Dumps

CP generates an abend dump whenever the system is restarted, or when a software error occurs while CP is still operational. Collect and save the dump so it can be used for problem diagnosis by your enterprise and by IBM service personnel.

Wait states

CP enters both enabled and disabled wait states. CP enters an enabled wait state when it is waiting for work. CP enters a disabled wait state when system operation is terminated due to an error, or when system shutdown is complete. z/VM service programs, such as HCPLDR, enter a disabled wait state when they terminate.

For more detailed information, refer to *CP Messages and Codes*, GC24-6119.

9.5 System configuration

The system configuration file (SYSTEM CONFIG on MAINT minidisk CFn) is one of the most important files on the system and is used by CP during IPL. The SYSTEM CONFIG file is a CMS file that can be edited using XEDIT and that is readable by CP.

In *9.2, "CP commands" on page 276*, we described the commands that could be issued to define various things. It is important to realize that some of the changes (for example, DEF TIMEZONE) that can be made will not be permanent and will

disappear after the next IPL. To ensure that such changes become permanent, you need to update SYSTEM CONFIG with the required changes.

The SYSTEM CONFIG file has many statements that define such things as:

- The devices that CP should bring online at IPL time
- The time zone that CP should select from a list of time zones at IPL time
- Whether CP should automatically attempt a warm start without changing the clock at IPL time
- The characters used as default terminal characters (such as line end and line delete)
- Whether CP should autolog special user IDs such as the accounting and symptom user IDs at IPL time

More details about this topic are covered 9.5, “System configuration” on page 282. For further information, refer to *CP Planning and Administration*, SC24-6083.

9.5.1 CP-owned DASD volumes

The SYSTEM CONFIG file contains statements that identify volumes that are specifically for the use of CP. These volumes must have been formatted and allocated by using the CPFMTXA utility discussed on page 279.

The CPFMTXA utility formats the disk into 4 K blocks and writes an allocation byte map on cylinder zero (0).

The byte map is simply a contiguous string of bytes, one for each cylinder of the disk. Each byte defines what that cylinder can be used for.

Figure 9-1 on page 284 illustrates formatting and allocating CP-owned volumes.

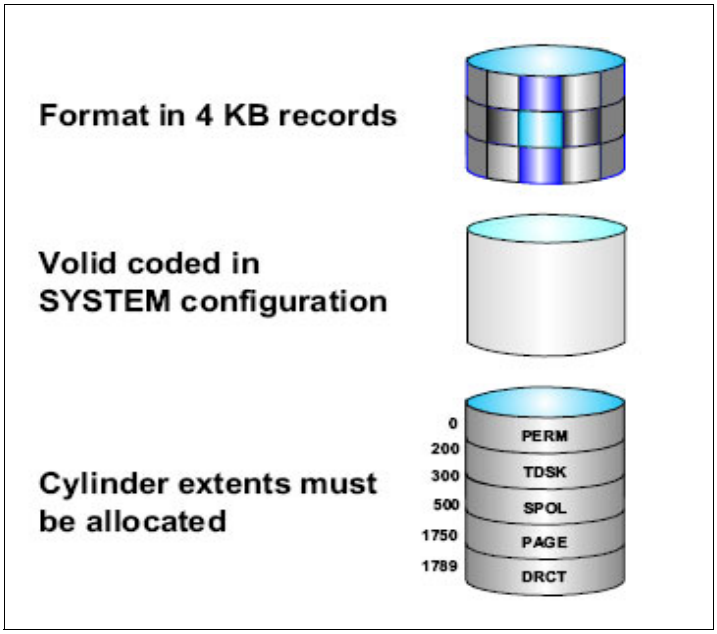


Figure 9-1 Formatting and allocating CP-owned volumes

The types of allocation are:

DRCT Area allocated to hold USER DIRECT file
SPOOL Area allocated to hold spool files
PAGE Area allocated to hold pages
TDISK Area allocated to hold temporary disks (tdisks)
PARM Area allocated to hold PARM minidisks
PERM Area allocated as permanent

Note: Use QUERY CPOWNED to display the list of CP-owned DASD volumes. If paging or spooling problems occur this can be checked to ensure that all required volumes available.

9.6 PARM disks

There are three minidisks that reside in space that has been allocated PARM, and these are read by CP at IPL time. On the MAINT user ID, they have the virtual device numbers CF1, CF2, and CF3.

The reason for having three minidisks is because that allows two minidisks to be taken away from CP for maintenance without affecting users. It also allows for a certain amount of resilience because an alternate can be used if corruption or hardware failure leads to one disk becoming unavailable.

Querying PARM disks

After you log on user ID MAINT, issue the command **QUERY CPDISKS** to query and display the status of the PARM disks as shown in Example 9-2.

Example 9-2 QUERY CPDISKS

```
q cpd
Label  Userid   Vdev Mode Stat Vol-ID Rdev Type   StartLoc     EndLoc
MNTCF1 MAINT    OCF1  A   R/O  LX6RES CD31 CKD          39        158
MNTCF2 MAINT    OCF2  B   R/O  LX6RES CD31 CKD         159        278
MNTCF3 MAINT    OCF3  C   R/O  LX6RES CD31 CKD         279        398
```

The example shows the following command response:

- Three PARM disks from user ID MAINT.
- Labels MNTCF1, MNTCF2, MNTCF3.
- Virtual devices CF1, CF2, and CF3 accessed mode A, B, and C, respectively.
- The disks are accessed as read only and reside in the volume-id LX6RES.
- The real device address CD31, CKD disk type.
- The PARM disk location starts at cylinder 39 and ends at cylinder 158.

9.6.1 Accessing the PARM disk

When logged on to the MAINT user ID, then in order to modify the SYSTEM CONFIG file, you must access the CF1 PARM disk in write mode. Even though you could have CP maintain R/O access to the disk while you are editing the files, it is recommended that you remove the disk from CP's list of accessed disks first by issuing the CPRELEASE command.

Follow this sequence of commands to access the CF1 PARM disk:

1. Release CP access to PARM disk accessed as letter A (CF1), link CF1 in write mode (MR) and access it as letter x:

   ```
   cprelease a
   link maint cf1 cf1 mr
   access cf1 x
   ```

2. After you xedit the SYSTEM CONFIG file and perform your modifications, save the SYSTEM CONFIG file, release the PARM disk, and restart the access to CP:

   ```
   release x (det
   ```

```
cpaccess maint cf1 a sr
```

Example 9-3 illustrates the entire sequence.

Example 9-3 CPRELEASE command and LINK to access CF1 PARM disk in write mode

```
cprelease a
link maint cf1 cf1 mr
access cf1 x
release x(det
cpaccess maint cf1 a rr
```

To make your job easier, you can write a simple exec to access the parm disks so that you can avoid typing all commands each time to get the access done, as shown in Example 9-4.

Example 9-4 ACCPARM EXEC - sample of a REXX exec to access PARM disk

```
/* ACCPARM EXEC SAMPLE */
'CPRELEASE A SYNC'
'VMLINK MAINT CF1 (WRITE FI'
'CPACCESS MAINT CF1 A SR'
exit
```

9.6.2 Displaying PARM disk content

When you are logged on to the MAINT user ID, you can display which files are on accessed PARM disks by using the **CPLISTFILE** command to display the files on CP-accessed minidisks, as shown in Example 9-5.

Example 9-5 CPLISTFILE command

```
cplistf * config
Filename Filetype FM Fmt LRecL    Records   Date     Time   Cache
LOGO     CONFIG   A  V      69         63 04/27/93 15:41:58 No
SYSTEM   CONFIG   A  F      80        250 05/03/07 13:04:48 No
```

Note: When CP accesses a minidisk, it reads the minidisk's file directory into storage. Any subsequent changes to the minidisk are *not* reflected into the file directory CP keeps in storage until you issue the CPACCESS command to have CP reaccess the minidisk.

For more detailed information about this topic, refer to *CP Planning and Administration*, SC24-6083.

9.7 CPLOAD MODULE

The CPLOAD MODULE is the CP nucleus, similar to the Linux kernel. The default module name is CPLOAD MODULE. You can have many CP modules in your installation, allowing for different conditions or providing backup. You select the CP module to be used:

- When you run Stand-Alone Program Loader Creation Utility (SALIPL)
- During IPL by overriding the SAPL defaults on the SAPL screen
- During SHUTDOWN REIPL by using the MODULE operand.

Although the file name of a CP module is variable, the file type must be MODULE. CP modules typically reside on a minidisk that is identified as a parm disk in the volume allocation table, but you can select the DASD volume and offset or extent from which SAPL will read. Therefore, you can have CP modules on multiple minidisks on multiple DASDs.

Note: Although SAIPL will accept a load origin when loading a CP module, it has no effect. CP will always relocate itself to location X'2000'.

In order to see which CPLOAD module was loaded after IPLling the system, you can issue the command **QUERY CPLOAD**, as shown in Example 9-6.

Example 9-6 Issuing QUERY CPLOAD command

```
q cpload
Module CPLOAD was loaded from minidisk on volume LX6RES at cylinder 39.
Parm disk number 1 is on volume LX6RES, cylinders 39 through 158.
Last start was a system IPL.
```

To check the level of CPLOAD module active in the system, you can issue the command **QUERY CPLEVEL**, as shown in Example 9-7.

Example 9-7 Issuing QUERY CPLEVEL command

```
q cplevel
z/VM Version 5 Release 3.0, service level 0701 (64-bit)
Generated at 05/02/07 16:28:04 EDT
IPL at 05/03/07 13:06:26 EDT
```

9.8 SYSTEM CONFIG file

The SYSTEM CONFIG file contains statements that define the characteristics of your z/VM system; Figure 9-2 illustrates a PARM disk.

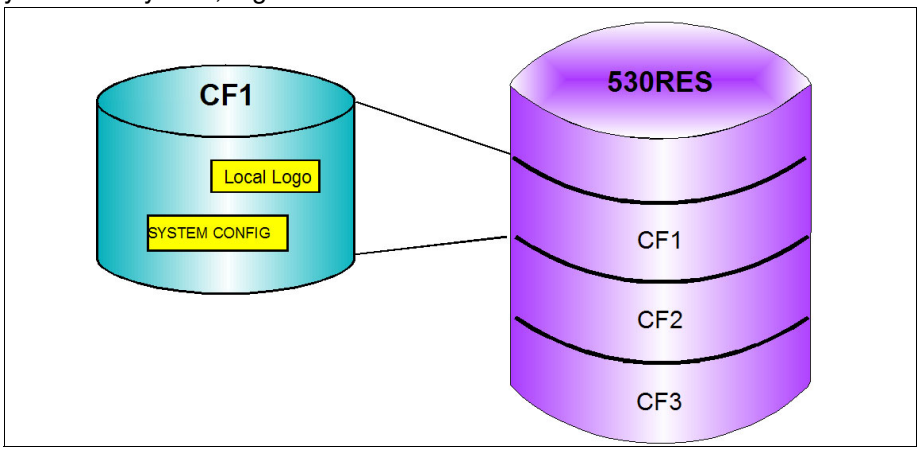


Figure 9-2 PARM disk

9.8.1 System Config file specifications

The following sections contain brief descriptions of some of the statements that you can specify in the system configuration file. For detailed information about all the system configuration statements and general rules for coding a system configuration file, see *z/VM CP Planning and Administration*, SC24-6083.

Minimum configuration

For the minimum configuration to get a system up and start running, define the three statements CP_OWNED, SYSTEM_RESIDENCE, and OPERATOR_CONSOLE and EMERGENCY_MESSAGE_CONSOLES, as explained here.

- CP_OWNED defines a volume to be CP-owned; see Example 9-8.

  After the system is running, you can use the DEFINE CPOWNED command to change this list.

Example 9-8 CP_OWNED statement

```
/*********************************************************************/
/*                    CP_Owned Volume Statements                     */
/*********************************************************************/
    CP_Owned   Slot   1  LX6RES
    CP_Owned   Slot   2  LX6SPL
```

```
CP_Owned   Slot   3  LX6PG1
CP_Owned   Slot   4  LX6WO1
CP_Owned   Slot   5  LX6WO2
CP_Owned   Slot   6  LX6PG2
CP_Owned   Slot   7  RESERVED
CP_Owned   Slot   8  RESERVED
CP_Owned   Slot   9  RESERVED
```

- SYSTEM_RESIDENCE

  This command specifies the location of the checkpoint and warm start areas; see Example 9-9.

Example 9-9 SYSTEM RESIDENCE statement

```
/*********************************************************************/
/*               Checkpoint and Warmstart Information                */
/*********************************************************************/
  System_Residence,
      Checkpoint  Volid LX6RES   From CYL 21  For 9 ,
      Warmstart   Volid LX6RES   From CYL 30  For 9
```

- OPERATOR_CONSOLE and EMERGENCY_MESSAGE_CONSOLES

  This command defines a list of console addresses from which CP should choose the operator console that receives the initialization messages during IPL, and also a list of consoles that CP should notify if there is an impending abnormal end or other system emergency; see Example 9-10.

  If you do not include this statement, CP uses the list specified on the OPERATOR_CONSOLES statement for the list of emergency consoles as well.

Example 9-10 Console statement

```
/*********************************************************************/
/*                      Console Definitions                          */
/*********************************************************************/
  Operator_Consoles     F003 F083 System_Console System_3270
  Emergency_Message_Consoles   F003 F083
```

Note: To reiterate, these three statements are all the system really needs to come up and start running.

Other important statements

You may also want to include the following statements:

- SYSTEM_IDENTIFIER and SYSTEM_IDENTIFIER_DEFAULT

Use this statement to define system identifiers for the processors on which you run z/VM; see Example 9-11. This is the identifier shown in the bottom right corner of your screen, when using the command line.

Example 9-11 SYSTEM IDENTIFIER

```
/*********************************************************************/
/*                  System_Identifier Information                    */
/*********************************************************************/
    System_Identifier_Default VMLINUX6
```

- TIMEZONE_DEFINITION and TIMEZONE_BOUNDARY

  Use this statement to enable the system to choose the correct local time zone definition; see Example 9-12.

Example 9-12 TIMEZONE DEFINITION

```
/*********************************************************************/
/*                        Timezone Definitions                       */
/*********************************************************************/
    Timezone_Definition EDT West 04.00.00
    Timezone_Definition EST West 05.00.00
    Timezone_Definition CDT West 05.00.00
    Timezone_Definition CST West 06.00.00
    Timezone_Definition MDT West 06.00.00
    Timezone_Definition MST West 07.00.00
    Timezone_Definition PDT West 07.00.00
    Timezone_Definition PST West 08.00.00
    Timezone_boundary on 2007-03-11 at 02:00:00 to EDT
    Timezone_boundary on 2007-11-04 at 02:00:00 to EST
    Timezone_boundary on 2008-03-09 at 02:00:00 to EDT
    Timezone_boundary on 2008-11-02 at 02:00:00 to EST
    Timezone_boundary on 2009-03-08 at 02:00:00 to EDT
    Timezone_boundary on 2009-11-01 at 02:00:00 to EST
```

- IODF

  This statement indicates that HCM and HCD will control the hardware and, optionally, the software I/O configuration.

- STORAGE

  This statement allocates real storage and trace frames.

- JOURNALING

  This statement specifies characteristics of the system's journaling facility. After the system is running, you can use the QUERY and SET JOURNAL commands to work with the journaling facility.

- PRIV_CLASSES

  This statement changes the privilege classes authorizing certain CP functions.

- SYSTEM_USERIDS

  This statement specifies user IDs that perform special functions during and after IPL. These user IDs identify the virtual machines that handle special records and system dump files. The operator and startup user IDs are also specified.

- USER VOLUME LIST

  USER VOLUME LIST is a DASD volume list that you specify in order to ensure that CP attaches them automatically to the system at IPL. Where a generic assignment has been made, the list can be overridden using the EXCLUDE and INCLUDE statements; see Example 9-13.

Example 9-13 USER VOLUME definition

```
USER_VOLUME_LIST
USER_VOLUME_EXCLUDE
USER_VOLUME_INCLUDE
```

Note: These statements can use wildcard characters (% and *) in defining volume serial identifiers.

- LOGO CONFIG

  This statement specifies the logo configuration file called LOGO CONFIG; see Example 9-14.

Example 9-14 LOGO definition

```
/********************************************************************/
/*                        Logo_Config                               */
/********************************************************************/
 Logo_Config   LOGO    CONFIG
```

9.9 LOGO CONFIG

You can define the characteristics of the logo that is displayed (see Example 9-15 on page 292) in a logo configuration file by using the exec DRAWLOGO and XEDIT profile X£DRWL£X that are provided as samples on MAINT 2C2 minidisk.

Example 9-15 z/VM LOGO screen

```
z/VM ONLINE


                                / VV           VVV MM        MM
                               /  VV         VVV  MMM      MMM
                      ZZZZZZ     /   VV       VVV   MMMM    MMMM
                         ZZ     /    VV     VVV     MM MM MM MM
                        ZZ     /     VV  VVV      MM  MMM  MM
                       ZZ     /      VVVVV       MM   M   MM
                      ZZ     /       VVV        MM       MM
                     ZZZZZZ /        V         MM       MM

                       built on IBM Virtualization Technology
z/VM 5.3.0 IESP

Fill in your USERID and PASSWORD and press ENTER
(Your password will not appear when you type it)
USERID   ===>
PASSWORD ===>

COMMAND  ===>
                                                          RUNNING   VMLINUX6
```

The LOGO CONFIG files defines where CP can find:

- Logos for local, logical, and attached screens.
- Status area definition: online message and input area definition information. The bit where you enter your user ID and password.
- Print separator pages for printers.

The logo configuration file contains the following four statements, which point to the files that define the areas of the logon screen:

CHOOSE_LOGO - the drawing area (upper center)
INPUT_AREA - where you enter the user ID and password (down left)
ONLINE_MESSAGE- where you put a message under the drawing area
STATUS - shows the status area and the system name (down right)

Note: You can dynamically change the LOGO configuration by issuing the command **REFRESH LOGOINFO LOGO CONFIG**.

9.10 User administration tasks

The z/VM directory (USER DIRECT) is a flat file that is used to manage the definitions of each user. Keeping track of these entries can quickly grow difficult if you have numerous users contained in it.

Using a directory manager such as IBM Directory Maintenance Facility (DIRMAINT) can greatly simplify the administration of users and DASD in a z/VM environment. Although DIRMAINT is a priced feature of z/VM, investing in it early can prevent many system administration issues in the future.

9.11 User directory

The z/VM user directory describes to CP the configuration and operating characteristics of each virtual machine. The source file of the z/VM user directory consists of directory control statements. The sample user directory is located on MAINT's 2CC minidisk and is called USER DIRECT. This file can be edited using XEDIT.

CP cannot read the source directory, so the DIRECTXA utility is used to create a CP readable version of the USER DIRECT. This compiles the file and places a copy of it in the area that has been allocated as DRCT on one of the CP owned volumes. The format of the command to execute the DIRECTXA utility is:

```
DIRECTXA fn ft
```

Where `fn` is the file name of your directory (USER is the DEFAULT fn), and `ft` is the file type of your directory (DIRECT is the DEFAULT ft).

9.11.1 DISKMAP

When adding users to the USER DIRECT it is desirable to ensure, when you allocate disk space to users, that you do not define minidisks that overlap either other users' minidisks or system areas. You can achieve this by using the DISKMAP utility.

The DISKMAP EXEC summarizes the MDISK statements in the user directory in a file called USER DISKMAP. This file contains information about the directory MDISK statements and it is organized by CP volume label. Gaps between minidisks and overlapping minidisks are flagged. Some overlaps are acceptable, when you see a full minidisk definition as MAINT 123 mdisk.

However, some overlaps are errors that must be corrected before use. Run DISKMAP and examine it before any changes are made to the directory.

Note: DISKMAP does *not* replace the EDIT option of the DIRECTXA command. Use both to check your directory after changes.

9.11.2 USER DIRECT control statements

DIRECTORY

The DIRECTORY control statement defines to CP the device on which you have allocated space for the object directory. The DIRECTORY control statement must be the first statement in your source directory; see Example 9-16.

Example 9-16 Example of a directory control statement entry

```
DIRECTORY 123 3390 LX6RES
```

PROFILE

If you are likely to have many users with very similar requirements, you can group together a sequence of statements that are required by all of them and place them in a PROFILE. This profile can then be used by putting an INCLUDE statement after the USER statement for all of the users. In the directory that is supplied with z/VM, there is a sample profile definition called IBMDFLT which includes statements as shown in Example 9-17.

Example 9-17 PROFILE IBMDFLT

```
PROFILE IBMDFLT
   I CMS
   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 MAINT 0402 0402 RR
   LINK MAINT 0401 0401 RR
   LINK MAINT 0405 0405 RR
```

More detailed information about this topic, refer to *z/VM: CP Planning and Administration*, SC24-6083.

USER

A user statement flags the beginning of a virtual machine definition. It contains a user ID, password, logon and maximum storage, and privilege classes.

Example 9-18 Example of a user definition entry

```
USER EDI PASSWORD 32M 32M G
   INCLUDE IBMDFLT
   MACHINE ESA
   OPTION QUICKDSP DIAG88
   MDISK 0191 3390 11 5 DKCD37 MR
```

Where:

- User ID is the virtual machine name (limited to eight characters). In the example, `USER = EDI`.
- Password is limited to eight characters. In the example, the password *is* `PASSWORD`.

  (There is a file called RPWLIST DATA, which lists passwords that cannot be used.)
- Minimum storage is the amount allocated when the machine first logs on. In the example, `32M` is the minimum storage.
- Maximum storage is the storage limit that can be defined by the user. In the example, `32M` is the maximum storage.
- Privilege class refers to one or more letters and numbers that represent the authority level you have to issue CP commands. In the example, `G` is the privilege class.

9.11.3 Adding guest virtual machines

To change your directory, follow these steps:

1. Log on the user ID MAINT.
2. Edit the file that contains the source directory USER DIRECT.
3. Add, delete, or change directory entries as required.
4. Bring your modified directory online by entering the command to start the DIRECTXA utility.

Note: Before bringing a directory online, if you want to check the directory for errors, use the EDIT option of the DIRECTXA utility; see Example 9-19.

Example 9-19 Checking directory errors

```
DIRECTXA USER (EDIT
```

```
where USER is the filename of the intire z/VM directory.
```

For more information about this topic, refer to the chapter on creating and updating a user directory in *z/VM: CP Planning and Administration*, SC24-6083.

9.11.4 DIRMAINT overview

z/VM Directory Maintenance Feature (DirMaint) is a CMS application that helps manage your VM directory. Directory statements can be added, deleted, or altered using the DirMaint directory statement-like commands. DirMaint provides automated validation and disk allocation routines to reduce the chance of operator error. Dirmaint is an interactive, multi-user application.

DIRMAINT uses two virtual machines, as explained here.

DIRMAINT - the primary server

The DIRMAINT server handles all aspects of source directory manipulation and controls the actions of all other servers. There is only one DIRMAINT server. To place directory changes online and log the results, the directory maintenance service machines use the DIRECTXA command. The virtual machine has these functions:

- It owns the CP source directory.
- It receives transactions from authorized users.
- It verifies that the transactions are valid.
- It makes the appropriate updates to the source directory.

DATAMOVE - for DASD-related tasks

A DATAMOVE server is responsible for manipulating minidisks on behalf of the DIRMAINT server. There can be multiple copies of the DATAMOVE server. The main functions are to:

- Format newly allocated DASD space for the user.
- Format a new mdisk to receive files from a user's existing mdisk, and copy files from this user's existing mdisk to the new mdisk.
- Format an old mdisk being deallocated again to prevent exposure of any residual data to the next user.

How DIRMAINT works

The Dirmaint user ID is the main server that controls, validates and distributes all requests received from users. If the user needs to change DASD characteristics, DIRMAINT checks the command, validates it, creates a WORKUNIT and asks DATAMOVE to execute the task. After DATAMOVE finishes it, the WORKUNIT is closed and the USER DIRECT entry is updated.

While a user ID is being worked on, the directory entry is locked to prevent any other authorized user from concurrently updating it.

Dirmaint HELP facility

To see the Dirmaint command help, issue the command **DIRM ?** or **DIRM HELP,** then position the cursor under *ADVH and press Enter. You will see the DIRMAINT commands list.

Dirmaint commands overview

The Dirmaint command must be preceded by the abbreviation **DIRM.** This routes the command to the DIRMAINT service machine, where the service machine does validation checking and either processes the request or rejects it with an appropriate message.

Here are examples of commands for user management:

ADD	To add a new user directory entry
REP	To replace an existing user directory entry
LOCK	To lock an existing user directory entry
UNL	To unlock an existing user directory entry

Here are examples of commands for mdisk management:

AMD™	To add a new mdisk user directory entry
CMD	To change a mdisk user directory entry characteristic
DMD	To delete a mdisk user directory entry

Here are examples of commands for administration:

DIRM CHKSUM	To verify the integrity of the source directory
DIRM DIRECT	To place the current directory structure online
DIRM SHUTDOWN	To shut down the DIRMAINT server
DIRM DAT SHUTDOWN	To shut down the DATAMOVE use ID

For further information, refer to *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6135, and *Directory Maintenance Facility Commands Reference*, SC24-6133.

9.11.5 Adding guest virtual machines using DIRMAINT

To add guest virtual machines by using DIRMAINT, follow these steps. You need to create the user directory entry and then have Dirmaint add the user ID.

1. Log on to your z/VM system using MAINT user and create the CMS virtual machine GUEST1 DIRECT file by issuing **XEDIT GUEST1 DIRECT**.

2. Add the statements shown in Example 14-3.

Example 9-20 CMS user directory

```
USER GUEST1 12345678 16M 64M G
   INCLUDE LNXDFLT
   IPL CMS PARM AUTOCR
   MACHINE ESA
   MDISK 191 3390 USRPK1 510 10 MR ALL ALL ALL
   MDISK 0F00 3390 DEVNO 5900 MR ALL ALL ALL
   MDISK 0F01 3390 DEVNO 5900 MR ALL ALL ALL
   MDISK 0F02 3390 DEVNO 5900 MR ALL ALL ALL
```

3. File the changes and use the following command to add the user definition to the directory:

   ```
   DIRM FOR GUEST1 ADD
   ```

   You should see an output similar to that shown in Example 14-4.

Example 9-21 DIRMAINT command add messages

```
DVHXMT1191I Your ADD request has been sent for processing.
DVHREQ2288I Your ADD request for GUEST1 at * has been accepted.
DVHBIU3450I The source for directory entry GUEST1 has been updated.
DVHBIU3424I The next ONLINE will take place immediately.
DVHDRC3451I The next ONLINE will take place via delta object directory.
DVHBIU3428I Changes made to directory entry GUEST1 have been placed online.
DVHREQ2289I Your ADD request for GUEST1 at * has completed, with RC
DVHREQ2289I = 0.
```

4. GUEST1 is now available for use.

> **Note:** To modify a user definition that is already defined, you must use the command **`DIRM FOR userid REP`** to replace the existing directory entry.
>
> First, however, you must put the user definition entry into lock mode by using the command **`DIRM FOR userid LOCK`**.

For more information about the structure of a directory entry, see the chapter on creating and updating a user directory in *CP Planning and Administration*, SC24-6083.

9.12 Managing storage

The amount of real storage you need depends upon the type of work your installation will be performing and the number and size of the virtual machines you define. Your system might include many virtual machines, and each one requires some real storage.

For example, you may find that the processor does not have enough storage for your system to provide acceptable response times. If you find your system's response time to be slow or erratic, you may need to do one or both of the following:

- Add more real storage.
- Run fewer or smaller virtual machines.

The minimum amount of real storage that is required to generate and operate z/VM is 32 MB. The maximum amount of real storage that z/VM supports is 256 GB.

There are commands that allow you to check real storage usage. These are:

QUERY FRAMES Use to display the status of host real storage.

QUERY MDC Use from a Class B user to query minidisk cache settings for the entire system, for a real device, an active minidisk, or a minidisk defined in the directory.

9.12.1 NSS and DCSS

When running guests that have the same function, z/VM has facilities to allow groups of users to share applications, data, and operating systems. The shared data and code is stored by CP in the dynamic paging area and is accessed by users as Named Saved Systems (NSS) and Discontiguous Shared Segments (DCSS). These are accessed by the guest as part of their virtual storage and therefore appear transparent to whatever is running in the guest.

From a performance perspective, there are benefits to this support: it substantially reduces the amount of real storage that we need, and provides better performance for a guest.

If viewed from a Linux perspective, having a large part of the kernel resident in storage would speed up the boot of the operating system significantly. A guest would boot a NSS called, for example IPL LNXTST, instead of using a virtual device number such as IPL 580.

The most common example of a NSS in z/VM is CMS. And if you issue the command Q NSS ALL (from a privileged user), you will see many other functions (such as HELP, CMS Pipeline and NLS or National Language Support), that are DCSSs and which benefit using this support.

For more detailed information about these topics, refer to *z/VM Saved Segments Planning and Administration*, SC24-6116, and *z/VM Virtual Machine Operation,* SC24-6128.

If you are using Linux, you can find more detailed information in *Linux on System z Device Driver, Features, and Commands*, SC33-8289.

9.12.2 Querying NSS

There are commands that you can use to display information about saved systems and segments. Example 9-22 shows the output after issuing the command **q nss name edi**, which queries a specific NSS with the name of EDI.

Example 9-22 Querying a specific NSS named EDI

```
q nss name edi
OWNERID  FILE TYPE CL RECS DATE  TIME     FILENAME FILETYPE ORIGINID
*NSS     0067 NSS  S  0001 06/12 10:49:25 EDI      DCSS     EDI2
*NSS     0068 NSS  A  1302 06/12 10:58:34 EDI      NSS      EDI2
Ready; T=0.01/0.01 15:55:35
```

Example 9-23 shows the output after issuing the command **q nss name CMS map**, which queries a specific NSS with a location map.

Example 9-23 Querying NSS with location map

```
query nss name CMS map
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0065 CMS      NSS      0000256K 00000  0000D   EW  P  00012   00-15     NO
                                00020  00023   EW
                                00F00  013FF   SR
0071 CMS      NSS      0000256K 00000  0000D   EW  P  00003   00-15     NO
                                00020  00023   EW
                                00F00  013FF   SR
0073 CMS      NSS      0000256K 00000  0000D   EW  A  00002   00-15     NO
                                00020  00023   EW
                                00F00  013FF   SR
Ready; T=0.01/0.01 16:23:23
```

It is worth paying attention to the column class that specifies the class of files to be queried. The operand *class* can be one of the following:

- Skeleton (S), when you define the NSS but have not saved it yet.
- Available nonrestricted (A) means the NSS is in production and active.
- Pending purge (P- means the NSS is about to be purged.
- Available restricted (R) means available, but with restricted access.

Example 9-24 shows the output after issuing the command **q nss users edi**, which queries the NSS users named EDI.

Example 9-24 Querying NSS EDI users

```
q nss users edi
FILE FILENAME FILETYPE CLASS
0068 EDI      NSS      A
EDI2
Ready; T=0.01/0.01 16:24:54
```

There are different types of NSS, as described here:

NSS	Named saved system
DCSS	Discontiguous saved segment
DCSS-S	Segment space
DCSS-M	Member segment

For more details, refer to *CP Commands and Utilities Reference*.

9.13 Backing up and restoring data

One of the major responsibilities a z/VM system administrator is to ensure the integrity of the system. This involves backing up or saving essential data on a regular basis.

DASD volumes

First the administrator must decide which volumes to back up; here are some suggested volumes:

- z/VM CP-owned disks
- Each major guest's virtual DASD
- Spool files

Note: There is no need to back up PAGING volumes because their contents are valid only when the z/VM system is running.

z/VM provides several programs to perform backup and restore:

- DASD Dump Restore (DDR) program

  DDR is a utility program that allows the administrator to back up and restore minidisks and complete DASD volumes. There is no option to do incremental backup.

  There are two ways of running the DDR program:

  – In CMS environment, by issuing the command **`DDR.`**
  – In a standalone way through a utility DDRXA, by issuing the command **`IPL tapeaddr`**.

  Note: DDRXA does not require an operating system, so using a backup tape is useful in disaster situations and in disaster recovery plans.

- SPXTAPE

  SPXTAPE is a program for backing up and restoring all spool files. As you know, in the spool area there are common spool files and system data files. The administrator has to take care of all files and back them up on a tape, using SPXTAPE, in a timely manner.

DDR

To start a backup of system disks, follow these steps:

1. Log on the user ID MAINT.
2. Attach a tape by issuing the command **`ATT tapeadr MAINT 181`**.
3. Verify which virtual address you want to back up by issuing **`Q V DA`**.
4. Write down the addresses so you can control the order in which you will do the backups.
5. Issue these commands, as shown in Example 9-25 on page 303:

```
DDR
IN 123 3390 LNX6RES
OUT 181 3480
DUMP ALL
YES
```

6. When the backup is complete, keep the tape in a secure location and record, at a minimum, information about the DISK volume label, date, and how long you have to keep it.

Note: The userid MAINT must have all system disks defined as minidisks in its directory entry. The minidisk virtual address defined is used as input to DDR command.

In Example 9-25, we are logged on using the MAINT user ID, and we issued a backup of the LX6RES with virtual address 123 to the tape attached as 181. The DDR command was DUMP ALL, in order to copy the entire disk.

Example 9-25 DDR example

```
ddr
z/VM DASD DUMP/RESTORE PROGRAM
ENTER:
in 123 3390 lx6res
ENTER:
out 181 3480
ENTER:
dump all
HCPDDR708E INVALID INPUT OR OUTPUT DEFINITION
ENTER:
HCPDDR711D VOLID READ IS TAP001
DO YOU WISH TO CONTINUE?  RESPOND YES, NO OR REREAD:
YES
DUMPING ALL
END OF DUMP
ENTER:
END OF JOB
PRT FILE 0227 SENT FROM MAINT    PRT WAS 0227 RECS 0006 CPY  001 A NOHOLD
NOKEEP
```

9.13.1 SPXTAPE

In this section, we explain how to use SPXTAPE to back up and restore spool files.

Spool files

In Chapter 5, “Control Program for new users” on page 103, we introduce the concept of spool devices. The normal spool devices you will see are reader, punch and print. The files associated with these devices reside on disk in an area on a CP owned disk that has been allocated as spool.

Types of spool files

There are standard spool files and system data files.

Standard spool files are printer, reader and punch spool files. In addition, special spool files called System Data Files (SDF) are used to store data associated with several system functions:

- Image libraries
- National language support files, such as message repository files

- Named saved systems
- Saved segments
- System trace files
- User class restructure files

SPXTAPE means “spool to tape”, and it is used to back up and restore files from the spool area. You can use SPXTAPE to selectively store on tape and retrieve standard spool files and system data files. SPXTAPE allows the operator to avoid an overload in the spooling area and quickly store the files temporarily on tape. The functions are:

1. Save standard spool files and system data files on tape
2. Restore SPXTAPE-format files from tape to the spooling system

To back up the spool files, you need to have a real tape attached. Then issue the command SPXTAPE DUMP to back up the files from spool and dump all the standard spool files to tape (see Example 9-26). To restore all spool files from tape, issue the command SPXTAPE LOAD.

Example 9-26 SPXTAPE DUMP and SPXTAPE LOAD commands

```
spxtape dump vdev1-vdev2 all run
spxtape load vdev1-vdev2 all run
```

9.14 Advanced DASD services under z/VM

z/VM supports some of the advanced DASD services or storage subsystem functions found in today’s Enterprise Storage Subsystems such as IBM TotalStorage DS8000, IBM TotalStorage DS6000™, IBM TotalStorage ESS, and so on. In this section we briefly explain the following advanced DASD functions:

- FlashCopy
- Peer-to-Peer Remote Copy (PPRC)
- Parallel Access Volumes (PAV)

Note: All of the advanced DASD functions discussed here will work only if the function is supported by the Storage Subsystem hardware unit used with z/VM.

A detailed discussion of these topics is beyond the scope of this book, so only simple explanations are provided here. However, for FlashCopy, we explain the z/VM **FLASHCOPY** command syntax.

9.14.1 FlashCopy

FlashCopy is an advanced “copy service” provided by the Enterprise Storage Subsystems from IBM. FlashCopy is an “instant” T0 (time zero) copy of a source volume to a target volume. FlashCopy can be used even when the source volume is in use.

z/VM FlashCopy works as follows:

1. Request copy from the source to the target.
2. The FlashCopy relationship is created between the volumes[1].
3. Tracks are copied from the source to the target.
4. Attempts to read or write data that is already copied are processed as normal.
5. Attempts to read a target track not yet copied are intercepted and data is obtained from the source.
6. Attempts to write a source track not yet copied are intercepted and source track is copied to the target before the update occurs.

Figure 9-3 illustrates a FlashCopy transaction.

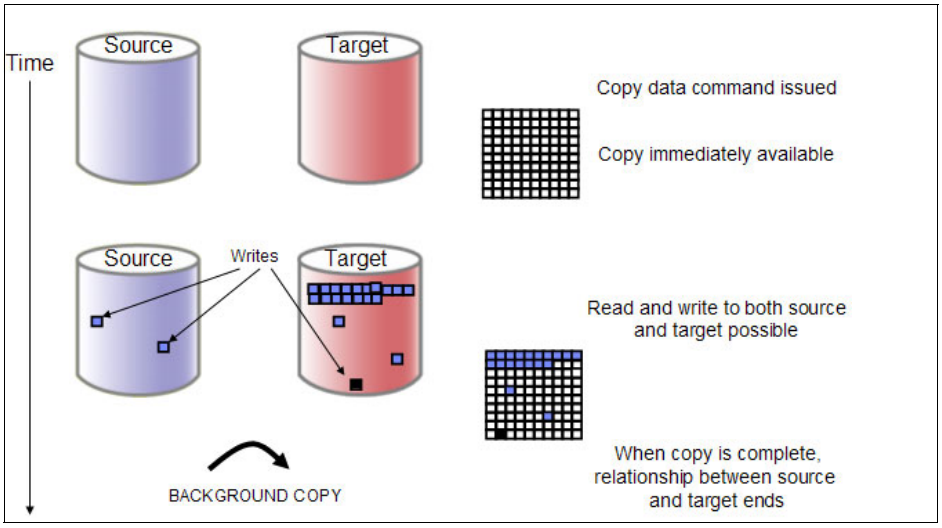


Figure 9-3 FlashCopy illustrated

[1] After the relationship is established, the source and target volumes are available for read and write operations. You do not have to wait for the background copy to complete in order to use the source or target.

z/VM V5.3 supports the new FlashCopy V2 feature of IBM System Storage™ disk storage devices. With FlashCopy V2, you can specify up to 12 target devices for a source (V1 supported one-to-one relationships only).

FlashCopy V2 also allows the source and target to be in different Logical Control Units (LCUs). z/VM allows guest FlashCopy via the CCW channel program support, but it is restricted to dedicated or full-pack minidisks. Example 9-27 demonstrates the **FLASHCOPY** command.

Example 9-27 z/VM **FLASHCOPY** *command*

```
FLASHCOPY E700 0 END E800 0
END
```

Note: z/VM Class B privilege or higher is required to execute the **FLASHCOPY** command.

9.14.2 Peer-to-Peer Remote Copy (PPRC)

Peer-to-Peer Remote Copy (PPRC) is yet another advanced “copy service” found on the IBM Enterprise Storage Subsystems. PPRC is generally used to continuously mirror a storage volume from one control unit to another control unit at a remote site. Note the following points:

- In the synchronous form of the PPRC, I/O is only considered to be complete when update to both the primary and the secondary have completed.
- In the asynchronous form of PPRC, tracks on the primary to be duplicated to the secondary will be flagged when time permits.

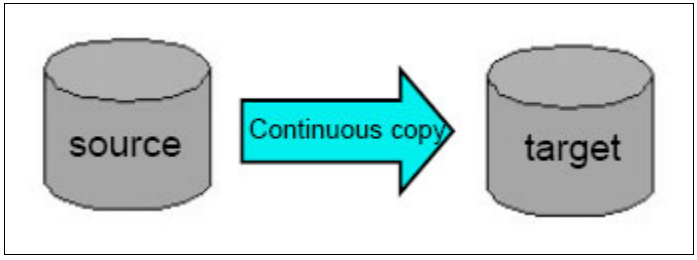


Figure 9-4 PPRC is used for continuous copy requirements

z/VM supports PPRC for continuous copy of source disk changes to target. Native support is via ICKDSF Release 17 running in CMS virtual machine and guest support via CCW channel program support. The guest support is restricted to dedicated or full-pack minidisks. The **CP QUERY DASD DETAILS** command

displays PPRC volume status (Primary, Secondary, or Cascading Intermediate) when a PPRC link is active for a disk.

9.14.3 Parallel Access Volumes (PAV)

Yet another advanced feature of IBM Storage Subsystems, the Parallel Access Volume (PAV) hardware feature allows you to configure one or more logical DASD volumes, each with a base and one or more alias subchannels; see Figure 9-5.

Note that the alias subchannels do not have their own data space; they are “shadows” of the space. This architecture allows concurrent accesses to a volume through its base and associated alias subchannels.

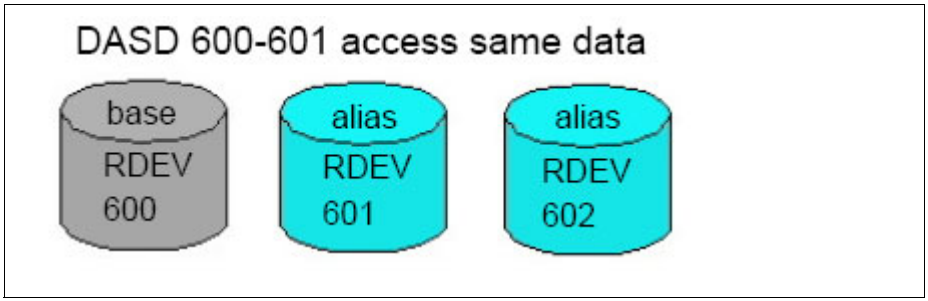


Figure 9-5 Parallel Access Volumes illustration

Note: A dedicated PAV volume may not have its alias subchannels spread among multiple guests. The use of shared minidisks provides that functionality.

HyperPAV

The HyperPAV function potentially reduces the number of alias-device addresses needed for parallel I/O operations, because HyperPAVs are dynamically bound to a base device for each I/O operation instead of being bound statically like basic PAVs.

HyperPAV function is provided by the IBM System Storage DS8000 and later family of disk storage systems. z/VM support for HyperPAV was introduced in V5.3.

z/VM provides support for HyperPAV volumes as linkable minidisks for guest operating systems, such as z/OS, that exploit the HyperPAV architecture. This support is also designed to transparently provide the potential benefits of

HyperPAV volumes for minidisks owned or shared by guests that do not specifically exploit HyperPAV volumes, such as Linux and CMS.

9.14.4 System disk maintenance

All system DASD areas should be monitored to avoid unnecessary outages of the z/VM system. In this section we explain how to add space, if needed.

Adding space

Adding space to both spool and paging is the same process so, with a little substitution, the following process applies to both.

The **Q ALLOC SPOOL** command shows how much space are you using at the moment. If the percentage looks too high, you may decide to add more spool DASDs; see Example 9-28. Alternatively, you could purge unimportant files.

Example 9-28 QUERY ALLOC

```
q alloc spool
                EXTENT     EXTENT  TOTAL  PAGES   HIGH    %
VOLID  RDEV      START        END  PAGES IN USE   PAGE USED
------ ---- ---------- ---------- ------ ------ ------ ----
LX6SPL CD34          1       3338 600840  36574  81800   6%
                                  ------ ------         ----
SUMMARY                           600840  36574           6%
USABLE                            600840  36574
6%
```

Procedure to dynamically add spool space

The following procedure will allow you to dynamically add more spool space. Before starting this procedure, read the detailed information about this subject in *CP Planning and Administration*.

Format and allocate using CPFMTXA, as follows.

1. Log on MAINT.
2. Attach a free disk to MAINT (CD39, as the disk address example):

   ```
   att CD39 *
   ```

3. Format the disk using CPFMTXA:

   ```
   CPFMTXA
   CD39
   SPL001 (example of disk label)
   0 end (all cylinders)
   YES
   ```

4. Wait until the end of disk formatting. It will show the following messages:

```
FORMATTING OF CYLINDER 3338 ENDED AT: 11:29:36
CYLINDER ALLOCATION CURRENTLY IS AS FOLLOWS:
TYPE     START      END        TOTAL
----          -----           ---             -----
PERM     0               3338         3339
```

5. When finished, you have to allocate the disk as SPOOL, typing the allocation type, the start cylinder, and the total amount of cylinders reserved to spool:

```
PERM 0 0
SPOL 1 END
END
```

At this point the formatting and allocation is done, and the disk is ready to use for spool.

Add spool disk in a slot into SYSTEM CONFIG file

This update will be activated only after the system is IPLed.

1. Access the PARM disk.
2. Use XEDIT to edit the SYSTEM CONFIG file.
3. Go to CP_Owned Statements and look at the sequence of slots defined, as shown here:

```
/*********************************************************************/
/*              CP_Owned Volume Statements                 */
/*********************************************************************/
   CP_Owned  Slot  1  LX6RES
   CP_Owned  Slot  2  LX6SPL
   CP_Owned  Slot  3  LX6PG1
   CP_Owned  Slot  4  LX6W01
   CP_Owned  Slot  5  LX6W02
   CP_Owned  Slot  6  LX6PG2
   CP_Owned  Slot  7  RESERVED
   CP_Owned  Slot  8  RESERVED
   CP_Owned  Slot  9  RESERVED
```

4. Pick the slot which is RESERVED to put this new spool disk. Substitute the word RESERVED with the name of the new disk SPL001, as shown here:

```
/*********************************************************************/
/*              CP_Owned Volume Statements                 */
/*********************************************************************/
   CP_Owned  Slot  1  LX6RES
   CP_Owned  Slot  2  LX6SPL
   CP_Owned  Slot  3  LX6PG1
   CP_Owned  Slot  4  LX6W01
```

```
CP_Owned   Slot  5  LX6W02
CP_Owned   Slot  6  LX6PG2
CP_Owned   Slot  7  SPL001
CP_Owned   Slot  8  RESERVED
CP_Owned   Slot  9  RESERVED
```

5. File.
6. Run the utility CPSYNTAX located in MAINT mdisk 193 to verify that there is no error in SYSTEM CONFIG file.
   - If there are no errors, then schedule a system IPL to pick up this update.
   - If there are errors, xedit the SYSTEM CONFIG file and correct the errors.

Next, you need to add a spool disk dynamically after the SYSTEM CONFIG file has been updated.

Add spool disk dynamically, after SYSTEM CONFIG is updated

This update will activate immediately, and the disk will be available for use. Proceed with this step after the SYSTEM CONFIG file is successfully updated.

1. Based on slot 7 being defined in SYSTEM CONFIG, issue the following command:

   ```
   define cpowned slot 7 SPL001 owned
   attach SPL001 system
   ```

2. Check your definition in the system by issuing the command:

   ```
   q cpowned
   ```

   The response should show SPL001 in slot 7 as defined.

3. Finally, verify that the spool disk was successfully added to the system.

   Issue the command **`q alloc spool`** to verify that the spool disk was added successfully.

9.15 Starting z/VM

Starting z/VM on the hardware requires you to have access to the Hardware Management Console (HMC). Figure 9-6 on page 311 shows an example of a Hardware Management Console screen that is similar to what you may see.

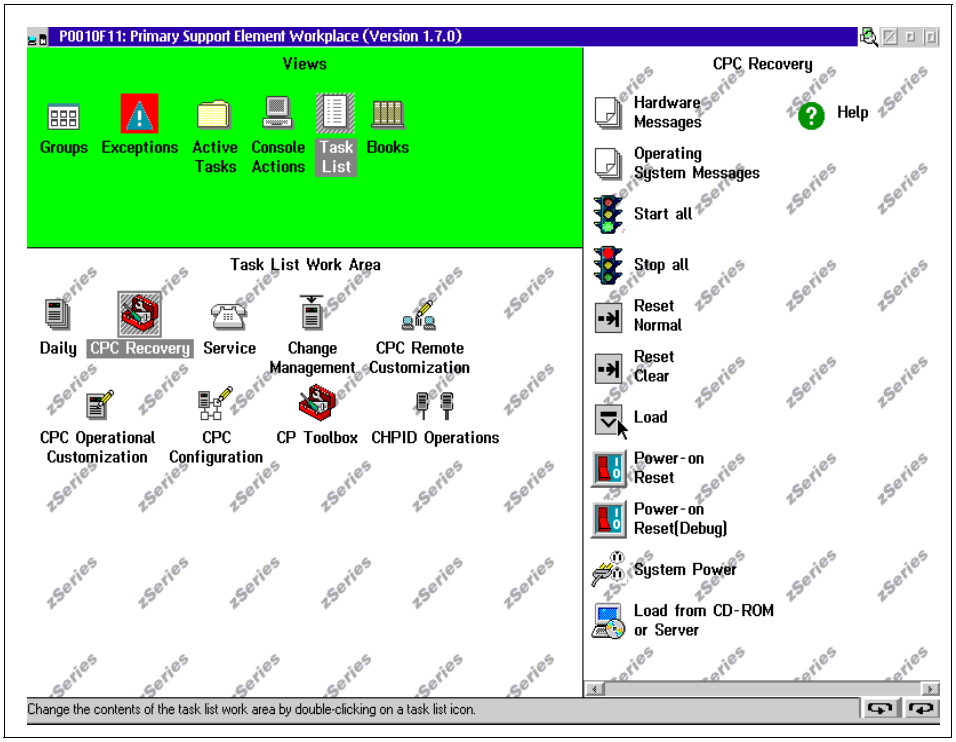


Figure 9-6 Hardware Management Console (HMC)

Where you go from here will probably be defined in your operations guidelines, but basically there two things to look at here. There will probably be profiles that define these functions:

Activation	This function will include a reload of the I/O configuration IOCDS into the LPAR, and then point to the Load profile to IPL.
Load	This function will load the operating system using a predefined device number and load parameters.

Using the mouse, you would drag the icon in the bottom left panel in Figure 9-6 that represented the LPAR that was to be loaded over the Activate or Load icon on the right panel.

If you choose to simply click the Load button, you will prompted to enter the relevant information manually; see Figure 9-7 on page 312.

Note: The Store Status is necessary if you are IPLing a standalone dump.

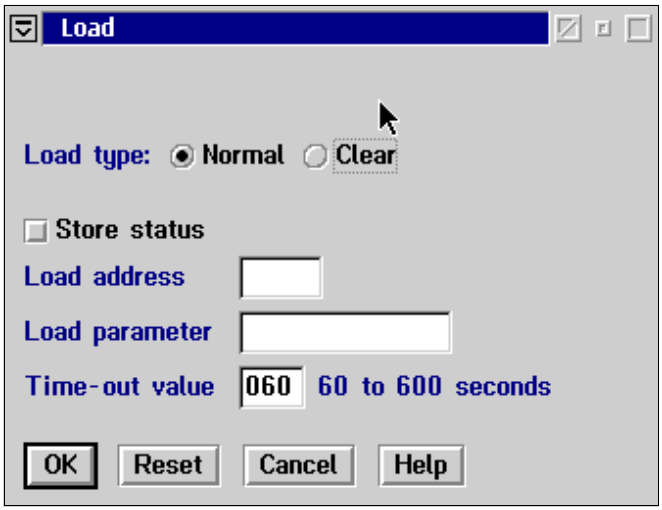


Figure 9-7 Load screen

If you specify a Load parameter in the Load screen, you will be presented with a screen similar to Figure 9-8.

```
STAND ALONE PROGRAM LOADER: z/VM VERSION 5 RELEASE 3.0

DEVICE NUMBER:   0123       MINIDISK OFFSET:   00000000   EXTENT:  1

MODULE NAME:     CPLOAD     LOAD ORIGIN:       1000

--------------------------------IPL PARAMETERS--------------------------------


-----------------------------------COMMENTS-----------------------------------


--------------------------------------------------------------------------------


9= FILELIST  10= LOAD  11= TOGGLE EXTENT/OFFSET
```

Figure 9-8 STAND ALONE PROGRAM LOADER (SAPL) screen

At this point, using the PF11 or PF9 keys, it is possible to change the IPL parameters that will be passed to z/VM. For example, you could change the CP minidisk that is searched for the load module, or select a different CPLOAD MODULE to use. You would then use PF10 to load the selected CP nucleus.

Note: It is possible that an automatic warm start will have been specified in the SYSTEM CONFIG file. In this case, you may not see the screen in Figure 9-9 on page 313 but instead, move on with the IPL. When the IPL is finished, it will disconnect the OPERATOR user ID.

If you do see the screen in Figure 9-9, then you must answer the prompt for which type of start that you would like. Normally you will specify Warm if you have performed a successful shutdown prior to this IPL. If the warm start fails, you will be prompted to try a Force start, which will try to rebuild the control structures that would have been saved by an orderly shutdown.

```
11:58:43 z/VM  V5 R3.0  SERVICE LEVEL 0701 (64-BIT)
11:58:43 SYSTEM NUCLEUS CREATED ON 2007-05-02 AT 16:28:04, LOADED FROM LX6RES
11:58:43
11:58:43 ************************************************************
11:58:43 * LICENSED MATERIALS - PROPERTY OF IBM*                      *
11:58:43 *                                                           *
11:58:43 * 5741-A05 (C) COPYRIGHT IBM CORP. 1983, 2007. ALL RIGHTS   *
11:58:43 * RESERVED. US GOVERNMENT USERS RESTRICTED RIGHTS - USE,     *
11:58:43 * DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE  *
11:58:43 * CONTRACT WITH IBM CORP.                                    *
11:58:43 *                                                           *
11:58:43 * * TRADEMARK OF INTERNATIONAL BUSINESS MACHINES.           *
11:58:43 ************************************************************
11:58:43
11:58:43 HCPZCO6718I Using parm disk 1 on volume LX6RES (device 0123).
11:58:43 HCPZCO6718I Parm disk resides on cylinders 39 through 158.
11:58:43 Start ((Warm|Force|COLD|CLEAN) (DRain) (DIsable)  (NODIRect)
11:58:43       (NOAUTOlog)) or (SHUTDOWN)
```

Figure 9-9 z/VM IPL

After all the prompts have been replied to, the IPL will continue to the point where you will see a message similar to Example 9-29.

Example 9-29 Messages

```
There is no logmsg data
FILES: NO RDR, NO PRT, NO PUN
LOGON AT 12:25:46 EDT WEDNESDAY 07/03/03
```

At this point, the IPL is complete and the z/VM logo should have appeared on attached local screens.

9.15.1 Shutting down z/VM

At some stage, it will be necessary to shut down z/VM. This could be for several reasons:

- A hardware upgrade
- A software upgrade (for instance, if maintenance has been applied to CP and it needs to be loaded)
- If performance analysis shows that there is a trend that may cause an unscheduled outage

To shut down z/VM, you need to use the SHUTDOWN command to:

- Systematically end all system function
- Checkpoint the system for an eventual warmstart

When CP is rebuilt, it is possible to save the new CPLOAD MODULE that is produced on the CP minidisk with a different name (for example, CPLODTST MODULE). If this has been done, SHUTDOWN can do an automatic warm start of the current or the new CP module when the REIPL parameter is used. This allows a rapid restart when loading a new version of CP that does not need you to access the HMC.

Part of the shutdown will signal other guests that might need an orderly shutdown and, if they support this signal, CP can be told to wait for a specified time to allow the guests' shutdown to complete before it shuts itself down. If no delay is required, you can use the IMMediate parameter to shut down without signalling the guests.

If REIPL is not specified, then when shutdown completes, the LPAR will load a wait state and a reIPL will be necessary from the HMC.

9.16 Basic automation

Whenever the system is IPLed and CP is loaded, part of the initialization routines look for a user ID called AUTOLOG1. This is a special user ID that is used to start basic automation of the z/VM system startup.

If it is found, CP will automatically log the user on, CMS will be automatically started in the virtual machine, and the PROFILE EXEC will be run. This profile can contain statements to start other user IDs and services, as well as to change aspects of the system. A sample EXEC might look like Example 9-30 on page 315.

Example 9-30 AUTOLOG1 PROFILE EXEC

```
/***************************/
/*  Autolog1 Profile Exec  */
/***************************/
/* System Monitoring commands */
/* Define the dcss MONDCSS used by the MONWRITE userid */
   'CP DEFSEG MONDCSS 2000-2BFF SC RSTD'
   'CP SAVESEG MONDCSS'
/* Enable the CP MONITOR EVENT                          */
   'CP MONITOR EVENT EN I/O  ALL'
   'CP MONITOR EVENT EN USER ALL'
   'CP MONITOR EVENT EN PROC    '
   'CP MONITOR EVENT EN STOR    '
Address Command
   'CP XAUTOLOG VMSERVS'
   ‘CP XAUTOLOG VMSERVU'
   'CP XAUTOLOG VMSERVR'
   'CP XAUTOLOG DTCVSW1'
   'CP XAUTOLOG DTCVSW2'
   'CP XAUTOLOG TCPIP'
   'CP XAUTOLOG DIRMAINT'
   'CP XAUTOLOG DATAMOVE'
   'CP LOGOFF'
```

Example 9-30 shows a simple profile that will define the monitor save segment and then issue some CP monitor commands. It then goes on to start three user IDs for the Shared File System (VMSERVx), two VSWITCH user IDs (DTCVSWx), TCP/IP and the two user IDs for DIRMAINT. These are performed using only CP commands in the EXEC.

We can expand the capabilities of the PROFILE EXEC by introducing some simple REXX. Using Example 9-30 as the starting point, we can add code to check whether the NSS MONDCSS has already been created and skip the definition if it has; see Example 9-31.

Example 9-31 Saving a DCSS

```
/* Define the dcss MONDCSS used by the MONWRITE userid */
'pipe cp q nss name mondcss|drop 1|specs 43-49|var havedcss'
if havedcss="MONDCSS"
then call monieven
else
   'CP DEFSEG MONDCSS 2000-2BFF SC RSTD'
   'CP SAVESEG MONDCSS'
/* Enable the CP MONITOR EVENT                          */
monieven:
   'CP MONITOR EVENT EN I/O  ALL'
```

```
   'CP MONITOR EVENT EN USER ALL'
   'CP MONITOR EVENT EN PROC    '
   'CP MONITOR EVENT EN STOR    '
```

In Example 9-31 on page 315, the pipe issues the CP QUERY command, drops the first record of the output, and then looks at columns 43-49 and puts this into a variable. This variable is then checked to see if says MONDCSS. If it is already there, skip the define and save and call the commands to enable the monitor.

You may also want to check the CPUID of the machine that you are running on and make decisions based on that information to decide which other virtual machines that you would like to autolog; see Example 9-32 for a sample exec.

Example 9-32 Example of automation exec

```
'pipe cp q cpuid|specs 9-*| var mycpu'
if mycpu="FF02991E20948000"
then
'CP XAUTOLOG MYSERVER'
else
'CP XAUTOLOG YOSERVER'
```

9.17 Advance messaging between users

The CP MSG command, as discussed in *9.2, "CP commands" on page 276*, allows you to exchange simple text messages with other users.

When you are running virtual machine servers it is sometimes necessary to send commands, in addition to messagesl. This is done using the CP SEND command.

However, this could be potentially dangerous because if the function is uncontrolled, you could encounter a situation where a class G user could send class A commands to a user that had that command class privilege.

To minimize the risk, the CP SEND command is controlled by using either the CONSOLE statement in the user's directory entry, or by using the CP **Set SECUSER** command. There are two versions of this command, one for class A and C users, and one for class G users. Example 9-33 is an example of the console directory entry.

Example 9-33 Console directory entry

```
CONSOLE 009 3215 T MAINT
```

Example 9-34 shows the syntax of the **Set SECUSER** command.

Example 9-34 SECUSER command

```
>>--Set--SECUSER--.-.----------.--OFF-.------------------------------>
                  | '-targetid-'      |
                  |-RESET-------------|
                  '-userid------------'
```

If you are a class G user, you may only specify a targetid if you are the SECUSER of that targetid and no longer want to be the SECUSER.

After you are authorized, you can send commands to the target virtual machine. An example of this is communicating with the CRR server VMSERVR.

In the directory entry for VMSERVR, MAINT is specified as a secondary user, and therefore it can issue commands to the server and receive output from it. In Example 9-35, note that the responses are prefixed with VMSERVR, which is the user ID of the virtual machine responding.

Example 9-35 SEND command

```
send vmservr crr query log
VMSERVR : Time: 13:11:46                    CRR QUERY LOG - VMSYSR
VMSERVR : Date: 06/04/07                    LUNAME - ZVMV5R30.RECOVER
VMSERVR :
VMSERVR :  Log Status:
VMSERVR :
VMSERVR :  Total number of Log minidisk 4K blocks:         352
VMSERVR :    Number of blocks for Log Ring:                240
VMSERVR :    Number of blocks for Logname Table:           112
VMSERVR :
VMSERVR :  Percent (%) of Log Ring space used:               0 %
VMSERVR :  Percent (%) of Logname Table space used:          0 %
VMSERVR :
VMSERVR :  Primary Log is Enabled
VMSERVR :  Secondary Log is Enabled
VMSERVR :
VMSERVR : DMS5BC3065I Operator command processing complete
```

There is another means of communication between virtual machines and that is by special messaging (SMSG). This allows a user to send messages to applications running in other virtual machines that have been programmed to accept the messages.

9.18 Installing and servicing the z/VM system

In this industry, things change! New technologies push the boundaries of what can be done and it is sometimes necessary to install new operating system software to take advantage of these technologies. In this section, we provide an overview of installing and servicing the z/VM system.

9.18.1 Installing

Installing your system always starts with ordering your new software. This will normally be ordered from IBM using a VM System Delivery Option (SDO) which contains a VM System DDR, a Recommended Service Upgrade (RSU), and any optional products that you may need. After you place your order, the z/VM SDO is shipped either on tape cartridge or on DVD.

If you are already running z/VM, you will probably install the new system in a virtual machine. If you are not already running z/VM, you will need to IPL the LPAR from either the cartridge or the DVD, as described in 9.15, “Starting z/VM” on page 310.

The basic installation process is very easy. For detailed information about the basic installation process, refer to *z/VM V5R2.0 Guide for Automated Installation and Service*, GC24-6099. Also consult *z/VM V5R2.0 Summary for Automated Installation and Service*, GA76-0406 (DVD) and GA76-0407 (tape).

The summary is provided in a form that can be used as a checklist as you progress through the installation and, if followed exactly, step by step, will enable you to install and IPL a z/VM system very easily.

Figure 9-10 on page 319 shows the basic steps necessary to get your system up and running.

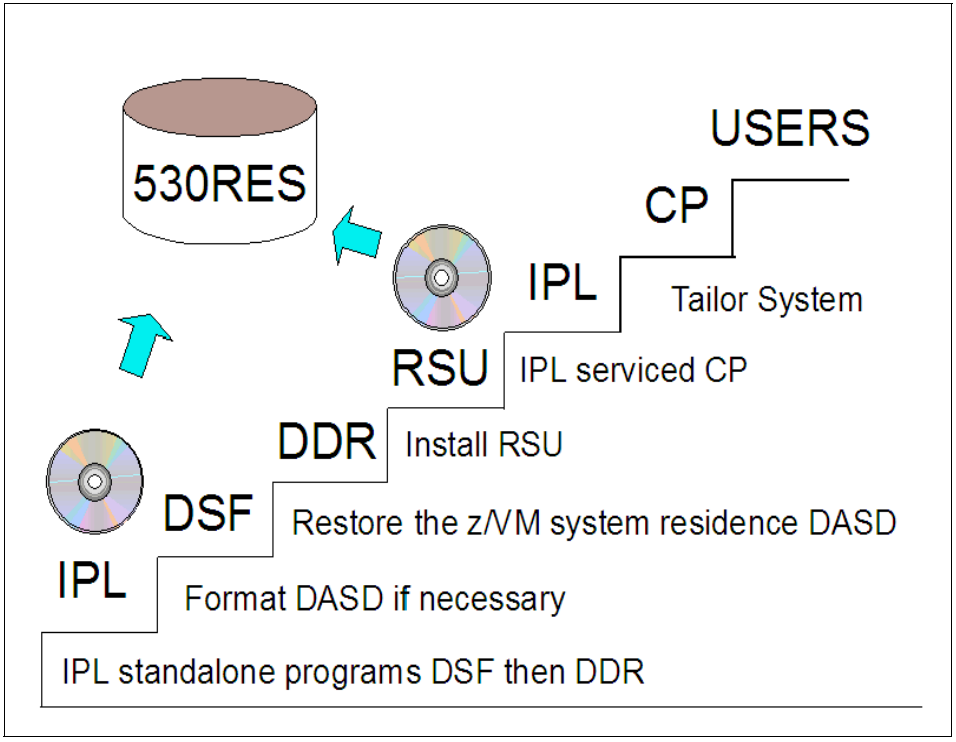


Figure 9-10 Installation steps - overview

Servicing

There may be a need to apply corrective service to your system for two main reasons: to correct a defect with the z/VM supplied code, or to add new function to z/VM. The service can be requested on various tape formats, or electronically as service envelopes. The service is applied most easily using the SERVICE exe, and is put in to production using the PUT2PROD exec.

For detailed information about servicing z/VM, refer to *z/VM Guide for Automated Installation and Planning*, GC24-6099*; z/VM Service Guide*, GC24-6117; and *VMSES/E Introduction and Reference*, GC24-6130.

9.19 Exercises

To help test your understanding of the material in this chapter, answer the following questions:

1. List at least three privileged CP commands.
2. Which CP utility is used to format and allocate DASDs?
3. List the Named Saved Segment (NSS) used by CMS users.
4. Which z/VM system component accesses the PARM disk?
5. What is the file name where you define your resource addresses, DASD lists and CP-owned disks?
6. If you want to back up your system, which programs you have to use in order to back up DASDs and SPOOL files?
7. Which command (or commands) do you have to issue in order to access the PARM disks?
8. List the major steps needed to apply a service using the VMSES/E commands.
9. Which command you execute in order to access the VMSES/E disks to CMS component?

10

Performance

In this chapter we introduce z/VM performance and capacity management.

Objectives

After completing this chapter, you will be able to:

- Describe what is meant by performance
- Understand the basics of z/VM scheduling and dispatching
- Describe the CP commands useful in performance monitoring
- Discuss what data should be collected
- Describe what to look for when determining performance problems
- Discuss some of the tools available for performance data collection

© Copyright IBM Corp. 2007. All rights reserved.

10.1 z/VM performance

Before we examine performance management, it is useful to consider what is meant by the term “performance”.

10.1.1 What is performance

There are many aspects to be considered when looking at the performance of a z/VM system, including the following areas:

- User response time
- Throughput
- Device utilization
- Number of users supported
- Reliability
- System capacity

Any one of these aspects can be affected by the resources that are available on the system; therefore, you must also take these resources into consideration:

- Applications

  Do the applications consist of simple execs, or a gigantic database?

- Storage

  Was the hardware sized correctly? Are there many guests with huge virtual machines?

- CPU

  Was the hardware sized correctly for the workload? Is there a CPU “hog”?

- I/O

  Is the I/O configured correctly? Do many users need the same device?

- Networking

  Are there enough ports enabled? Could there be a denial-of-service attack?

- Paging

  Is there enough paging space? Are some users causing too much paging load?

10.2 Recognizing a performance problem

If you were the administrator of a z/VM system and the system developed a performance problem, how would you know? Often your first clue will be that users contact you with the following complaints:

- My terminal is not responding.
- My job has not finished.
- My Linux users cannot log in.
- The Web server is not responding.
- The batch workload on my guest is taking too long.
- The system is dead.

It is at this point that the knowledge that you gain from this chapter will help you to understand the problem and, if possible, correct it.

Note: Always keep in mind, however, that you may not be able to fix everything if you have insufficient resources available. So you will encounter situations when the only answer will be to upgrade your processor, storage, or other hardware to match demand or to reduce the load on your system.

10.3 CP scheduling and dispatching

Before looking at the commands and facilities used to monitor and tune a z/VM system, however, we first explain how CP shares resources among users; refer to Figure 10-1 on page 324.

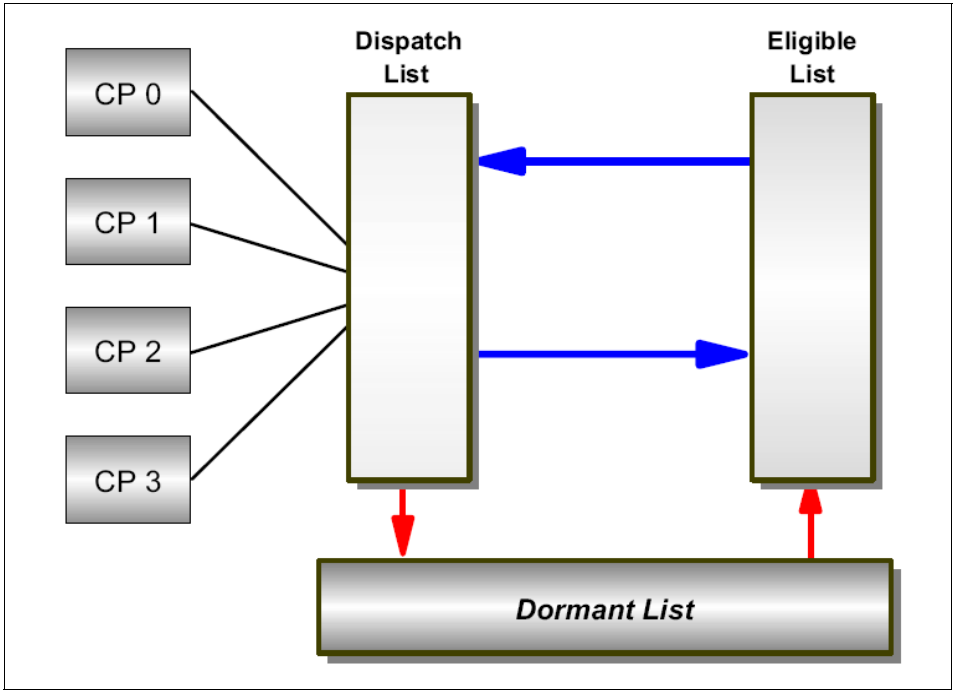


Figure 10-1 z/VM Scheduling overview

When you log on to z/VM, CP creates a control block[1] that is the anchor for a large amount of information about your virtual machine. This block is called a Virtual Machine Descriptor Block (VMDBK). It is pointers to this block that get moved around the lists that you can see in Figure 10-1.

If your virtual machine is not doing anything, then CP will place it in the dormant list, which implies that your virtual machine is “asleep”. If you “wake up” (for instance, by pressing the Enter key on your console), it will get moved to the eligible list where it is eligible to receive some system resource.

Before it is given any resource, however, the virtual machine is examined by the scheduler. The scheduler looks in the VMDBK of your virtual machine to see what resources you have used in the past. Processor, storage, and paging are some of the resources examined.

If you have just logged on, there will not be much information in the VMDBK, so the scheduler thinks of you as an *interactive user*, also known as a class 1 user.

[1] There are many control blocks used in z/VM. Control blocks are areas of storage that contain information pertinent to the operation that they are associated with.

A class 1 user is normally a CMS interactive user or a very lightly loaded guest operating system.

When the scheduler determines that there is enough resource available to satisfy your needs without putting any other users in jeopardy, you will be moved to the dispatch list where you wait in turn for your allowance of CPU time (time slice). When your turn arrives, your virtual machine is run on an available processor.

The time slice comes to an end after a certain time known as the *dispatcher minor timeslice*. If you have more work to do, you can stay in the dispatch list and get another time slice until either you have finished (in which case you will be moved to the dormant list) or you will have to pay a visit to the scheduler again so that it can reassess your resource requirements before rescheduling you.

The scheduler may decide, after you have visited the dispatch list a few times, that you are not really an interactive user but that you really do need some more processing capacity. You will then be upgraded to be a class 2 user and allowed to stay in the dispatch list for longer intervals to see if you then manage to finish your work.

This has an additional benefit because it reduces the amount of policy decisions that the scheduler has to make to manage eligible users. A class 2 user might typically be a CMS user compiling programs, or a Linux Web server.

After you have visited the dispatch list as a class 2 user several times and still not finished your work, the scheduler will upgrade you to a class 3 user and you will be allowed to stay in the dispatch list even longer to see if you manage to finish your work. Normally, a class 3 user will be a production guest operating system.

There are drawbacks to being a higher class user. For instance, if the system resources (such as paging or storage) become constrained, then the scheduler will hold higher class users in the eligible list and let lower class users run.

However, there is one type of user, known as a class 0 user (it has OPTION QUICKDSP in its directory entry or has had SET QUICKDSP issued on its behalf) who will never be held in the eligible list. Typically, this user type is only used for important servers and special user IDs.

Table 10-1 lists and explains the various user classes.

Table 10-1 User classes

User class	Explanation
Class 0	This class indicates the users that were added to the dispatch list with no delay in the eligible list, regardless of the length of their current transaction.

User class	Explanation
Class 1	This class indicates the users that have just begun a transaction, and therefore are assumed to be currently processing short transactions.
Class 2	This class indicates the users that did not complete their current transactions during their first dispatch list stay and therefore are assumed to be running medium-length transactions.
Class 3	This class indicates the users that did not complete their current transactions during their second dispatch stay and therefore are assumed to be running long transactions.

If you have command privilege class E, you can issue the CP INDICATE LOAD command to view information about these classes of user; see Figure 10-2.

Qn indicates users in the dispatch list. En indicates users in the eligible list where n is the class of the user 0, 1, 2, or 3.

```
ind
AVGPROC-000% 04
XSTORE-000000/SEC MIGRATE-0000/SEC
MDC READS-000001/SEC WRITES-000000/SEC HIT RATIO-084%
PAGING-0/SEC STEAL-000%
Q0-00000(00000)                          DORMANT-00016
Q1-00000(00000)           E1-00000(00000)
Q2-00000(00000) EXPAN-001 E2-00000(00000)
Q3-00000(00000) EXPAN-001 E3-00000(00000)

PROC 0000-000% CP       PROC 0001-000% CP
PROC 0002-000% CP       PROC 0003-000% CP

LIMITED-00000
```

Figure 10-2 INDICATE LOAD command

Note: Any count in the En fields normally indicates a performance problem (probably a storage or paging constraint).

10.4 Performance monitoring

Performance management in a z/VM system needs tools to collect and analyze data, and to control resource usage by virtual machines. These tools are provided in CP by commands and monitoring functions.

- CP INDICATE
- CP QUERY
- CP MONITOR
- CP SET

Additionally, there are other products from IBM and vendors that can be used to collect, analyze and present data in various display, report and graphical formats. These products include:

- Performance Toolkit, shipped with the z/VM base
- Omegamon for z/VM
- Vendor products

10.4.1 CP commands

Several CP commands are useful when you are gathering performance data; the most important one is the CP INDICATE command.

For more information about the commands listed in this section, refer to *CP Commands and Utilities Reference*, SC24-6081.

INDICATE

The CP INDICATE command gives a snapshot of resource utilization. Because it is only a snapshot, however, commands need to be issued several times and the results examined for changes.

Some of the commands have an EXPanded option, which will give additional detail. The INDICATE command can be used by system resource operators, system programmers, system analysts, and general users (class B, C, E, and G users) although different classes of user will see different results to some commands.

Table 10-2 on page 328 lists and describes the CP INDICATE parameters.

Table 10-2 CP INDICATE parameters

Parameter	Description
ACTIVE	Use INDICATE ACTIVE to display: ► The total number of users active in a specified time interval. ► The number of users in the dispatch, eligible, and dormant lists that were active in a specified time interval.
IO	Use the INDICATE IO command to identify the virtual machines currently in an I/O wait state and the real I/O device number that they are waiting on.
LOAD	Use the INDICATE LOAD command to display information about system resources. For a general user, CP displays a subset of the information. Note that the processor usage information shown by this command is actually a “smoothed” average over a period of time. Changes in system load will likely take a few minutes to influence this number. For this reason, the Performance Tool Kit is often used to monitor instantaneous processor load. *This is probably the most useful command to start with when analyzing system performance problems.*
NSS	Use the INDICATE NSS command display information on named saved systems (NSS) and saved segments that are loaded in the system and are in use by at least one user.
PAGING	Use INDICATE PAGING to display: ► A list of the virtual machines in page wait status. ► Page residency data for all system users
QUEUES	Use INDICATE QUEUES to display, in order of their priority, current members of the dispatch and eligible lists. If users have a virtual multiprocessor, you may see more than one entry for a single user.
SPACES	Use INDICATE SPACES to display information about a user’s address space and paging usage.
USER	Use INDICATE USER to display the resources used or occupied by a virtual machine or by the system.General users can enter the INDICATE USER command to obtain resource usage statistics about their own virtual machines. A system analyst can enter the INDICATE USER command to obtain these statistics for any virtual machine. CP displays a set of statistics for each of the virtual machine's virtual processors. The EXPanded option will show more detail.

For further information about syntax and details about these commands, see the z/VM HELP information or *z/VM CP Command and Utility Reference.*

CP QUERY commands

The Query commands can be used to display a great deal of information about the system and users. Table 10-3 lists and describes some query commands that may be useful when investigating performance problems.

Table 10-3 CP QUERY commands

Query	Description
ALLOC	Use QUERY ALLOC to display the number of cylinders or pages that are allocated, in use, and available for DASD volumes attached to the system. Use Q ALLOC PAGE for detailed information about paging space.
CACHE	Use QUERY CACHE to display caching status for all storage subsystems that support caching when investigating I/O-related problems.
CHPIDS	Use QUERY CHPIDS to display all 256 of the machine's channel paths and their physical status. I/O performance problems can occur if CHPIDs are offline or not available.
CPLOAD	Use QUERY CPLOAD to display information regarding the last CP IPL. The information displayed includes the location of the CP module that was last used, the location of the parm disk, and how CP was started.
CPOWNED	Use QUERY CPOWNED to display the list of CP-owned DASD volumes. If paging or spooling problems occur, this can be checked to ensure that all required volumes are available.
FRAMES	Use QUERY FRAMES to display the status of host real storage. It may be necessary to do this if you are getting users in the eligible list when you need to check storage utilization.
MAXUSERS	Use QUERY MAXUSERS to display the maximum number of logged-on users allowed. If users cannot logon, this may be a reason.
MDC	Use QUERY MDC from a Class B user to query: ► Minidisk cache settings for the entire system, for a real device, an active minidisk, or a minidisk defined in the directory. ► A user's ability to insert data into the cache. This command is useful for investigating I/O problems.
NAMES	Use QUERY NAMES to display: ► A list of all logged-on users. ► The real or logical device number of the display to which each user is connected.

Query	Description
PATHS	Use QUERY PATHS to display: ► All paths installed to a specific device or range of devices ► Installed path status Use this command when investigating I/O problems.
PENDING	Use the QUERY PENDING command to display the device commands that you have entered and, optionally, that others have entered for which the associated asynchronous function has not yet completed. Use this command when investigating stopped users or I/O problems.
QIOASSIST	Use QUERY QIOASSIST to determine the current status of the queue-I/O assist for a virtual machine. Use this command when investigating I/O or networking problems.
QUICKDSP	Use QUERY QUICKDSP to display the QUICKDSP attribute for a user.
RESERVED	Use QUERY RESERVED to display the number of reserved real storage frames. Use SET RESERVED to reserve pages of storage for a user. Use this command when tuning users in a storage-constrained system.
SRM	Use QUERY SRM (system resource manager) to display system-wide parameters use by the scheduler to set the priority of system resource access. The CP SET SRM command is useful if you need to control a user's use of resources, depending on the class of user. Use this command carefully and monitor usage because eligible lists can occur if not used properly.
STOR	Use QUERY STORAGE or QUERY STORE to display the size of real storage.
SYSTEM	Use QUERY SYSTEM to display current user access to a system DASD volume.
XSTOR	Use QUERY XSTORAGE or QUERY XSTORE to display the assignment of real Expanded Storage. Use this command to investigate response time or paging problems.

For further information about the details and syntax of these commands, refer to the z/VM HELP information or *z/VM CP Command and Utilities Reference.*

CP SET commands

There are several CP SET commands that can change the performance characteristics of the entire system or of a single user, as listed and described in Table 10-4.

Table 10-4 CP SET commands

Set	Description
CACHE	Use SET CACHE to activate or deactivate the cache function by device or by subsystem. This command may be useful when you investigate I/O problems.
MAXUSERS	Use SET MAXUSERS to control the number of users able to log on. If the CPU is constrained, you could use this to limit the number of users.
MDC	Use the SET MDC command from a class B user to: ► Change minidisk cache settings for the entire system, for a real device, or for an active minidisk. ► Purge the cache of data from a real device or an active minidisk. ► Change a user's ability to insert data into the cache. This command can be useful if you do not have much minidisk activity and you want to release some storage.
QIOASSIST	Use SET QIOASSIST to control the queue-I/O assist (QDIO performance assist for V=V guests) for a virtual machine. This interpretive-execution assist applies to devices that use the Queued Direct I/O (QDIO) architecture, HiperSockets devices, and FCP devices.
QUICKDSP	Use SET QUICKDSP to assign or unassign a user's immediate access to system resources.
RESERVED	Use SET RESERVED to establish the number of real storage frames that are available to a specific virtual machine. This command might be useful when you are trying to tune specific guests in a storage-constrained environment.
SHARE	Use SET SHARE to change the system-resource-access priority for users. This is a complex command that can have a profound effect on system and user resource usage, so use with care.
SRM	Use SET SRM (system resource manager) to change system parameters. These parameters define the size of the time slice, the access to resources for different user classes as seen by the scheduler. This is a complex command and you should clearly understand its effects before using it.

Set	Description
THROTTLE	Use SET THROTTLE to control the number of I/O operations that a guest operating system can initiate to a specific real device. This prevents a guest from interfering with, or dominating, I/O resources.

User directory commands

The CP SET commands that can be used to allocate shares of resources to users are discussed in Table 10-4 on page 331.

There are equivalent statements, and other statements, that can go into the definition of the user in the user directory to change the performance characteristics. Some of these statements are listed and described in Table 10-5.

Table 10-5 User directory commands

Statement	Description
DEDICATE	Use DEDICATE to allow guest exclusive use of a real device. This command does the same as the ATTACH command.
IOPRIORITY	Use IOPRIORITY if the LPAR supports it to control a guest's I/O priority queueing range.
MINIOPT	Use MINIOPT to control caching and minidisk caching for a guest. This is useful if a guest minidisk gets written to frequently (for example, a log or swap disk, in which case you should use the NOMDC option to turn off caching for this minidisk).
OPTION	Use OPTION to set some performance criteria for the guest. Examples include: ► QUICKDSP ► NOMDCFS
SHARE	Use the SHARE statement to specify a virtual machine's share of CPU power.
STORAGE	Use STORAGE to determine the size of the virtual machine's memory.

CP Monitor

The CP Monitor facility records system performance data into an area of shareable storage known as a Discontiguous Saved Segment (DCSS). With the base z/VM installed, there is one predefined segment called MONDCSS which users who have an IUCV *MONITOR statement in their user directory entry can access.

This data can be processed by an external data reduction program to produce statistics to give you an understanding of system operation or help you analyze the use of, and contention for, major system resources.

Monitored resources include processors, storage, input/output devices, and the paging subsystem. You can control the amount of data and the type of data collected.

In general, monitoring is done in this order:

1. We use the CP privileged command, **MONITOR**, to control monitoring, including the type, amount, and nature of data to be collected.
2. The monitor collects performance data during CP operation and stores it, in the form of monitor records, in a saved segment.
3. An application program running in a CMS virtual machine connects to the CP *MONITOR System Service.
4. The application program retrieves monitor records from the saved segment, processes them, and saves them on disk or tape.

A program supplied in z/VM, called MONWRITE, can be used as the application program to retrieve monitor records. MONWRITE not only retrieves monitor records from the saved segment, but also stores them on tape or in a CMS file on disk. Another application program can then read the records from the file and perform data reduction on the performance data found in the records.

Types of monitor data

The monitor data can be of two types, event data and sample data, as explained here.

Event data

Event data is collected and reported each time a designated system event occurs. The reported data represents system status at the time the event occurs. For example, to monitor the DASD device at address 1234, use the MONITOR ENABLE EVENT I/O DEVICE 1234 command. Events recorded for the device include VARY ON, VARY OFF, ATTACH, and DETACH.

Sample data

Sample data is reported at the end of a designated time interval. Two varieties of sample data are collected: single-sample data, and high-frequency sample data.

- Single-sample data

  Single-sample data is collected and reported once, at the end of the time interval. Some of the data represents system status at the time of collection.

Other data represents accumulated counters, states, or elapsed times since the start of sampling.

- High-frequency sample data

  High-frequency data is collected more frequently than reported. At each high-frequency sampling time, the collected data is added to the corresponding counters. The data is reported once at the end of the time interval (along with single-sample data), and it represents the accumulated counter or state values since the start of high-frequency sampling.

The event data and sample data are subdivided into domains that describe the area of the system from which the data was collected, as listed here:

- MONITOR
- PROCESSOR
- STORAGE
- SCHEDULER
- SEEKS
- USER
- I/O
- NETWORK
- APPLDATA

Certain application programs are able to write monitor records. TCP/IP is an example of an application that produces these records.

10.4.2 Monitor data collection

To enable the collection of monitor data on your system, follow these steps:

1. Create a DCSS.

   If a DCSS does not already exist, you will need to create one. Issue the command **CP QUERY NSS NAME MONDCSS** to check for the existence of a DCSS for monitor. Example 10-1 illustrates a DCSS segment already defined.

Example 10-1 Query for DCSS

```
CP QUERY NSS NAME MONDCSS
OWNERID  FILE TYPE CL RECS DATE  TIME     FILENAME FILETYPE ORIGINID
*NSS     0071 NSS  R  0001 05/03 12:27:51 MONDCSS  DCSS     AUTOLOG1
Ready; T=0.01/0.01 14:11:32
```

2. If DCSS is not defined, then you can use the CP privileged commands shown in Example 10-2 on page 335 to create a DCSS slightly larger than 10 megabytes in size, starting at the 35 megabyte address. This is big enough

for typical systems. Ensure that the defined segment does not overlap other segments in the user ID.

Note: A default MONDCSS definition is provided with your z/VM system.

Example 10-2 Define DCSS

```
CP DEFSEG MONDCSS 2300-2FFF SC RSTD
CP SAVESEG MONDCSS
```

3. If you do not currently run monitor, you need to set up a user ID to run the MONWRITE utility. The statements in Example 10-3 must be in the user entry to allow the user to access the monitor records.

   default MONWRITE user ID is provided in the USER DIRECT file on the MAINT 2CC minidisk provided with your z/VM system.

Example 10-3 Defining USERID

```
MACH XA
NAMESAVE MONDCSS
IUCV *MONITOR MSGLIMIT 255
```

4. Enable or set the data to be collected using monitor. In Example 10-4, we are interested in the following domains of Storage, Processor, I/O and all APPLDATA.

Example 10-4 Enabling domains

```
CP MONITOR SAMPLE ENABLE ALL
CP MONITOR EVENT ENABLE STORAGE
CP MONITOR EVENT ENABLE PROCESSOR
CP MONITOR EVENT ENABLE I/O ALL
CP MONITOR EVENT ENABLE APPLDATA ALL
```

5. Issue the command shown in Example 10-5 from a privileged ID to start monitor.

Example 10-5 Starting monitor

```
CP MONITOR START
```

6. Start MONWRITE. From the user ID generated, issue the command shown in Example 10-6 on page 336 to collect data to a file named MONITOR DATA B.

Example 10-6 Command to collect data to a CMS file

```
MONWRITE MONDCSS *MONITOR DISK filename filetype filemode
```

7. From the MONWRITE ID, stop MONWRITE and monitor; see Example 10-7.

Example 10-7 Stopping MONWRITE and monitor

```
MONWSTOP
CP MONITOR STOP
```

The MONVIEW tool enables you to view raw monitor data. The tool is available on the downloads page at the z/VM Web site:

http://www.vm.ibm.com/download/packages

10.4.3 NSS and DCSS

When running guests that have the same function, z/VM has facilities to allow groups of users to share applications, data and operating systems. The shared data and code is stored by CP in the dynamic paging area and is accessed by users as Named Saved Systems (NSS) and Discontiguous Shared Segments (DCSS).

The difference between these two facilities is that an NSS can be IPLed (for example, IPL CMS), and the DCSSs are linked to by applications that see them as existing in the virtual machine. They both are accessed by the guest as part of their virtual storage, and therefore appear transparent to whatever is running in the guest.

From a performance perspective, this arrangement offers the following benefits:

- It substantially reduces the amount of real storage that is needed.
- It provides better performance for a guest.

If viewed from a Linux perspective, having a large part of the kernel resident in storage would speed up the boot of the operating system significantly. A guest would boot a NSS called, for example, IPL LNXTST, instead of using a virtual device number such as IPL 580.

The most commonly used NSS in z/VM is CMS. If you issue the command Q NSS ALL from a privileged user, you will also see many other functions such as HELP, CMS Pipeline and NLS (National Language Support), which are DCSSs, benefit using this support.

For more information about DCSS, refer to *z/VM Saved Segments Planning and Administration*, SC24-6116. For more information about NSS, refer to *z/VM Virtual Machine Operation*, SC24-6128. If using Linux, refer to *Linux on System z Device Driver, Features, and Commands*, SC33-8289, for more details on NSS and DCSS.

10.5 Performance Toolkit

The Performance Toolkit for VM is an enhanced real-time performance monitor and fullscreen operator that allows system programmers to monitor system performance and to analyze bottlenecks. The toolkit can help system programmers to make more efficient use of system resources, increase system productivity, and improve user satisfaction.

In addition to analyzing z/VM performance data, the Performance Toolkit for z/VM can process Linux performance data obtained from the IBM Resource Management Facility (RMF) Linux performance gatherer, rmfpms. Some of the other functions provided by the Performance Toolkit for z/VM include:

- Operation of the system operator console in full-screen mode
- Management of multiple z/VM systems (local or remote)
- Post-processing of performance toolkit for VM monitor data captured by the MONWRITE utility
- Viewing of performance monitor data using either Web browsers or PC-based 3270 emulator graphics
- TCP/IP performance reporting

Monitoring resources

The following resources are examples of what can be monitored using Performance Toolkit.

- General CPU performance
- System and user storage utilization and management
- Channel and I/O device performance, including cache and SEEKs analysis data
- Detailed I/O device performance, including information about the I/O load caused by specific minidisks on a real disk pack
- General user data: resource consumption, paging information, IUCV and VMCF communications, wait states, response times
- Detailed user performance, including status and load of virtual devices
- Summary and detailed information about Shared File System servers
- Configuration and performance information for TCP/IP servers
- Linux performance data

10.5.1 Modes of operations

You can access the performance monitor by using several methods:

- From the command line of the PERFSVM operator console by typing `MONITOR`; see Figure 10-3.

```
FCX124          Performance Screen Selection  (FL530        )    Perf. Monitor

General System Data        I/O Data                  History Data (by Time)
1. CPU load and trans.     11. Channel load          31. Graphics selection
2. Storage utilization     12. Control units         32. History data files*
3. Reserved                13. I/O device load*      33. Benchmark displays*
4. Priv. operations        14. CP owned disks*       34. Correlation coeff.
5. System counters         15. Cache extend. func.*  35. System summary*
6. CP IUCV services        16. DASD I/O assist       36. Auxiliary storage
7. SPOOL file display*     17. DASD seek distance*   37. CP communications*
8. LPAR data               18. I/O prior. queueing*  38. DASD load
9. Shared segments         19. I/O configuration     39. Minidisk cache*
A. Shared data spaces      1A. I/O config. changes   3A. Storage mgmt. data*
B. Virt. disks in stor.                              3B. Proc. load & config*
C. Transact. statistics    User Data                 3C. Logical part. load
D. Monitor data            21. User resource usage*  3D. Response time (all)*
E. Monitor settings        22. User paging load*     3E. RSK data menu*
F. System settings         23. User wait states*     3F. Scheduler queues
G. System configuration    24. User response time*   3G. Scheduler data
H. VM Resource Manager     25. Resources/transact.*  3H. SFS/BFS logs menu*
                           26. User communication*   3I. System log
I. Exceptions              27. Multitasking users*   3K. TCP/IP data menu*
                           28. User configuration*   3L. User communication
K. User defined data*      29. Linux systems*        3M. User wait states

            Pointers to related or more detailed performance data
            can be found on displays marked with an asterisk (*).


Select performance screen with cursor and hit ENTER
Command ===> _____________________________________________________________________
F1=Help  F4=Top  F5=Bot  F7=Bkwd  F8=Fwd  F12=Return
```

Figure 10-3 3270 PTK menu

- From a Web browser; see Figure 10-4 on page 339.
- From a different user if the VMCX MODULE is available. Access is more limited using this method.

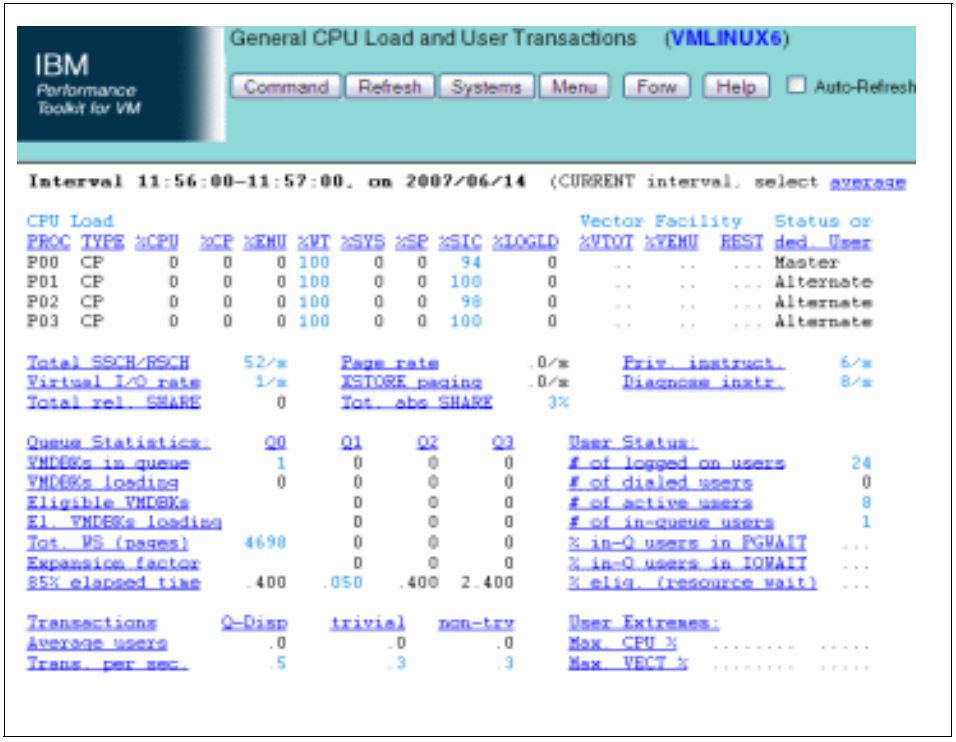


Figure 10-4 PTK access from browser

10.6 Tivoli Omegamon for z/VM and Linux

OMEGAMON® is a real-time software performance monitor for the VM (Virtual Machine) operating system. It runs under the Conversational Monitor System (CMS) operating system. OMEGAMON warns you of exceptional conditions automatically, and also displays the status of VM internal operations and resources in real time.

All of the OMEGAMON features and facilities are designed around the concept of a logical tuning approach for improving the performance of your system. The logical tuning approach consists of these steps:

- Defining standards for VM performance at your installation.
- Monitoring your system to measure actual performance against these standards.
- Identifying the cause of performance problems.
- Initiating action to correct performance problems.

Exception Analysis feature

The OMEGAMON Exception Analysis feature displays messages that warn of existing or impending hardware and software problems from both system-wide and individual VM user perspectives. OMEGAMON triggers exception messages when system conditions do not comply with the service levels your installation has set as exception thresholds.

Impact and Bottleneck Analysis features

OMEGAMON Impact Analysis is a unique feature that uses degradation data to identify which virtual machines are competing for the same resources. It allows you to quickly analyze workload contention so that you can take immediate action to solve performance problems.

The Bottleneck Analysis feature uses the same data to analyze response time by pinpointing the causes of delays, such as CPU waits, I/O waits, and paging delays.

10.6.1 Performance monitoring

Through the prerequisite IBM Performance Toolkit for z/VM, Tivoli OMEGAMON XE on z/VM and Linux provides detailed performance metrics for your Linux applications and the underlying z/VM virtual machine operating system that assigns CPU, storage and other computing resources to those applications. The product reveals statistics for all Linux processes that run:

- CP control blocks
- z/VSE partitions in VM
- z/VM user s CPU utilization, storage activity, and I/O
- Main storage and DPA utilization
- Paging and SPOOLing
- z/VM trace table
- Critical system resource consumption
- Minidisk and T-disk utilization

10.6.2 Tivoli OMEGAMON workspaces

With OMEGAMON XE on z/VM and Linux workspaces, shown in Figure 10-5 on page 341, various resource and performance statistics (which include terminal response, virtual machine throughput and resource activity) can be monitored easily.

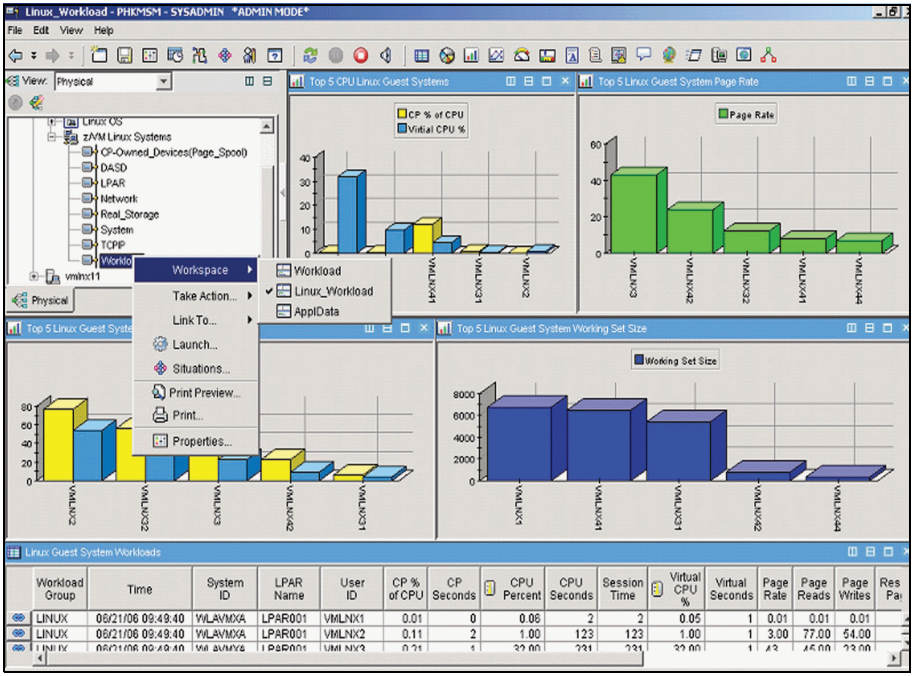


Figure 10-5 OMEGAMON XE on z/VM and Linux workspaces

In addition, operators can observe workload details for virtual machines, groups, response times and logical partitions (LPARs) and compare those statistics against baseline thresholds, in order to quickly isolate and pinpoint problems.

The OMEGAMON workspaces help you to easily investigate problems related to z/VM and Linux performance, with views of metrics including CPU consumption (guest/virtual and z/VM overhead), paging rates, total page reads (and writes) and working set size. Using Dynamic Workspace Linking, you can make OMEGAMON understand the intricate relationship between monitored data and subsystems such as IBM IMS™, IBM DB2, storage and others. And Dynamic Workspace Linking automatically provides links to the most relevant workspaces, thus freeing users from the need to comprehend those relationships.

For added control, you can create alerts based on any attribute monitored by Tivoli OMEGAMON XE on z/VM and Linux, from the details of Linux file systems to those of z/VM minidisks.

10.7 Analyzing your data

In 10.4, “Performance monitoring” on page 327, we discuss the tools that can be used to collect performance-related data. Here, we explain how to analyze the data that is gathered in order to identify where the constraint that is causing the problem occurred. But first we explain what we mean by a “constrained system”.

In a fairly simple model of z/VM, the main resources can be summarized as CPU, Storage, Paging and I/O. In Figure 10-6, each of these resources is represented by a pipe of a given diameter, which represents how much of that resource we have available. The workload is represented by the size of the arrow on the left side.

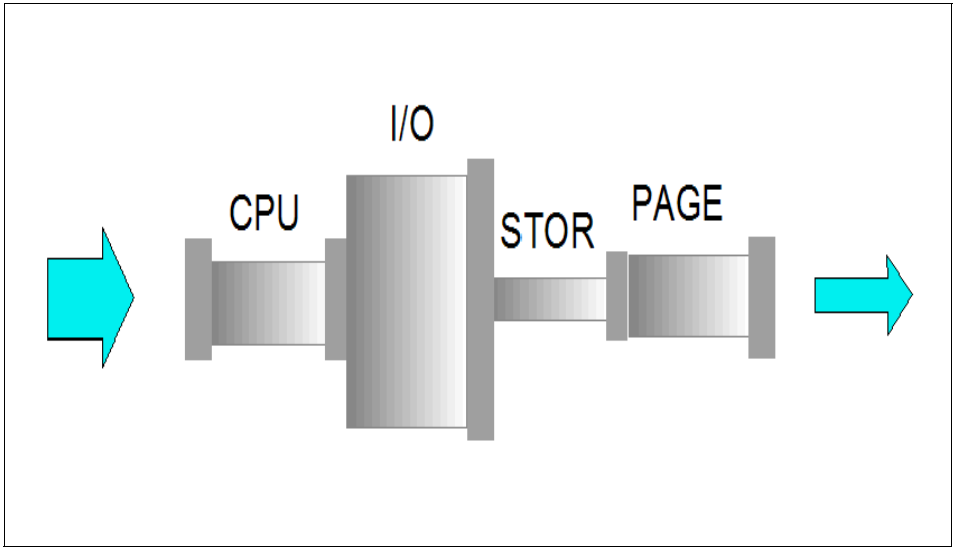


Figure 10-6 Constraint example

In this figure, you can see that the constraint in this system is storage and that is where you should direct our efforts when trying to achieve the workload that you want to have. It is also clear in the figure that if the storage constraint is relieved then, if the workload is to increase, we may need to take action prior to this increase to upgrade the capacity of the system. Skill in performance and capacity management is required to identify these constraints (or “bottlenecks”, as they are sometimes called).

Another thing to consider when analyzing the data is “the law of diminishing returns”. Your greatest performance benefits usually come from your initial efforts. Further changes generally produce smaller and smaller benefits and require more and more effort.

Also be aware that no amount of analysis and tuning can make up for a real shortage of a given resource; you really may need to add more storage or another processor.

It is not always easy to recognize a performance problem if you do not have a starting point for reference. When the system is performing well, we recommend that you save some performance data for reference. Then, should you encounter a performance problem, you can refer back to this data to make comparisons.

Example 10-8 is a simple REXX exec that will pipe the output of several CP commands and append it to a file. This could be useful in a situation where you need to make such comparisons.

Example 10-8 CP command exec

```
/*   CP Performance data    */
'pipe literal ***********' time() '|>> myperf data a'
'pipe literal ***********' date() '|>> myperf data a'
'pipe cp q alloc page   |>> myperf data a'
'pipe cp ind load       |>> myperf data a'
'pipe cp ind queues exp |>> myperf data a'
'pipe cp q frames       |>> myperf data a'
exit
```

You should also collect data before and after any changes are made that may affect the system performance. Such changes might be:

- A hardware upgrade
- New applications added
- Additional guests defined
- A new release of an operating system
- Performance tuning changes (for example, SRM STORBUF)

An additional best practice to follow, especially when changing system parameters, is to only change one thing at a time and then measure the result. If failures occur after a mass change, it is very difficult to find which change produced the failure.

There are two main analysis areas where you might find yourself involved in z/VM performance management:

- Reactive, when you are reacting to a user-reported problem
- Predictive, when you are analyzing data to either predict potential areas of constraint, or to plan for more capacity

In the following sections we describe system performance scenarios that you might encounter, and suggest actions that you might take to solve the problems.

10.7.1 Reactive analysis

When you are analyzing data, ask the following questions to help you to determine the cause of reported problems:

- Is a single user having the problem?
  - What application or guest operating system are they running?
  - Is there anything special about this user?
  - What did they do immediately prior to the problem?
  - Has the guest been run before?
- Is a group of users having the problem?
  - Do they use the same data?
  - Do they have similar virtual machines?
  - Do they use the same network resources?
- Are all users having the problem?
  - Has anything changed recently on the system?
  - Have any trends shown any change?
- What type of problem is it?
  - Data access?
  - Response time?
  - Logon?
  - Abnormal ending of a program (abend)?

Whatever the problem, probably the first thing that needs to be done is to issue a few CP commands; run an exec similar to that suggested in Example 10-8 on page 343; or access any real time monitor that may be available.

For comprehensive information about analyzing performance-related problems, refer to *z/VM Performance Version 5 Release 3*, SC24-6109, and *z/VM Performance Toolkit Guide*, SC24-6156.

- Problem detection/preliminary analysis
- Response time
- Analyzing a paging bottleneck
- Analyzing a storage problem
- Analyzing an I/O bottleneck
- Analyzing an overloaded CPU
- Analyzing a single virtual machine

In the following sections, we provide examples that show what sort of performance issues should be examined, and also provide suggested solutions.

Example: User using too much CPU

In this case, as shown in Example 10-9 on page 345, a user is using far too much of a resource.

Example 10-9 Looping user

```
ind user clive
USERID=CLIVE    MACH=XA  STOR=32M VIRT=V XSTORE=NONE
IPLSYS=CMS      DEVNUM=00012
PAGES: RES=00000177 WS=00000177 LOCKEDREAL=00000000 RESVD=00000000
NPREF=00000000 PREF=00000000 READS=00000000 WRITES=00000000
XSTORE=000000 READS=000000 WRITES=000000 MIGRATES=000000
CPU 00: CTIME=00:02 VTIME=000:15 TTIME=000:15 IO=000049
        RDR=000000 PRT=000000 PCH=000000 TYPE=CP   CPUAFFIN=ON
ind user clive
USERID=CLIVE    MACH=XA  STOR=32M VIRT=V XSTORE=NONE
IPLSYS=CMS      DEVNUM=00012
PAGES: RES=00000177 WS=00000177 LOCKEDREAL=00000000 RESVD=00000000
NPREF=00000000 PREF=00000000 READS=00000000 WRITES=00000000
XSTORE=000000 READS=000000 WRITES=000000 MIGRATES=000000
CPU 00: CTIME=00:02 VTIME=000:22 TTIME=000:22 IO=000049
        RDR=000000 PRT=000000 PCH=000000 TYPE=CP   CPUAFFIN=ON

ind queues exp

PERFSVM       Q0 R00 00004308/00004288 .I..  .0004 A03
CLIVE         Q3 R02 00000177/00000177 ..D.  .0199 A02
PVM           Q1 PS  00000346/00000323 .I..  99999 A00

01 11:43:56 FCXUSL317A User CLIVE %CPU 97.9 exceeded threshold 30.0 for 5 min.
```

This example shows some of the indicators of a user that is running a looping (never-ending) program. The **IND USER** command should be issued several times with a small delay between each and the VTIME, TTIME and IO fields should be examined.

If all are incrementing, it is likely that the user is processing data and there may not be a problem. If VTIME is increasing but there is no corresponding increase in IO, then the user may be in a loop. The **CP INDICATE QUEUES** can also be used to see if the user is waiting on any system resource.

The example also shows that PTK has detected an exception and this will be highlighted in the PTK exception report, as well as on the PTK operator console. You can gather more detailed information from the PTK menu item **21. User resource usage*** and by then selecting the user from the list that is presented.

Suggested solution

In a case like this you would suggest that the user enter: `HX` if it is a CMS application and see if that had any effect, or enter: **#CP IPL CMS**. If all user actions fail to produce a result, then a **CP FORCE userid** could be used to log the user off.

If a guest operating system such as z/OS displays symptoms like this, it is usually best to let the z/OS operators sort out the problem. But it may be necessary to force the guest if nothing can be done.

Example: System hang

In this case, the entire system hangs for a few minutes and then recovers. Users, if not autologged, have to log back on.

This may be the symptom that is seen when an internal error occurs in CP and an abnormal end (abend) is forced. When this happens, CP will try to reIPL itself and will produce a dump of storage. If the defaults in the SYSTEM CONFIG have not been changed, the dump of storage will be available on the reader of user ID OPERATNS. This can be processed, and you can research the cause of the abend in *z/VMCP Messages and Codes*, GC24-6119, in the chapter on system codes. Some of the codes can indicate a severe performance problem; for example, an abend code of PGT004 would most likely indicate a severe lack of paging space on disk.

Suggested solution

Use the **CP QUERY ALLOC PAGE** command to show the percentage of space used. If this is over 70%, add some paging space as soon as possible to prevent a reoccurrence.

Example: Users cannot log on

In this case, some guest operating systems are running without a problem, but other users are unable to log on.

In some situations CP will be so short of a resource that it starts “thrashing”, a term that means that CP dominates the system trying to get the resources necessary to dispatch users but never gets around to actually dispatching them. The users already in the dispatch list may carry on running. Other users will show as being on the eligible list; see Example 10-10.

Example 10-10 Eligible list

```
ind load
AVGPROC-058% 03
XSTORE-000200/SEC MIGRATE-0055/SEC
MDC READS-000022/SEC WRITES-000000/SEC HIT RATIO-100%
PAGING-567/SEC STEAL-075%
Q0-00002(00000)                              DORMANT-00022
Q1-00005(00000)           E1-00000(00000)
Q2-00009(00000) EXPAN-001 E2-00000(00000)
Q3-00012(00000) EXPAN-001 E3-00002(00000)

PROC 0000-058% CP       PROC 0001-057% CP
PROC 0002-061% CP       PROC 0003-058% CP

LIMITED-00000
```

Suggested solution

This problem can be caused by insufficient storage. Use the QUERY FRAMES command to display current usage of storage. The SET SRM STORBUF command can be used to control this for different user classes if necessary, but this is more likely to be caused by either insufficient real storage or by having too many guests with very large virtual machines. Review all users to see if any virtual machines can be made smaller.

10.7.2 Predictive analysis

Part of the job of a system administrator may be to monitor and report on resource usage. There are several ways that you can do this, depending on what is required.

The most simple way is to periodically run the exec as shown in Example 10-8 on page 343. The data accumulated in the file that is produced can be examined using XEDIT macros such as **`ALL /data`** to produce lists that you can scroll through and examine for trends or inconsistencies.

The more detailed way is to gather monitor data and use a program such as Performance Toolkit (PTK) to analyze this data and produce the reports that are needed. PTK, by default, keeps three daily files and an accumulation file with performance information, although this can be changed using a PTK exit if required.

PRINT command

The PTK command **`PRINT`** can be used to produce many reports whenever they are required. The reports can be tailored using the FCONX REPORTS file, which gives the following areas in which modifications can be made:

- General system data
- I/O device data
- Virtual network data
- User data
- System load by time
- Benchmarking logs for specific I/O devices and users

Graphics

PTK has facilities to produce graphs from the daily history files and the accumulation file. The graphics menu can be displayed from the main PTK menu using option 31. This can be useful in detecting trends.

Tip: When selecting the data that you would like displayed, place a question mark (?) in the box and a selection will be made available.

The selection will be either in a CMS window (if you are logged on to the PERFSVM virtual machine), or in a drop-down menu if you are using the graphical interface from a browser; see Figure 10-7 on page 348.

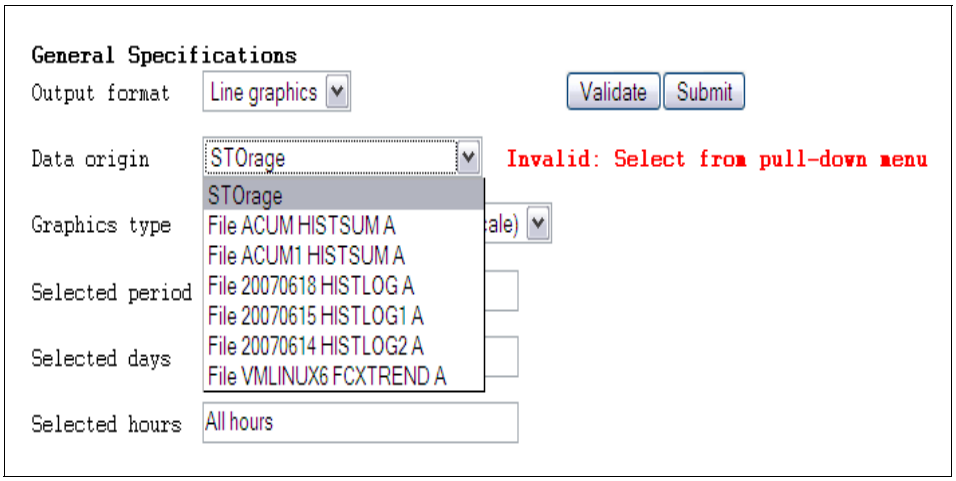


Figure 10-7 PTK graphics selection

Figure 10-8 on page 349 shows an example using PTK Web access to produce a graph of a trend in emergency scans that eventually required an IPL to relieve the problem. The graph shows the gradual build-up until the IPL, and a return to none after the IPL.

An emergency scan occurs when CP is trying to find pages of storage in which to run virtual machines. If there is not enough storage, or if there are too many users with large virtual machines, then CP will not be able to satisfy the requirements to run guests and will have to take active pages from other users. A situation like this could occur if users were added over a period of time without first planning for the required capacity. Using this information, it may be possible to avoid unscheduled outages.

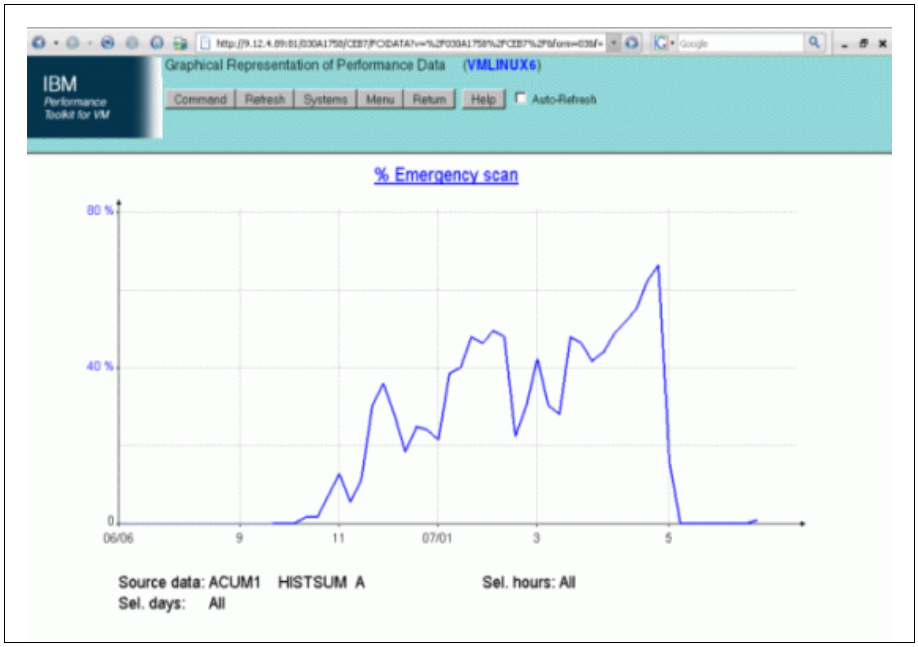


Figure 10-8 ACUM file

10.7.3 Tuning guidelines

So far, we have discussed the tools available to help you gather data, provided references to other sources for additional information that will help you to analyze and tune your system. Next, we present some ideas to consider if you find that your system is showing symptoms of degraded performance.

CPU

There is not much that you can do if you are genuinely using 100% CPU, except perhaps to upgrade your hardware or reduce workload. If you do find the system in this state, you can do a few things to minimize the impact.

- Try to persuade users to stagger their workload. For instance, database reorganizations should be scheduled outside prime shift.
- Reduce the number of virtual processors that heavy CPU users have available.
- Use SET SHARE nn% LIMITHARD to cap heavy users. Other SHAREs can be used to apportion CPU according to user needs. This may cause users to enter the Limit list.
- Use SET SRM DSPBUF to limit the users in the dispatch list.

- Use SET SRM DSPSLICE to see if either making the dispatcher time slice smaller or larger has any effect.

Storage

A shortage of storage is normally associated with increased paging activity, so these should be looked at together. If you do not have many frames available and you are not paging or stealing (page taken from in-queue user), then you are probably not seeing any performance problems.

If, however, you do not have many frames free and you are seeing high rates of paging and stealing, then some problem determination may be needed. Here are some suggested actions:

- Use SET SRM STORBUF to reserve some storage for more interactive users. Use caution and measure, as this may cause heavier users to enter the eligible list.
- Reduce the size of virtual machines.
- Investigate the use of RESERVE or LOCK for some guests.

Paging

Paging is a very basic part of any operating system that uses virtual memory, and a paging subsystem that works well is essential for optimum performance. To ensure that this occurs, use the Q ALLOC PAGE command regularly to check on usage. The I/O to the paging DASD performs better if there is a great deal of free space available. If you suspect a paging problem, or if the utilization is very high, you can perform the following actions:

- Add more paging space.
- Use SET LDUBUF to give interactive users better paging resource availability.

I/O

When reports show that certain devices are not performing as well as they should, or if certain devices seem to be busy more than others, you can perform the following actions:

- Check the data on the devices and see if it can be spread across more paths.
- Use the THROTTLE command to limit guests that are heavy users.

10.7.4 Other references

There have been numerous studies made of z/VM performance and capacity planning, many of which can be found on the z/VM Web site:

`http://www.vm.ibm.com/perf/`

This site also has numerous links to other performance resources that you will find useful. For further information about these topics, you can also refer to the following IBM Redbooks publications:

- *Linux on IBM eServer zSeries and S/390: Performance Measurement and Tuning*, SG24-6926
- *Linux on IBM eServer zSeries and S/390: Performance Toolkit for VM, SG24-6059*
- *Using the z/VM INDICATE Command*, TIPS0592
- *Using Discontiguous Shared Segments and XIP2 Filesystems With Oracle Database 10g on Linux for IBM System z*, SG24-7285

10.8 Exercises

1. Which CP command should you use to find out paging space utilization?
2. How could you find out if a user is using too much CPU?
3. What CP system service does a user need to access MONITOR data?
4. Which three lists are used by the scheduling and dispatching code?
5. What could you use a DCSS for?
6. Which command would you use to display information about real storage?
7. List six displays that you could use with the INDICATE command.
8. What types of information does the monitor collect?

11

Networking and connectivity

This chapter introduces you to the various networking options available in z/VM, and shows you how to use them in a hands-on manner. You will also learn some of the basic networking commands that are available when using or troubleshooting connectivity problems.

Objectives

After completing this chapter, you will be able to:

- List the types of networking devices supported by z/VM
- Discuss the various virtual network types z/VM offers
- Explain how to create a virtual switch
- Grant network access to guest operating systems
- Use common TCP/IP related commands provided by z/VM
- List the network services supported by z/VM

© Copyright IBM Corp. 2007. All rights reserved.

11.1 Introduction to networking in z/VM

If the Internet age has taught us anything, it is that the value of a resource, such as a computer system, is significantly greater when it is connected to a network and allowed to share information.

z/VM recognizes this and supports most if not all of the hardware connectivity options available on System z, and also implements many more in virtual form. The z/VM virtual networking options can be faster than using real hardware if the network traffic is to another guest, because it will travel at the speed of main memory. Virtual networking also does not require as much network hardware, so it is usually less expensive.

To provide connectivity to a guest operating system you must either dedicate real hardware I/O channels to the guest or create a virtual network interface card (NIC) and connect (*couple*, in z/VM terms) it to a virtual local area network (LAN) segment.

In addition to providing network connectivity to guest operating systems, z/VM itself has a Transmission Control Protocol/Internet Protocol (TCP/IP) stack that allows you to connect to the system via a 3270 terminal emulator to administer the system and its guests.

11.1.1 I/O channel requirements

The network interface cards most often used on System z require three I/O channels for connectivity: a read channel, a write channel, and a data channel. The read and write channels are used to receive and send control data to the network adapter. The data channel is used to transmit the network traffic. These three channels together allow a system to send and receive data over a network.

Note: Typically the three I/O channels have consecutive device addresses, such as 600, 601, and 602. Older mainframes had a microcode limitation that required the read channel to be an even number and the write channel to be one greater than the read channel. You could then use any remaining channel for the data channel.

When dealing with real hardware, you usually have to explicitly specify all of the device addresses you wish to use. For virtual NICs or LANs, z/VM often requires only the base address (that is, the first address in the sequence) and will automatically compute the next two consecutive addresses for you.

11.2 Supported network devices

As previously mentioned, z/VM supports a large number of hardware network devices. The following list is just a sample:

- Open Systems Adapter 2 (OSA-2)
- Open Systems Adapter Express (OSA Express)
- HiperSockets
- Channel-to-channel (CTC)
- LAN Channel Station (LCS)

Any of these devices can be dedicated or defined for individual guests.

More information on these devices can be found in *z/VM TCP/IP Planning and Customization*, SC24-6125. Also see *z/VM Connectivity*, SC24-6080. *IBM System z Connectivity Handbook*, SG24-5444, provides excellent explanations of the hardware options available on System z.

11.2.1 Open Systems Adapter

The Open Systems Adapter (OSA) is a network controller that you can install in a mainframe I/O cage. The adapter integrates several hardware features and supports many networking transport protocols, including fast Ethernet, gigabit Ethernet, 10 gigabit Ethernet, Asynchronous Transfer Mode (ATM), and token ring.

There are several versions of the OSA available: OSA-2, OSA-Express, and OSA-Express2, each of which have slightly different features and functionality. The older OSA-2 cards have been superseded by the newer OSA-Express and OSA-Express2 cards and are no longer available, but many older mainframes still use them.

Due to the fact that mainframes typically have many guest operating system instances running on them, the latest OSA-Express2 cards can have up to 640 TCP/IP stacks connected to them at any one time. This means that up to that many operating systems can be utilizing the card at any one time. While 640 is a decent number, older OSA cards and mainframe models had lower limits that were often not sufficient.

Another key feature of OSAs is support for the System z Queued Direct Input/Output (QDIO) Hardware Facility, which allows the OSA card to buffer data directly in the host's main storage, bypassing much of the I/O process.

11.2.2 HiperSockets

System z HiperSockets is an extension to the QDIO Hardware Facility that provides a microcode feature that enables high-speed TCP/IP connectivity between virtual servers within a System z server.

Regular OSA devices require a physical network connection in order to communicate with systems outside of the CEC. The communication path for a HiperSockets network is within main memory, so it operates at memory speeds using the internal QDIO (iQDIO) feature of the System z hardware. This also means that no physical cable is necessary, because the communication takes place entirely with the CEC.

11.2.3 Channel-to-channel connection

A channel-to-channel (CTC) connection is a direct, one-to-one connection that allows two guests to communicate with one another. Think of this as the “telephone” game that you may have played as a youngster: you have a tin can and it is connected to your friend's tin can by a string, which allows you to communicate with one another via the cans if the string is kept taut. This is similar to CTCs (they are both point-to-point connections), and has the same limitation: if you want to talk to anyone other than that friend, you need additional pairs of cans and more string.

Today CTCs are used to connect different CECs together so that one z/VM system can communicate with other z/VM systems natively. In order for the two systems to communicate, a dedicated cable must be plugged in to both. Fortunately, z/VM can emulate a CTC; so if you only need to communicate with a system running in the same z/VM instance, you can use a virtual CTC.

Prior to the addition of guest LANs in z/VM 4.2, CTCs were one of the two options for Linux network connectivity, which was often quite involved when there was a significant number of systems because each pair would need to be cabled together or have virtual CTCs configured.

CTC devices also differ slightly from most System z networking options in that they only require two I/O channels to send and receive data.

11.3 Virtual network types supported by z/VM

In addition to dedicating hardware network adapters to guests, z/VM also supports several different virtual networking options that allow you to connect additional systems to the network.

The current types supported by z/VM 5.3 are:

- Inter-User Communications Vehicle (IUCV)
- QDIO guest LAN
- HiperSocket guest LAN
- Virtual switch (VSWITCH)

In the following sections we examine each LAN type in more detail and discuss when you should use each type.

11.3.1 Inter-User Communications Vehicle (IUCV)

IUCV is similar to CTC in that it is a point-to-point connection. It is used extensively when z/VM components need to communicate with each another or with CP. Recent patches have even added IUCV support to the Linux kernel in the form of the AF_IUCV socket family.[1] Before the addition of guest LAN support to z/VM, IUCV was the other option for Linux network connectivity.

11.3.2 Guest LAN

A guest LAN represents a simulated LAN segment that can be connected to simulated network interface cards. There are two types of simulated LANs that z/VM supports:

- QDIO, which emulates an OSA-Express
- Internal QDIO (iQDIO), which emulates a HiperSockets connection

Note: Internal QDIO is usually referred to as “HiperSockets guest LAN” in z/VM literature, and we use the same convention here.

Each guest LAN is isolated from other guest LANs on the same system (unless some member of one LAN group acts as a router to other groups).

By default, the standard z/VM guest LANs allow any number of guests to be connected to the network. Guest LANs make it very easy for guests to communicate on their own subnet. The primary disadvantage is that either a z/VM service machine or Linux guest must be present to route traffic if there is any communication outside of the physical machine or to any other subnet.

Another feature of guest LANs is that any class G guest can create its own transient LAN that other guests can connect (*couple* in z/VM terminology) to. Additionally, by default, guest LANs are unrestricted, meaning anyone can

[1] See “AF_IUCV protocol support for Linux on System z” which is available on the Web at: http://www.ibm.com/developerworks/linux/linux390/useful_add-ons_af-iucv-v1.html

couple to it without any special permissions. This makes it very easy to create private LANs that function only between virtual machines.

When a guest LAN is created, you specify which type of LAN you want it to be (QDIO or HiperSockets). Note that each network interface card (NIC) that is coupled to it must be of the *same* type; otherwise, they will be unable to communicate. Therefore, QDIO NICs are coupled to QDIO guest LANs, and HiperSockets NICs are coupled to HiperSockets guest LANs.

Figure 11-1 illustrates a logical view of how a guest LAN operates.

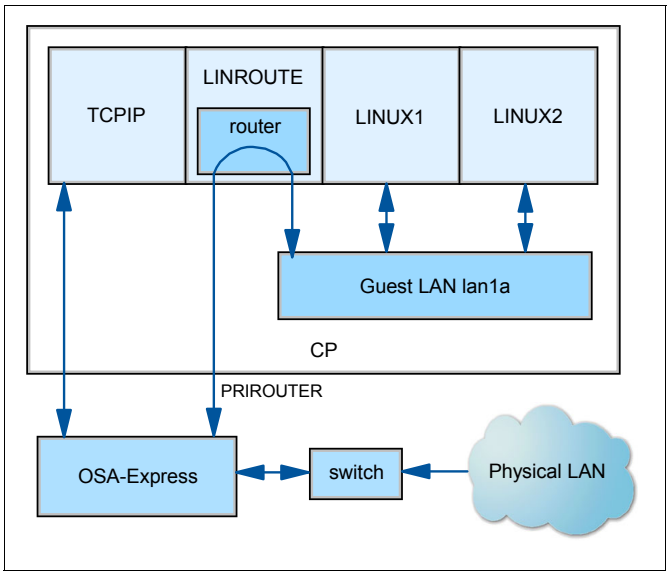


Figure 11-1 Architecture diagram for a guest LAN

11.3.3 Virtual switch

Virtual switch, or VSWITCH, is the newest LAN type supported by z/VM. It was introduced with the 4.4 release of the operating system and is a special type of guest LAN that provides external LAN connectivity through an OSA-Express device without the need for a routing virtual machine.

As of z/VM 5.1, virtual switches can operate in either ethernet (link layer) or IP mode (network layer) mode.[2] Note the following difference between these modes:

- With ethernet, z/VM guests act just like a standard PC on a network and use the media access control (MAC) address associated with the network card to send and receive data.

- IP mode packets are forwarded based on the IP address associated with the network card.

The primary advantage to using ethernet virtual switches versus IP mode virtual switches is that non-IP traffic such as Systems Network Architecture (SNA), Internetwork Packet Exchange (IPX™), or NetBIOS can be used, in addition to standard IPv4 or IPv6 traffic. Using layer 2 mode is recommended whenever possible both for performance and flexibility reasons.

Note: z/VM itself does not yet support ethernet networks, so at least one IP mode OSA port must be available for the z/VM TCP/IP stack.

While there is no need for a routing virtual machine, z/VM does require that one particular machine own the OSA-Express devices that are used by the VSWITCH. These special virtual machines are called *VSWITCH controllers*, and are essentially extra TCP/IP stacks that manage the OSA on behalf of the systems connected to the VSWITCH. With z/VM 5.2 and newer there are two predefined VSWITCH controllers: DTCVSW1 and DTCVSW2.

Another benefit of a VSWITCH is that it can easily be configured with redundant OSA devices and additional controllers so that in the event of a problem, either the switch can fail over to a backup OSA or controller.

Figure 11-2 on page 360 shows a logical view of a virtual switch and where it fits.

[2] See the “OSI model” topic on Wikipedia for more information about the Open Systems Interconnection Basic Reference Model: http://en.wikipedia.org/wiki/OSI_model

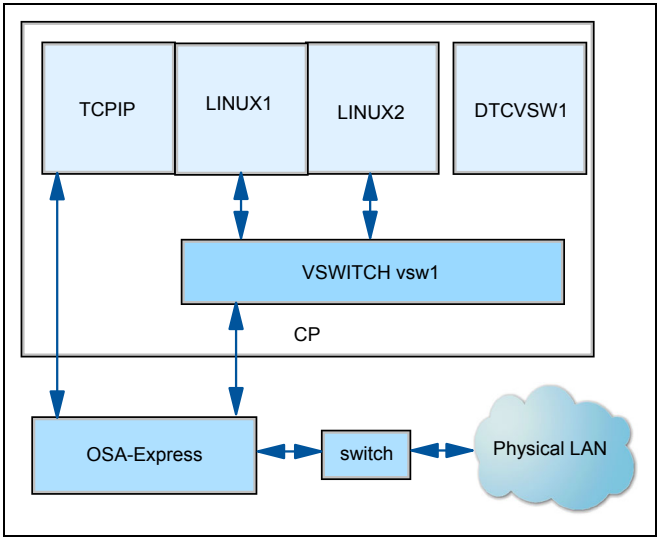


Figure 11-2 Architecture diagram for a virtual switch

Notice that in the case of virtual switches, the VSWITCH vsw1 is part of the same LAN segment as the external switch shown in the bottom of the diagram. Contrast this with the case of a guest LAN (such as the one shown in Figure 11-1 on page 358), where the guest LAN and the external switch are on different LAN segments connected via a router virtual machine.

11.4 Defining a VSWITCH

A VSWITCH can either be created dynamically by MAINT or another class B user with the following command:

```
DEFINE VSWITCH <switchname> RDEV <rdev>
```

Where `switchname` is the desired virtual switch name and `rdev` is the base address of the OSA device triplet. Example 11-1 shows the dynamic creation of a VSWITCH.

Example 11-1 Dynamically creating a VSWITCH

```
define vswitch vsw1 rdev 111
VSWITCH SYSTEM VSW1 is created
```

You can then use the QUERY VSWITCH command to verify that it was defined correctly, as demonstrated in Example 11-2 on page 361.

Example 11-2 Using the QUERY VSWITCH command

```
query vswitch
VSWITCH SYSTEM VSW1     Type: VSWITCH Connected: 0    Maxconn: INFINITE
  PERSISTENT RESTRICTED    NONROUTER                 Accounting: OFF
  VLAN Unaware
  MAC address: 02-00-00-00-00-01
  State: Ready
  IPTimeout: 5         QueueStorage: 8
  RDEV: 0111 VDEV: 0111 Controller: DTCVSW2 A
```

As you can see, the VSWITCH was created and as line **A** shows, z/VM automatically assigned DTCVSW2 to be the VSWITCH controller. If there were additional virtual switches defined on the z/VM system, they too would be displayed.

By default, VSWITCH definitions are volatile and will not exist if the system is restarted. To make the change permanent, the VSWITCH definition must be added to the SYSTEM CONFIG configuration file.

11.4.1 Enabling VSWITCH failover

The DEFINE VSWITCH command lets you specify the real devices for external connectivity. Starting in z/VM 5.3, up to eight real devices can be specified for use by the virtual switch. Prior to z/VM 5.3, the limit was three real devices.

If you specify more than one real device address, the additional devices are used as standbys in case of a problem with the primary OSA-Express device. For example, if you specify the following, then the VSWITCH will use devices 111-113 to provide access to the real hardware LAN:

```
DEFINE VSWITCH vsw1 RDEV 111 222 333 CONTROLLER *
```

If there is a problem with the connection devices, then 222-224 are used next to provide the connection. If those devices fail to connect, then devices 333-335 are used.

Specifying `CONTROLLER *` (or allowing it to default to that value) is important because it means z/VM will automatically spread the controller functions across multiple TCP/IP stacks, thereby providing more flexibility in case of a failure.

By default, z/VM will check time stamps to make sure that the active VSWITCH controller is responding to requests. If it is not, CP moves to a backup OSA-Express device and controller if one is available. For more information about Virtual Switch Failover, refer to *z/VM: Connectivity*, SC24-6080.

11.5 Connecting guests to the network

In the following sections, we describe in detail the various methods of connecting guests to the network.

11.5.1 Dedicating OSA devices

In *9.2, "CP commands" on page 276*, you learned that if you want to grant a particular virtual machine sole use of a physical resource, you use the ATTACH command or the DEDICATE directory statement to do so. The results of the ATTACH command are transient, and will be undone when the guest logs off. By adding a DEDICATE statement to the guest's directory entry, the assignment becomes persistent.

Typically if a system administrator wants a single guest to have network connectivity but does not want to create virtual network segments, the administrator will dedicate a set of OSA devices to the guest.

To attach an OSA device to a guest, log on as the MAINT user and use the following command:

```
ATTACH xxx-yyy TO <guestname>
```

Where `xxx` and `yyy` are a three-device range of OSA device addresses that will be attached to the specified guest, as demonstrated in Example 11-3.

Example 11-3 Temporarily attaching OSA devices using real addresses

```
attach 606-608 to linux1
606-608 ATTACHED TO LINUX1
```

Some system programmers like to attach real hardware using virtual device addresses in order to have consistent hardware environments. This is particularly important in disaster recovery situations, where the hardware at the backup location may not be the same as the primary. By using virtual device addresses, you could make the backup system look like the original and boot your operating systems.

If you wanted to attach them using a different device address, you need to attach the OSA address by address using the following form of the ATTACH command:

```
ATTACH <rdev> TO <guestname> AS <virtdev>
```

Example 11-4 on page 363 demonstrates using this form of the command to attach the same OSA devices to a guest.

Example 11-4 Temporarily attaching OSA devices at virtual addresses

```
attach 606 to linux1 as 600
OSA 606 ATTACHED TO LINUX1 0600
attach 607 to linux1 as 601
OSA 607 ATTACHED TO LINUX1 0601
attach 608 to linux1 as 602
OSA 608 ATTACHED TO LINUX1 0602
```

When a set of OSA devices has been dedicated to a guest, you can verify that it worked (after the guest has logged off and logged back on) by using the QUERY OSA command as demonstrated in Example 11-5.

Example 11-5 Output of the QUERY OSA command

```
query osa
OSA  0600 ON OSA   0606 SUBCHANNEL = 0000
     0600 DEVTYPE OSA         CHPID 01 OSD
     0600 QDIO-ELIGIBLE       QIOASSIST-ELIGIBLE
OSA  0601 ON OSA   0607 SUBCHANNEL = 0001
     0601 DEVTYPE OSA         CHPID 01 OSD
     0601 QDIO-ELIGIBLE       QIOASSIST-ELIGIBLE
OSA  0602 ON OSA   0608 SUBCHANNEL = 0002
     0602 DEVTYPE OSA         CHPID 01 OSD
     0602 QDIO ACTIVE         QIOASSIST ACTIVE
     0602
     0602 INP + 01 IOCNT = 00435919  ADP = 008 PROG = 000 UNAVAIL = 120
     0602          BYTES = 00000000047343A2
     0602 OUT + 01 IOCNT = 00000000  ADP = 000 PROG = 000 UNAVAIL = 128
     0602          BYTES = 0000000000000000
     0602 OUT + 02 IOCNT = 00000000  ADP = 000 PROG = 000 UNAVAIL = 128
     0602          BYTES = 0000000000000000
     0602 OUT + 03 IOCNT = 00067481  ADP = 000 PROG = 128 UNAVAIL = 000
     0602          BYTES = 0000000000DC582D
     0602 OUT + 04 IOCNT = 00000000  ADP = 000 PROG = 000 UNAVAIL = 128
     0602          BYTES = 0000000000000000
```

To make the change permanent, you need to edit the guest's directory entry and add a line similar to the following for each of the three OSA device addresses:

```
DEDICATE <virtdev> <realdev>
```

Where `virtdev` is the virtual device address that you would like the real device (`realdev`) address attached as. Example 11-6 on page 364 continues our previous example and shows the additions to the directory entry for LINUX1.

Example 11-6 Permanently dedicating OSA devices

```
USER LINUX1 NOP4SSWD 512M 1G G
...
DEDICATE 0600 0606
DEDICATE 0601 0607
DEDICATE 0602 0608
...
```

Note: The order of the arguments in the DEDICATE statement is the opposite of similar statements like LINK that are used in z/VM.

11.5.2 Coupling to a VSWITCH or guest LAN

In order to connect a guest to a VSWITCH or guest LAN, it first needs a virtual NIC. Depending on the guest's directory entry, one or more NICs may have already been defined. The command to view NICs is QUERY NIC. Example 11-7 shows the output from a guest with two NICs.

Example 11-7 Output of the QUERY NIC command

```
query nic
Adapter 0700  Type: QDIO      Name: UNASSIGNED  Devices: 3
  MAC: 02-00-00-00-00-01         LAN: * None               MFS: 8992
Adapter 0800  Type: QDIO      Name: UNASSIGNED  Devices: 3
  MAC: 02-00-00-00-00-03         LAN: * None               MFS: 8992
```

Note: OSA devices do not appear in the output of the QUERY NIC command. You must use the QUERY OSA command as already discussed in order to see dedicated OSA devices.

Assuming a NIC did not already exist, you would need to use the following two commands to define a NIC and to couple it to the VSWITCH:

```
DEFINE NIC <devnum> TYPE QDIO
COUPLE <devnum> TO SYSTEM <switchname>
```

Note that `devnum` is the base device address to use and `switchname` is the name of a virtual switch that exists on the system. Example 11-8 shows the results of these two commands.

Example 11-8 Coupling a NIC to a VSWITCH

```
define nic 700 type qdio
NIC 0700 is created; devices 0700-0702 defined
```

```
Ready; T=0.01/0.01 12:20:07
couple 700 to system vsw1
NIC 0700 is connected to VSWITCH SYSTEM VSW1
Ready; T=0.01/0.01 12:27:08
query nic
Adapter 0700  Type: QDIO      Name: UNASSIGNED  Devices: 3
  MAC: 02-00-00-00-00-01         VSWITCH: SYSTEM VSW1
Adapter 0800  Type: QDIO      Name: UNASSIGNED  Devices: 3
  MAC: 02-00-00-00-00-03         LAN: * None               MFS: 8992
```

By default, NICs defined using DEFINE NIC are transient and only persist until the guest is logged off. To permanently define a NIC for a guest, you need to edit the guest's directory entry. Refer to *CP Planning and Customization* for detailed explanations about how to edit the directory and define a NIC.

Understanding VSWITCH access controls

Recall that virtual switches are restricted which means that any new guests that want to couple to a VSWITCH must first be granted access. For access, which will remain in effect until z/VM is rebooted, you can log on as MAINT and use the GRANT command as follows:

```
SET VSWITCH <switchname> GRANT <guestname>
```

Example 11-9 demonstrates using GRANT.

Example 11-9 Granting access to a VSWITCH

```
set vswitch vsw1 grant linux1
Command complete
```

After access has been granted, you can verify the access list by using the QUERY VSWITCH ACCESSLIST command as demonstrated in Example 11-10.

If you are on the access list for any virtual switch, or if you are currently coupled to one, the QUERY VSWITCH ACCESSLIST command will work.

Example 11-10 Querying the access list for a virtual switch

```
q vswitch accesslist
VSWITCH SYSTEM VSW1     Type: VSWITCH Connected: 0    Maxconn: INFINITE
  PERSISTENT  RESTRICTED    NONROUTER                 Accounting: OFF
  VLAN Unaware
  MAC address: 02-00-00-00-00-01
  State: Ready
  IPTimeout: 5         QueueStorage: 8
  Authorized userids:
```

```
     LINUX1   SYSTEM
  RDEV: 0111 VDEV: 0111 Controller: DTCVSW2
```

To give permanent access, you need to edit the system configuration file. If you are using an external security manager (ESM) such as RACF/VM, you need to add the appropriate rules to the RACF database.

11.6 TCP/IP commands provided by z/VM

z/VM provides a number of commands designed to help you obtain information about the state of the network. Most of these tools reside on the TCPMAINT user's 592 minidisk, which all users should be able to link to and access. TCPMAINT is a special user account that is used to administer the TCPIP stack and other network service machines.

Although some of the basics are covered in the following sections, you can find complete instructions about using and administering TCP/IP services in z/VM in *z/VM TCP/IP User's Guide*, SC24-6127.

11.6.1 NETSTAT

The NETSTAT command displays information about the status of the local host. It will display information about the TCP/IP configuration, connections, network clients, gateways, devices, and the Telnet server. NETSTAT also drops connections and executes commands for the users in the TCPIP virtual machine's OBEY list.

The OBEY list is a special statement in the z/VM TCP/IP configuration that controls which users are allowed to use privileged TCP/IP functions. By default the only users with the proper access are MAINT and TCPMAINT, along with a few of the special service virtual machines. Refer to *z/VM TCP/IP Planning and Customization*, SC24-6125, for more information about the OBEY list.

NETSTAT ALL

The NETSTAT ALL command displays information about all TCP/IP connections, such as the window and sequence numbers, and the sender frustration level.

Example 11-11 on page 367 shows part of the output from running NETSTAT ALL.

Example 11-11 Information displayed by the NETSTAT ALL command

```
Client: FTPSERVE                          Last Touched: 382:10:27 A
  Local Socket:   *..FTP-C
  Foreign Socket: *..* B
     BackoffCount: 0
     ClientRcvNxt: 0
     ClientSndNxt: 774115649
     CongestionWindow: 0
     Local connection name: 1004
     Sender frustration level: Contented
     Incoming window number: 0
     Initial receive sequence number: 0
     Initial send sequence number: 774115648
     Maximum segment size: 536
     Outgoing window number: 0
     Precedence: Routine
     RcvNxt: 0
     Round-trip information:
        Smooth trip time:   0.000
        Smooth trip variance:   1.500
     SlowStartThreshold: 0
     SndNxt: 774115648
     SndUna: 774115648
     SndWl1: 0
     SndWl2: 0
     SndWnd: 0
     MaxSndWnd: 0
     State: Listen C
     No pending TCP-receive
```

The output you receive may vary, but here are a few interesting pieces of information revealed in the output:

A This shows how long the socket has been idle.

B This indicates that there is no remote socket connected.

C This socket is listening for connections, explaining why there is no foreign socket.

NETSTAT CONFIG

The NETSTAT CONFIG command displays the TCP/IP service machine's configuration information. Additional options can be passed to this command to view subsets of the configuration. The available parameters are documented in *z/VM TCP/IP User's Guide*. Example 11-12 on page 368 shows some of the output from NETSTAT CONFIG.

Example 11-12 Sample output from NETSTAT CONFIG

```
netstat config
VM TCP/IP Netstat Level 530

Assorted Parameters
  Check Consistency:         No       CLAW Double NOP:           No
  CP Dump:                   No       VM Dump:                   No
  Equal Cost Multipath:      No       IPv6 Equal Cost Multipath: No
  Ignore Redirects:          No       IPv6 Ignore Redirects:     No
  ACB Cushion:               Yes      Forwarding Enabled:        Yes
  Warn on Level Mismatch:    Yes      RFC1323 Support            Yes
  UDP Queue Limit:           Yes      Override Precedence:       No
  Permitted Users Only:      No       Proxy ARP:                 Yes
  Restrict Low Ports:        Yes      Secure Local:              No
  Source VIPA:               No       IPv6 Source VIPA:          No
  Stop On CLAW Error:        No

Internal Client Settings
  Asynchronous Input:   No           CCS Terminal Name:       TCPIP
  Connection Exit:      <none>       EOJ Time Out:            120
  Go Aheads Disabled:   No           Ignore EAU Data:         No
  Inactivity Timeout:   0              LDev Range:               0000 -
OFFF
  Scan Interval:        60             Timemark Timeout:        600
  TN3270E Enabled:      Yes            Use SC Exit for TN3270E: No
  TN3270E Exit:         <none>         Transform:               No
  Secure Connection:    Never          TLS Label:               <none>
  Port(s):              23
...
```

NETSTAT CONN

The NETSTAT CONN command shows what connections are currently active on your system; see Example 11-13. The command displays the following information about active TCP/IP connections: user ID, connection number, local and foreign socket information, and the connection state. This command provides much of the same information as the Linux **netstat** command.

Example 11-13 Sample output from NETSTAT CONN

```
netstat conn
VM TCP/IP Netstat Level 530

Active IPv4 Transmission Blocks:
```

```
User Id  Conn    Local Socket          Foreign Socket        State
-------  ----    ------------          --------------        -----
INTCLIEN 1001    *..TELNET             *..*                  Listen
INTCLIEN 1000    9.12.4.200..TELNET    9.12.5.248..3298
Established
FTPSERVE 1004    *..FTP-C              *..*                  Listen

Active IPv6 Transmission Blocks: None
```

NETSTAT DEVLINKS

The NETSTAT DEVLINKS command provides information that is similar to that provided by the Linux **ifconfig** command. The NETSTAT DEVLINKS command is particularly useful for viewing how much data has been sent or received via the device (see the BytesIn and BytesOut values in Example 11-14).

Example 11-14 Output of the NETSTAT DEVLINKS command

```
netstat devlinks
VM TCP/IP Netstat Level 530

Device OSA2040D                Type: OSD            Status: Ready
  Queue size: 0     CPU: 0     Address: 2040        Port name: OSA2040
  IPv4 Router Type: NonRouter  Arp Query Support: Yes
    Link OSA2040L              Type: QDIOETHERNET   Net number: 0
      BytesIn: 47348747        BytesOut: 1743741
      Forwarding: Enabled      MTU: 1500            IPv6: Disabled
      Broadcast Capability: Yes
      Multicast Capability: Yes
      IPv4 VIPA ARP
      Group                                    Members
      -----                                    -------
      224.0.0.1                                   1
```

NETSTAT DOS

One of the security features in the z/VM TCP/IP stack is denial-of-service (DOS) detection. Periodically, when running NETSTAT commands, you may see a message similar to this:

```
DTCNET400W A denial-of-service attack has been detected; issue
NETSTAT DOS for more information
```

Running the NETSTAT DOS command will provide you with a listing of potential attackers, as illustrated in Example 11-15 on page 370.

Example 11-15 Output from NETSTAT DOS

```
netstat dos
VM TCP/IP Netstat Level 520

Maximum Number of Half Open Connections: 256
Denial of service attacks:
                                                  Attacks   Elapsed    Attack
Attack   IP Address                             Detected      Time  Duration
-------- ------------------------------------ --------- --------- ---------
Smurf-IC 192.168.70.95                              108 765:57:18  19:26:44
         192.168.70.94                               80 761:21:16  11:36:17
         192.168.70.184                             916 322:33:55  48:36:58
         192.168.70.185                             308 322:16:07  61:16:32
         192.168.70.99                               27 205:31:38   0:00:37
```

NETSTAT POOLSIZE

If you find that your z/VM system is having problems sending or receiving packets, it may be that you need to tune the data buffer pools if network traffic is particularly heavy. In particular, certain versions of z/VM had problems with the FTP MPUT or MGET commands where they would use separate connections to transfer each file, eventually consuming all of the server control block (SCB) buffers in the pool. When the pool was empty, all socket connections would be blocked.

In order to determine whether buffers are being exhausted, you can either wait until z/VM sends an alert, or you can use the NETSTAT POOLSIZE command, which displays statistics about the various types of buffers the TCP/IP stack uses. Example 11-16 shows sample output from the command on a fairly idle system.

Example 11-16 Output from NETSTAT POOLSIZE

```
netstat poolsize
VM TCP/IP Netstat Level 530

TCPIP Free pool status:

Object            No. alloc    No. free    Lo-water   Permit size
======            =========    ========    ========   ===========
ACB                    1024        1020        1012           102
CCB                     154         144         143            15
Dat buf                 160         158         155            16
Sm dat buf               12          10           6             1
Tiny dat buf             10          10          10             1
```

```
Env                    750         750         747          75
Lrg env                 50          49          50           5
RCB                     51          51          51           5
SCB                    264         260         258          26
SKCB                   256         256         256          25
TCB                    258         255         252          25
UCB                    102         102         102          10
Add Xlate             1512        1512        1512           5
NCB                   1501        1501        1501           5
IP Route               625         623         623           6
IPv6 Route             619         619         619           6
Segment ACK           5160        5160        5156         516
FPSP total locked pages: 352, Unused locked pages: 73
FPSP allocation threshold: 2306, Low-water mark: 0
TCPIP machine size: 32M, Pools: 4609K, Avail: 8796K, Max block: 8788K
```

For more information about types of buffers, refer to the z/VM TCP/IP Performance Web site[3]. For details about diagnosing TCP/IP-related problems, refer to *z/VM TCP/IP Diagnosis Guide*, GC24-6123.

NETSTAT SOCKETS

The NETSTAT SOCKETS command displays information about each client using the z/VM socket interface. This can be useful when you attempt to debug services such as the systems management API, as well as others that use the portmapper service. Example 11-17 shows the communication between a Linux guest and the VSMSERVE guest.

Example 11-17 Output of the NESTAT SOCKETS command

```
netstat sockets
VM TCP/IP Netstat Level 520

Socket interface status:

Name: PORTMAP    Subtask: 00da9480   Path id: 1    Pending call: select
  Socket: 3   Type: Dgram
    BoundTo: *..PMAP
    ConnTo:  Not connected
  Socket: 4   Type: Stream    State: Listen        Flags: L    Conn: 1003
    BoundTo: *..PMAP
    ConnTo:  *..*
Name: VSMSERVE   Subtask: INET4001   Path id: 2    Pending call: select
  Socket: 0   Type: Stream    State: Listen        Flags: L    Conn: 1001
```

[3] http://www.vm.ibm.com/devpages/bitner/presentations/tcpip

```
  BoundTo: 172.16.1.1..845
  ConnTo:  *..*
Socket: 1   Type: Stream    State: Established   Flags:      Conn: 1008
  BoundTo: 172.16.1.1..845
  ConnTo:  172.16.1.2..956
Socket: 2   Type: Stream    State: Established   Flags:      Conn: 1004
  BoundTo: 172.16.1.1..845
  ConnTo:  172.16.1.2..895
```

11.6.2 TRACERTE

The TRACERTE command can help you to locate broken network links by showing you the path that packets take to get to their destination. The z/VM TRACERTE command is very similar to the **traceroute** command in Linux and to the **tracert** command in Windows.

TRACERTE works by sending User Datagram Protocol (UDP) requests with varying time-to-live (TTL) values, and listening for TTL-exceeded messages from the routers between the local host and the remote host.

Example 11-18 shows a typical execution of TRACERTE. Notice that the second hop does not send TTL exceeded messages and sometimes packets are lost (hops 6, 7, and 10).

Example 11-18 Using TRACERTE to check network links

```
tracerte www.zxy.com
Trace route to WWW.ZYX.COM (10.2.91.34)
 1 (10.67.22.2) 67 ms 53 ms 60 ms
2 * * *
3 (10.67.1.5) 119 ms 83 ms 65 ms
4 (10.3.8.14) 77 ms 80 ms 87 ms
5 (10.158.1.1) 94 ms 89 ms 85 ms
6 (10.31.3.1) 189 ms 197 ms *
7 * * (10.31.16.2) 954 ms
8 (10.34.31.33) 164 ms 181 ms 216 ms
9 (10.2.95.1) 198 ms 182 ms 178 ms
10 (10.2.91.34) 178 ms 187 ms *
```

11.6.3 PING

The PING command determines the accessibility of a foreign node. It behaves very similar to the **ping** command in other operating systems in that it sends an

Internet Control Message Protocol (ICMP) echo request to the specified host, which then sends a reply back to the requester. The request and response can help you determine if the remote machine is available and also how long the ICMP request/response packets took to traverse the network.

The syntax of the PING command is:

```
PING <ip address or hostname>
```

Example 11-19 demonstrates PING command use.

Example 11-19 Using PING to check a remote system's availability

```
ping landisk
Ping level 530: Pinging host LANDISK (192.168.1.107).
                Enter #CP EXT to interrupt.
PING: Ping #1 response took 0.024 seconds. Successes so far 1.
```

Note: Some hosts or networks may block ICMP packets. If you are trying to ping a computer system that you know is online and it is not responding, you may want to check with your network administrator to verify that ICMP is allowed.

11.7 The z/VM network service model

In z/VM, optional functionality is modularized and implemented as service virtual machines. A *service machine* is like any other z/VM guest, except that it typically runs a single daemon process from CMS that provides the desired functionality.

If you look in the CP user directory, you will find a number of common network services listed: Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP), Domain Name System (NAMESRV), dynamic routing (MPROUTE), and TCPIP (yes—even the TCP/IP stack for z/VM is implemented as a separate guest).

If you want your z/VM system to be accessible via a 3270 terminal emulator, then the TCP/IP guest needs to be logged on to the system. If you want to have other network services handled by z/VM, edit the DTCPARMS file and add the service machine to the PROFILE TCPIP file. Both of these tasks are described fully in *z/VM TCP/IP Planning and Customization*, SC24-6125.

11.8 Exercises

1. In what release of z/VM was guest LAN support added?
2. List three types of hardware network adapters supported by z/VM.
3. The two types of NICs supported by z/VM are QDIO and _____?
4. Name the two primary point-to-point network types supported by z/VM.
5. What is the difference between a layer 2 network and a layer 3 network?
6. What is the primary advantage of a VSWITCH over a guest LAN?
7. You have just added an OSA device to a guest. When you run QUERY NIC to verify it was added, nothing shows up. Why? Is there another command you can use to verify whether it was added?
8. You want to couple a NIC on your guest to a guest LAN that was defined in the normal manner. What command do you need to run to get access to the LAN?
9. You now want to couple a NIC on your guest to a VSWITCH. What command do you need to run to get access to the LAN?
10. What command can you use to view the access list for a VSWITCH?
11. If you wanted to temporarily add an OSA device to a guest, what command would you use?
12. Assume you wanted to add the same OSA device to the guest permanently. What steps would you need to perform?
13. What command would you use to view how much data has been sent or received by an OSA device?
14. Log on to a z/VM system as MAINT and examine the user directory. How many service virtual machines can you identify?

12

z/VM security

Protection against attempts to breach the security of a system and against inadvertently compromising the integrity of the system and data should always be kept in mind when using or administering a z/VM system. This chapter provides an overview of the security mechanisms available in z/VM and outlines a number of ways to preserve security and data integrity.

Objectives

After completing this chapter, you will be able to:

- Explain what an external security manager is
- Describe the login process flow
- Discuss privilege classes and the roles they define
- Understand how hardware tape encryption works
- Explain how to add a hardware cryptographic card to a z/VM guest

© Copyright IBM Corp. 2007. All rights reserved.

12.1 Introduction to z/VM security

Security and system integrity are paramount when developing and using a virtualization product such as z/VM that allows you to run multiple operating systems in a shared environment. People want to know that their data is safe and the computing environment that they have spent so much time and money building is secure from both malicious users and inadvertent mistakes. z/VM provides countless features designed to support these goals and in the following sections many of them are examined and explained.

For additional details about the security and integrity features of z/VM, refer to Chapter 12 in *CP Planning and Administration*, SC24-6083.

12.2 External security managers

z/VM is designed to be extremely extensible and modular, and these concepts extend to its own security model. While z/VM is very secure on its own, it also allows you to plug in an external security manager (ESM) such as IBM Resource Access Control Facility (RACF), which can provide additional levels of security and additional tools to manage it.

RACF allows you to identify and authenticate users, authorize users to access protected resources, control the means to access resources, as well as to log and audit these events. Resources can include minidisks, tape devices, terminals, shared user identifiers, virtual LANs, and other CP commands and resources.

ESMs such as RACF usually have their own database, for managing users and access control lists (ACLs), that separate from the base CP user directory. There are several useful IBM publications which document RACF and its functionality, including *RACF: General User's Guide*, SC28-1341, and *RACF: Security Administrator's Guide*, SC28-1340.

12.3 Directory management

Directory managers, such as the IBM directory management feature DirMaint, allow you to grant varying levels of administrative privileges to system administrators. Like CP, DirMaint breaks up its commands into different command classes that allow you to delegate administrative authority and grant varying levels of privileges to users. For example, user `steve` could have full privileges. In contrast, user `mike` may only be allowed to alter existing user

accounts or manage disk pools. For a detailed explanation of this feature, refer to Chapter 8, “Delegating Administrative Authority”, in *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6135.

12.4 User authentication and authorization

Typically in a production environment, the external security manager is configured so that it is the final determiner of whether a user is granted access to the system or to a resource. In other environments, the ESM is sometimes configured so that it will fall back on CP if it is unsure of whether a user should have access to a particular system or resource.

When a user attempts to log in to the system, CP passes the request to the ESM, which checks its database. If the credentials are correct, access is granted. Otherwise the request falls back to CP, which will check the user directory (whether its being managed by CP or by a directory management feature such as DirMaint). If CP is able to verify the user name and password, access is granted. If not, the login is denied. Figure 12-1 on page 378 shows a graphical representation of the process flow.

Resource access attempts operate in a very similar manner to logins. Resource access includes linking or attaching DASD or minidisks, coupling to a LAN, or accessing other system resources. It is important to remember that once you allow a resource to be managed by an ESM, all requests for that resource will be checked by the ESM; therefore, keeping the ACLs current is essential.

Note: It is possible to configure CP to not fall back, so that the grant/deny decision by the ESM is the final determiner. Although this could improve security, it can also lead to problems if the ESM malfunctions or is improperly configured.

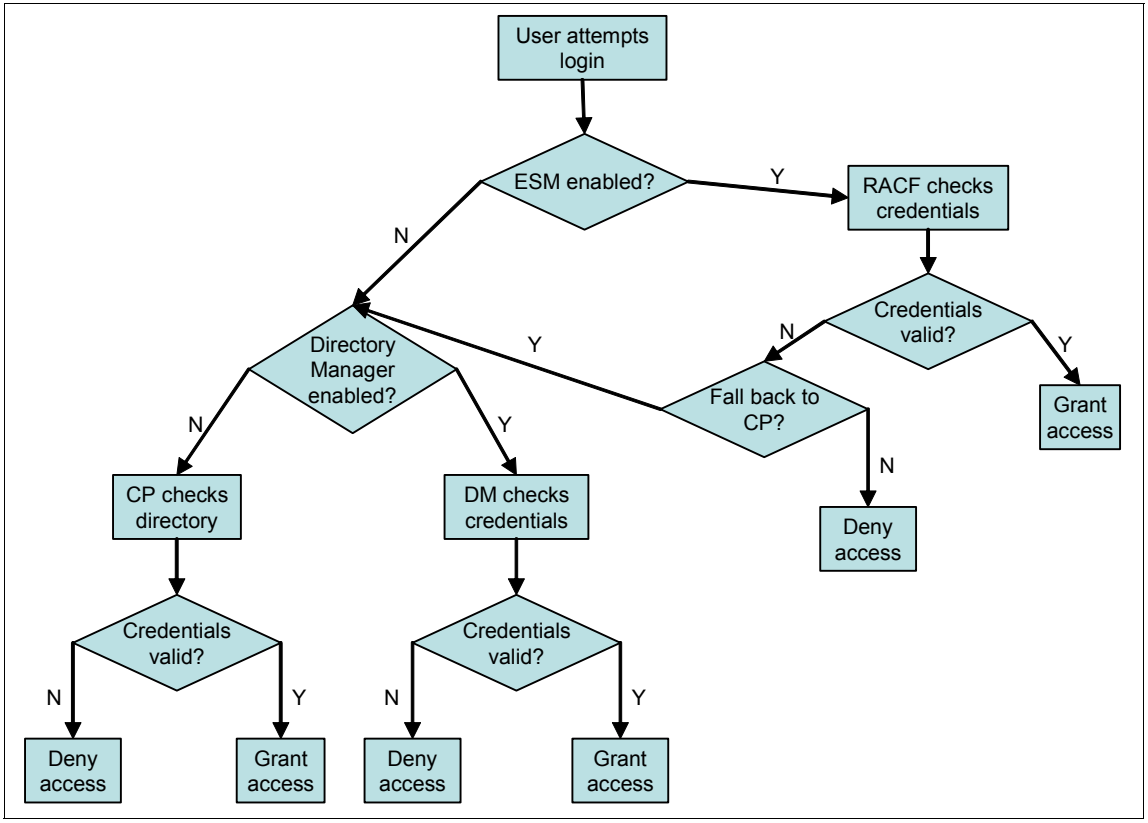


Figure 12-1 The z/VM login process

12.4.1 Privilege classes

As mentioned in 5.2.2, “Examining your virtual machine” on page 108, z/VM provides different classes of privileges to all system users. The various privilege classes that z/VM has define several different user roles, which are listed and explained in Table 12-1.

Table 12-1 CP command privilege classes

Class	User and function
A	System Operator: The Class A user controls the z/VM system. The System Operator is responsible for the availability of the z/VM system and its resources. Also, the System Operator controls system accounting, broadcast messages, virtual machine performance options, and other options that affect the overall performance of z/VM.

Class	User and function
B	System Resource Operator: The Class B user controls all the real resources of the z/VM system, except those controlled by the System Operator and the Spooling Operator.
C	System Programmer: The Class C user updates or changes system-wide parameters of the z/VM system.
D	Spooling Operator: The Class D user controls spool files and the system's real reader, printer, and punch equipment allocated to spooling use.
E	System Analyst: The Class E user examines and saves system operation data in specified z/VM storage areas.
F	Service Representative: The Class F user obtains, and examines in detail, data about input and output devices connected to the z/VM system.
G	General User: The Class G user controls functions associated with a particular virtual machine.
Any	Commands belonging to Class Any are available to any user, regardless of the privilege class. These commands are primarily used to gain access to, or relinquish access from, the z/VM system.
H	Reserved for IBM use.
I-Z 1-6	Classes I through Z and 1 -6 are reserved for redefinition through user class restructure (UCR) by each installation for its own use.

When defining users, it is important to only allow them the privilege classes that they absolutely need, in order to minimize the potential security exposure should the account be compromised. It is even more important when a guest operating system such as Linux will be running in the virtual machine, because now it is not only z/VM security that needs to be considered, but also the security of the guest operating system. Depending on the network services that are enabled on the Linux guest, it could be very easy to break in and then use tools like **vmcp** to issue commands to z/VM.

If there are specific commands that a guest requires from privilege classes A-F, it is possible to define your own custom privilege class and include those commands in it. The guest could then be given the custom privilege class, thus minimizing the potential damage that could be done maliciously or inadvertently by the user.

12.5 z/VM security features

The combination of System z hardware with the system integrity features of z/VM provides extremely high levels of security for guest operating systems. In the following sections we take a brief look at two particular areas of concern: processor and memory protection, and disk protection.

12.5.1 Processor and memory protection

Once concern when running many operating systems on a single piece of hardware is how secure data is when it is being used in a shared system such as z/VM. Because of the way z/VM dispatches guest operating systems, the contents of all hardware registers for the previously dispatched guest are saved to its control block before the next guest's registers are loaded. This prevents one guest from accessing the registers for another running operating system.

Additionally, CP and the System z hardware provide extensive controls that prevent one guest from accessing another guest's storage without approval. z/VM also helps prevent inadvertent memory access by providing each guest with its own virtual address space that has no meaning outside of that particular guest.

You can find additional details about the system integrity offered by z/VM in Chapter 12, "Security and Integrity in z/VM", in *CP Planning and Administration*, SC24-6083, as well as in 13.7, "Confidentiality and integrity on z/VM", in *Introduction to the New Mainframe: Security*, SG24-6776.

12.5.2 Disk protection

The primary protection for DASD volumes comes from the IOCDS that defines which devices are available to an LPAR—because a DASD cannot be added to z/VM or to a guest operating system (such as Linux) if it is not defined in the IOCDS.

Note: It is still possible to dynamically alter the IOCDS, but it is assumed that the necessary SE or HMC access has been restricted by the owner of the hardware and as such, is off-limits to the typical z/VM user.

For those DASD volumes that are made available to an LPAR running z/VM, the next layer of protection comes from the CP user directory or the external security manager, if one is active.

Link passwords

As stated in 5.4.4, “DASD (disk devices)” on page 126, the minidisk definitions in a user’s directory entry define how the disk can be linked by other guests and optionally, specifies a password that is required to link the disk. If another guest does not have the password, they cannot link and access the minidisk. A sample directory entry is shown in Example 12-1.

Example 12-1 Sample directory entry with link passwords

```
USER LINUX1 NOP4SSWD 512M 1G G
   IPL CMS PARM AUTOCR
   MACHINE ESA
   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
   MDISK 0191 3390 1886 5 530W02 MR ALL ALL ALL 1
   MDISK 0191 3390 1891 5 530W02 MR ALL HELLO HELLO 2
```

1 Specifies that anyone can link this minidisk in read, write, or multiple-write mode without a password.

2 Allows anyone to link this disk in read mode, but requires a password (HELLO) to link the disk in write or multiple-write mode.

Additional details about the MDISK directory statement can be found in Chapter 17, “Creating and Updating a User Directory”, in *Implementing WebSphere Business Integration Express for Item Synchronization*, SG24-6083.

Using an external security manager to protect DASD

The external security manager (ESM) can apply additional rules and security to DASD volumes, thus limiting who can use the them. The ESM could be configured with a set of rules that grants access to a few specific DASD volumes and denies access to everything else (whether it has been defined to the ESM or not). Alternatively, the ESM may allow free access to all but a few volumes that have been restricted by the security administrator.

How your specific environment and directory or ESM have been configured is wholly dependent on your organization’s security policies and the administrators who enforce the policies.

Some z/VM administrators like to make all of their DASD available to their guests in case they need to bring up a different operating system or link to a particular volume. It is important to remember that while this may make some tasks easier, it also could allow someone that has compromised one guest operating system access to all the rest by varying it online and changing the files. The solution to this problem is to either limit access to DASD, or to ensure that a comprehensive security policy has been defined via the ESM.

12.5.3 Tape security

In recent years the loss of personal information from both governments as well as corporations has led to a number of new laws relating to the handling of personal or sensitive information. One relatively new result of this legislation is the availability of drive-level tape encryption in enterprise storage products.

The IBM TS1120 tape subsystem offers a wide variety of encryption schemes that can be used from z/OS, Linux, AIX, and (as of Version 5 Release 3) z/VM.

In order for z/VM to provide system-managed tape encryption, a Java-based application called the Encryption Key Manager (EKM), which manages the encryption keys for the tape drive, must be installed and running in a guest operating system or another system (such as Linux) and operate on behalf of z/VM. This is due to the fact that z/VM itself does not have a Java interpreter. For complete information about this topic, refer to *IBM System Storage TS1120 Tape Encryption Planning, Implementation, and Usage Guide*, SG24-7320.

Using an encryption-capable tape device with Linux for System z and system-managed encryption is identical to a normal tape device, except that an extra parameter is used when attaching the tape drive in z/VM to the desired guest. The parameter specifies whether or not to use encryption and, optionally, an alias that refers to the encoding mechanism and encryption key that should be used. If no key alias is specified, the EKM uses its default keys. For example:

```
ATTACH E00 TO LINUX1 KEY
```

This command would attach the tape drive at device address E00 to the guest LINUX1. When it came time for the tape drive to encrypt the data, the EKM would provide the drive with its default encryption keys. Figure 12-2 on page 383 shows the high level sequence of events.

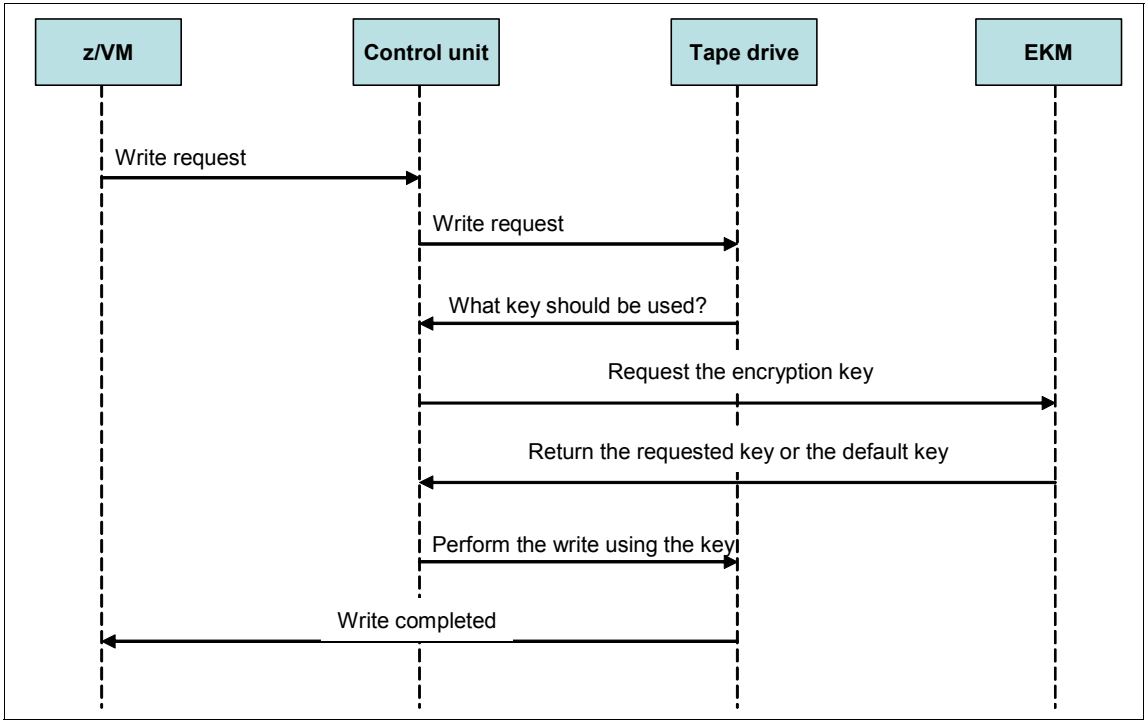


Figure 12-2 System-managed tape encryption sequence

12.6 Available cryptographic facilities

There are a number of hardware cryptographic features available for System z. IBM produces both separate hardware cryptographic accelerators and coprocessors, and integrates cryptographic assist processors into the general purpose CPs that are used by z/VM and other operating systems.

In zSeries z990 and newer models, IBM has integrated a cryptographic assist processor into the CP chip that helps it perform symmetric encryption functions faster. This feature is referred to as the CP Assist for Cryptographic Function (CPACF). Supported methods include Data Encryption Standard (DES), Triple DES, Advanced Encryption Standard (AES), Message Authentication Code (MAC) message authentication, and two version of the Secure Hash Algorithm (SHA-1 and SHA-256).

IBM also offers a hardware feature called the Crypto Express2 (CEX2), which is installed in an I/O cage in the mainframe and provides two Peripheral

Component Interconnect Extended (PCI-X) adapters that can be configured as coprocessors, accelerators, or as one coprocessor and one accelerator.

When the card operates in coprocessor (CEX2C) mode it can help to speed up asymmetric cryptographic functions, and it also supports several government standards for handling secure key cryptography. For more detailed information about secure key crypto, refer to the IBM white paper *Clear Key/Secure Key Primer*, WP100647[1].

The card can also operate in accelerator (CEX2A) mode, which helps perform modular arithmetic operations that are typically used by the Rivest-Shamir-Adelman (RSA) encryption algorithm.

Regardless of the type of card being used, z/VM has support for sharing the cards among its guest operating systems or dedicating the card to a specific guest.

To take advantage of these functions, a guest must have the APVIRT statement in its directory entry. This statement allows the guest to access the cryptographic features managed by z/VM. The alternative is to use the DEDICATE statement to give a z/VM guest sole access to one of the devices on the cryptographic card. No special access is needed to access the hardware instructions provided by the CPACF feature.

For additional information about IBM cryptographic offerings on newer mainframes, refer to *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070, and to *z9-109 Crypto and TKE V5 Update*, SG24-7123.

12.7 Best practices

As mentioned throughout the chapter, there are a number of practices to strongly consider implementing in a z/VM environment. Here we present a few particularly noteworthy practices.

1. Use an external security manager such as RACF. Consider restricting access to all resources and granting only the specific privileges that are needed by a guest.
2. When creating z/VM guests, assign them the smallest number of privilege classes possible instead of full (A-G) privileges. If there are specific commands needed by a guest, consider adding them to a custom privilege class and then granting the guest access to that priviledge class. Thus, if a

[1] This paper is available on the Web at: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100647

guest is not allowed to use a command, you do not need to be concerned about that command being abused or compromised in case of an intruder.

3. If you have a directory manager such as DirMaint enabled, consider granting varying levels of administrative privileges instead of using an “all-or-nothing” approach.
4. Consider enabling logging and audit records in RACF and DirMaint. Review the logs and records on a regular basis to look for security threats or situations where security has been compromised.
5. If you have a tape subsystem that supports encryption, *use* it. It is very easy to configure and could avoid major issues if tapes are deliberately or inadvertently lost.
6. If you are going to be performing asymmetric encryption in guest operating systems, consider granting that guest shared access to a CEX2C or CEX2A card via APVIRT to speed up the process and reduce the load on any CPs or IFLs.

12.8 Exercises

1. What kinds of resources can an external security manager be used to secure?
2. How many privilege classes are defined by default on a z/VM system?
3. What user role does the class C privilege define?
4. What type of encryption and hashing methods are supported by the CPACF feature of newer mainframes?
5. What is the name of the optional System z cryptographic hardware feature, and what two modes does it support?
6. What directory statement must be added to a guest in order for it to share access to a hardware cryptographic card?

13

Guest operating systems

As previously discussed, the major use of z/VM is to provide a software hypervisor on the mainframe. Therefore, it is important to know what operating systems can run as a guest of z/VM. This chapter explains the guest support under z/VM.

Objectives

After completing this chapter, you will be able to:

- Describe guest support under z/VM
- Discuss the advantages of running guest operating systems under z/VM
- List which operating systems are supported as guests under z/VM

© Copyright IBM Corp. 2007. All rights reserved.

13.1 Guest support

Guest support in z/VM allows you to run multiple copies of production operating systems. You can use guest support in z/VM to develop, test, manage and migrate operating systems that run on System z alongside your production systems.

13.1.1 Guest simulation

Guest support in z/VM is capable of hosting as many virtual machines as you might need, each simulating a real machine. They could all be different, some z/OS, some z/VSE, and some Linux, or they could all be the same. This environment proves the following advantages:

- You can create an exact replica of your production system. On this replica, you can test your new programs, services, and procedures.

  This is a relatively inexpensive way to have your own test system (or as many test systems as you would like). It is also a safe way to test a new function because your real production system, along with its applications and data, is protected from any damage that the new function might cause if you tested it on the host system.
- You can have a flexible migration system. This eliminates the inconvenience to users which migration usually causes.
- You can create a multiple production system environment.
- Guest systems may see performance improvements by exploiting z/VM features. For example, both virtual disk in storage and minidisk cache allow guests to avoid real I/O by using data in storage and caching techniques.
- Guest systems can share devices such as channels, printers, and direct access storage device (DASD), which z/VM efficiently manages. A useful example is z/VM minidisk support, which allows one real disk to function as if it were several smaller disks (such as multiple IPLable minidisks).
- z/VM emulation of some devices (for example, switches and LANS) may help your installation avoid having to purchase real ones.
- You can build guest virtual machines to simulate systems at your organization's other sites. Therefore, z/VM can become a valuable disaster recovery resource.
- Advanced systems management, administration, and accounting tools are provided.
- If you use IFLs, it is possible to increase the throughput of your system without incurring additional software charges.

Note: For useful information about the z/VM virtual machine operations, see *z/VM V5R2.0 Virtual Machine Operation,* SC24-6128.

13.2 Supported guest operating systems

All of the supported mainframe operating systems can run under z/VM, including z/VM itself. For a list of supported mainframe operating systems and the type of support z/VM offers, along with other useful information, refer to notes in *z/VM General Information*, GC24-6095.

Tip: Also refer to *z/VM General Information*, GC24-6095, for a list all of supported hardware that can be used with any release of z/VM.

13.2.1 Linux as a guest operating system

Typically, in a Linux environment, a single application might run on each physical server. The networking demands and complexity of running these systems can be greatly reduced by transferring them to run as guests under z/VM.

It is possible to run hundreds of Linux servers on one physical machine. This process is normally referred to as *server consolidation*. Here we list some of the advantages of running Linux under z/VM:

- Sharing of resources:
  - CPU
  - I/O
  - Storage (using Linux as an NSS)
  - Maintainability (multiple guests sharing the same kernel and data)
- Reliability
- Exploitation of the unique features of z/Architecture
- Centralized management
- Greater security using cryptographic coprocessors
- Cost savings on software licensing
- Less energy and cooling required

Since Linux became available for z/architecture, a number of applications have been written in some Linux distributions that increase the synergy between Linux and z/VM. These include device drivers to access shared segments and CMS minidisks, the ability to issue CP commands, and performance reporting tools.

You can refer to the following publications for detailed explanations about how to run Linux as a guest under z/VM.

- *z/VM: Getting Started with Linux on System z*, SC24-6096
- *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES9*, SG24-6695
- *z/VM and Linux on IBM System z: The Virtualization Cookbook for RHEL4*, SG24-7272

13.2.2 z/OS as a guest operating system

z/OS can run as a guest operating system under z/VM. The older versions or predecessors of z/OS (such as OS/390®) may not be supported on the newer mainframes. Such operating systems may run in a z/VM virtual machine.

Using the simulation capabilities of z/VM, it makes it easier to test new hardware and software features on z/OS by running it as a guest under z/VM. Virtual machines can be duplicated very quickly using the cloning facilities available in z/VM. So this makes test and development faster.

When z/OS systems are running as guests of z/VM, it is easy to share devices such as tape drives among multiple systems. For example, the **MULTIUSER** operand for the **DEDICATE** control statement in the z/VM USER Directory is one way to serially share a real tape drive with multiple guest systems. z/OS is often hosted under z/VM at disaster recovery (DR) sites where the virtualization of device numbering means that few, if any, changes need to be made to the z/OS guests.

Note: For step-by-step instructions about how to run z/OS as a guest under z/VM, see *IBM TotalStorage Peer-to-Peer Virtual Tape Server Planning and Implementation Guide*, SG24-6115.

z/OS Parallel Sysplex under z/VM

In addition to being able to run z/OS systems as guests under z/VM, you can simulate advanced *Coupling Facility* (CF) and *message facility* (MF) functions.

z/VM guest coupling provides for the simulation of one or more complete Parallel Sysplexes within a single z/VM system image. The intent is to provide a preproduction testing platform for a coupled-system installation, and for training people.

The z/VM simulated environment is not intended for production use because its single points of failure negate the intent of the Parallel Sysplex® environment.

Another reason for running a z/OS Parallel Sysplex under z/VM could be to perform disaster recovery testing, which may be a very cost-effective solution.

The z/VM guest coupling simulation support consists of three components: Coupling Facility service machines, coupled guests, and simulated message facility. Figure 13-1 illustrates the simulated Coupling Facility environment on z/VM.

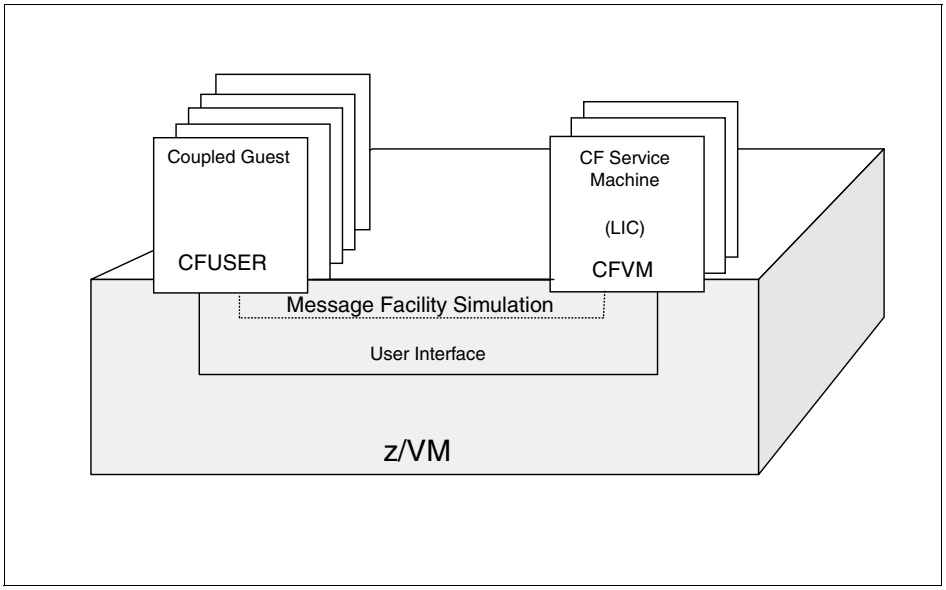


Figure 13-1 z/VM Coupling Facility support

For detailed information about running z/OS under z/VM and setting up a Parallel Sysplex on z/VM, refer to *IBM TotalStorage Peer-to-Peer Virtual Tape Server Planning and Implementation Guide*, SG24-6115, and *Using z/VM for Test and Development Environments: A Roundup*, SG24-7355.

13.2.3 z/VSE as a guest operating system

z/VSE has run as a guest operating system under VM for many years, using facilities made available by CP. Here we list some of the advantages of running z/VSE under z/VM:

- Shared I/O units are possible.
- Communication between guest systems using virtual devices.
- CP can be used to do paging.
- Exploitation of expanded storage.
- Connectors to expand capability and accessibility using applications on Linux.

A set of functions known as the VM/VSE Interface routines are shipped with z/VSE, and they provide an interface with multiple z/VSE guests from CMS. The VM/VSE Interface routines is a set of VSE phases and CMS modules.

The VM/VSE Interface routines are distributed in the z/VSE library IJSYSRS.SYSLIB. These can be punched using a supplied job SKVMVSE from the z/VSE library and installed on a CMS minidisk.

The following functions are supported by the VM/VSE Interface:

- Have none, some, or all messages from a job or from the system echoed to a specified owner (CMS user ID).
- Reply to messages resulting from the execution of a job. The job must have a unique job owner ID (CMS user ID).
- Submit jobs from a CMS terminal to a z/VSE guest system.
- Issue VSE commands (including REDISPLAY commands) to a z/VSE guest system, and have the resulting messages echoed to the CMS user.
- Issue CP commands for execution in the virtual machine, and have the resulting CP messages routed to the CMS job owner.

For more information about running z/VSE as guest under z/VM, see *z/VM V5R2.0 Running Guest Operating Systems*, SC24-6115.

13.2.4 z/VM as a guest operating system

Running z/VM itself as a guest system offers opportunities for more flexible system management. In this environment, you can:

- Install and test new releases of z/VM
- Test new application programs
- Test new maintenance procedures and modifications
- Train operators and system programmers

A significant advantage provided by running a second level z/VM system is the ability to generate a new system without disturbing normal production activity. System programmers can log on their own virtual machines and go through the generation steps at their own pace, while the rest of the installation uses the real z/VM system undisturbed. After the system is tested it can be placed online, thereby replacing the previous version with minimal disruption to production activity.

For more information about running z/VM as guest under z/VM, refer to *z/VM V5R2.0 Running Guest Operating Systems*, SC24-6115.

13.3 Exercises

To help test your understanding of the material in this chapter, answer the following questions:

1. What is the difference in the memory utilization of the physical mainframe system when operating systems are running in LPAR mode and when they are running as a guest of z/VM?
2. True or False - At many sites, it is typical to see several hundreds of Linux guests hosted on a single z/VM LPAR.
3. True or False - To run a Parallel Sysplex under z/VM, special hardware is required on the physical mainframe system.
4. When the z/VSE Operating System is running as a guest under z/VM, what is the benefit in terms of “paging” requirements?
5. Is it possible to install a third level z/VM system?

A

Enhancements in z/VM Version 5, Release 3

This textbook was written using examples from z/VM version 5, Release 3. Some students may be on earlier releases. This appendix identifies how z/VM has been changed and enhanced.

In this section we describe the product changes grouped in the following sections:

- Enhanced scalability and constraint relief
- Virtualization technology and Linux enablement
- Network virtualization
- Security
- Systems management
- Installation, service, and packaging changes
- Additional changes

© Copyright IBM Corp. 2007. All rights reserved.

Enhanced scalability and constraint relief

This section describes enhancements that can help support increased workloads on z/VM.

Support for up to 256 GB of real memory

Changes to page table allocation in z/VM V5.3 allow z/VM images to support significantly more real memory (storage) than the prior limit of 128 GB, as well as more virtual memory, up to 256 GB of real memory and up to 8 TB of total virtual memory in use by guests.

The actual amount of usable real and virtual memory is dependent on the amount of real memory in the z/VM logical partition, the hardware server model, firmware level, and configuration, and the number of guests and their workload characteristics. This can benefit customers with large amounts of real storage, and may help reduce or eliminate the need to spread large workloads across multiple z/VM images.

Enhancements to the management of contiguous frames may also reduce storage management overhead and improve performance. Better z/VM management of real storage can benefit most customers who experience storage constraints, regardless of the amount of central storage configured for z/VM use.

Up to 32 real processors in a single z/VM image

z/VM V5.3 can support customer growth by allowing up to 32 real processors in a single z/VM image on an IBM System z server, an increase of 33% from the prior maximum of 24. The particular workload will influence the efficiency with which a z/VM system can use large numbers of processors. Generally, z/VM overhead is expected to be lower with fewer, more CPU-intensive guests than with many lightly loaded guests.

Enhanced memory management for Linux guests

z/VM V5.3 adds support for the Collaborative Memory Management Assist (CMMA) on the IBM z9 EC and z9 BC processors. This z/VM support, in conjunction with CMMA exploitation in guest operating systems such as Linux on System z, allows the z/VM V5.3 Control Program (CP) host and its guests to communicate attributes for specific 4 KB blocks of guest memory.

This exchange of information can allow both the z/VM host and its guests to optimize their use and management of memory in the following ways:

- CP knows when a Linux application releases storage and can select those pages for removal at a higher priority or reclaim the page frames without the overhead of paging-out their data content to expanded storage or disk.
- CP recognizes clean disk cache pages, the contents of which Linux is able to reconstruct, allowing CP to bypass paging-out the data contents when reclaiming the backing frames for these pages. If Linux or its application subsequently tries to refer to the discarded page, Linux is notified that the page has been discarded and can reread the contents from disk or otherwise reconstruct them.
- The guest further benefits from the Host Page-Management Assist (HPMA) announced in the Hardware Announcement dated July 27, 2005. In conjunction with CMMA, HPMA allows the machine to supply fresh backing page frames for guest memory when the guest reuses a previously discarded page, thereby eliminating the need for the z/VM hypervisor to intercept and resolve these host page faults.

z/VM 5.3 is the delivery vehicle for providing enhanced memory management support on z/VM. This satisfies the statement of direction made in the Software Announcement dated July 27, 2005.

Refer to the Preventive Service Planning (PSP) bucket for your z9 EC or z9 BC server for required updates. To avoid system outages, required minimum MCL levels must be applied prior to IPLing z/VM V5.3 and exploiting new functions.

IBM is working with its Linux distribution partners to provide CMMA exploitation in future Linux on System z distributions or service updates.

Enhanced memory utilization using VMRM between z/VM and Linux guests

Virtual Machine Resource Manager (VMRM) assists in managing memory contention in the z/VM system. Based on CP monitor data, the z/VM V5.3 VMRM detects when memory is constrained and notifies the Linux guests. These guests can then take action to adjust their memory consumption to help relieve the memory constraint, such as by releasing pages containing the least recently referenced file cache data. The installation controls which guests are notified.

HyperPAV support for IBM System Storage DS8000

z/VM V5.3 supports the Hyper Parallel Access Volume (HyperPAV) function optionally provided by the IBM System Storage DS8000 disk storage systems. HyperPAV support complements the existing basic PAV support in z/VM V5.2, for applicable supporting disk storage systems.

The HyperPAV function potentially reduces the number of alias-device addresses needed for parallel I/O operations, because HyperPAVs are dynamically bound to a base device for each I/O operation instead of being bound statically like basic PAVs. z/VM provides support of HyperPAV volumes as linkable minidisks for guest operating systems, such as z/OS, that exploit the HyperPAV architecture. This support is also designed to transparently provide the potential benefits of HyperPAV volumes for minidisks owned or shared by guests that do not specifically exploit HyperPAV volumes, such as Linux and CMS.

Enhanced FlashCopy support

z/VM V5.3 support for the FlashCopy V2 feature of IBM System Storage disk storage devices has been enhanced to simplify the tasks required to automate backups. This includes the capabilities to:

- Specify multiple target minidisks
  - The CP FLASHCOPY command can now accept up to 12 target minidisks to be copied.
- Determine the status of FlashCopy requests
  - The new CP QUERY Virtual FLASHCOPY command allows the user to query the number of Flashcopy relationships active for one or more of their virtual DASD.
- Exploit hardware asynchronous cache destage and discard
  - This is designed to eliminate delayed hardware response messages and provides quicker responses to the CP FLASHCOPY command. This makes the FlashCopy appear synchronous to the virtual machine, and may simplify automating processes that exploit this technology.

In addition, z/VM has reduced the number of FlashCopy hardware-related error conditions that can be reflected to the guest for the z/VM FLASHCOPY command. z/VM will attempt to re-drive the I/O on some error conditions before reflecting the command response back to the guest.

Support for the IBM System Storage SAN Volume Controller

The IBM System Storage SAN Volume Controller can transform the traditional relationship between a host and its volume manager. The SAN Volume Controller can be attached to the storage network to provide a virtualized pool of storage shared by all hosts. The physical disks are discovered and organized into virtual disks that are constructed from any portion or combination of physical disks chosen by the storage administrator. These virtual disks are the storage media presented to the host systems.

The SAN Volume Controller is designed to:

- Combine storage capacity from multiple vendors into a single reservoir of capacity that can be managed from a central point
- Help increase storage utilization by providing host applications with more flexible access to capacity
- Help improve productivity of storage administrators by enabling management of combined storage volumes from a single interface
- Support improved application availability by insulating host applications from changes to the physical storage infrastructure
- Enable a tiered storage environment in which the cost of storage can be better matched to the value of the data
- Support advanced copy services from higher-cost to lower-cost devices and across subsystems from multiple vendors.

With the SAN Volume Controller, data can be moved from one physical disk to another, or even from one vendor's disk to another, without affecting the virtual disks seen by the host systems. IT managers can plan for physical changes in the storage infrastructure more effectively, typically without interruption to business applications.

IBM System Storage SAN Volume Controller Storage Engine 2145

The IBM System Storage SAN Volume Controller Storage Engine is the hardware component of the IBM System Storage SAN Volume Controller solution. The components of the SAN Volume Controller include highly specialized software, storage engines installed in pairs, a master console, and uninterruptible power supplies (UPSs).

The SAN Volume Controller hardware is designed to combine servers into a cluster designed to support high availability. Each of the servers in the cluster is populated with 8 GB of high-speed memory that serves as the cluster cache. Each also includes a 4-Gbps host bus adapter (HBA), designed to allow the SAN

Volume Controller to connect and operate at the 4-Gbps SAN speed. The SAN Volume Controller storage engines are always installed in pairs for redundancy. Currently installed Model 8F2 engines can be upgraded by a 4-Gbps HBA adapter feature.

The uninterruptible power supply (UPS) is designed to help protect against data loss resulting from a loss of electrical power.

A separate server is the master console for SAN Volume Controller storage engine management. The master console software is preloaded on the master console, and it provides the user interface to the SAN Volume Controller. A software-only version of the master console, which can be loaded onto a server that meets certain minimum configuration requirements, is available as an option. The master console can, using a virtual private network (VPN), provide a remote support interface. This can help reduce the requirement for onsite support.

IBM System Storage SAN Volume Controller Software V4.1

IBM System Storage SAN Volume Controller V4.1 introduces the optional advanced copy services capability of Global Mirror to support distance replication solutions. Building on the original Metro Mirror capabilities of SAN Volume Controller software, Global Mirror's asynchronous peer-to-peer remote-copy function can help provide the critically important ability to maintain a minimally delayed copy of data at a distance sufficient to survive metropolitan or regional disasters.

This software runs on the new IBM 2145-8F4 storage engines, with 4-Gbps Fibre Channel HBA attachment capability to the SAN fabric, as well as on previously released SAN Volume Controller storage engines.

SAN Volume Controller V4.1 continues to be designed to improve the customer's total storage management environment with key support enhancements, which include:

- The ability to upgrade individual SAN Volume Controller storage engines non-disruptively within existing I/O groups
- New reporting facilities for tracking virtual disk performance, cache usage, port utilization, and CPU utilization
- New audit log facility that records which user performed each configuration action
- Access control for hosts on a per-port basis

z/VM support for the 2145 SAN Volume Controller

z/VM and its guest operating systems are designed to access SCSI FCP storage capacity from multiple vendors as a single reservoir of capacity that can be managed from a central point. z/VM supports the SAN Volume Controller through the generic SCSI device driver of z/VM. The SAN Volume Controller handles the device-specific requirements for whatever collection of different storage devices a customer has attached to the SAN Volume Controller.

z/VM support for the SAN Volume Controller allows the z/VM control program (CP) and guest operating systems that use SCSI devices (such as Linux on System z and z/VSE, as well as z/VM itself) to access IBM System Storage disk subsystems, including the DS4000™ series, as well as disk subsystems from other manufacturers supported by the SAN Volume Controller.

This support adds 2145 as an operand on the **EDEVICE** configuration statement, as well as on the **SET EDEVICE** and **QUERY EDEVICE** commands.

The SAN Volume Controller can be used to provide SCSI devices as emulated FBA devices for use by CP and guest operating systems. This support is planned to be available in z/VM V5.3 and, with the PTF for APAR VM64128, in z/VM V5.2.

Use of SCSI devices accessed through the SAN Volume Controller by dedicated FCP subchannels is available to guest operating systems in any release of z/VM V5 without the application of any PTFs.

For Linux on System z guests, SAN Volume Controller V4.1 is supported for SLES 8, SLES 9, and RHEL 4.

Virtualization technology and Linux enablement

This section describes extensions to z/VM virtualization technology in support of Linux, z/OS, and other guests.

Support for IBM System z specialty engines (processors)

Integrated Facility for Linux (IFL) processors are dedicated to Linux workloads. IFLs enable you to purchase additional processing capacity exclusively for Linux workloads, without affecting the MSU rating or the IBM System z model designation. This means that acquiring an IFL will not necessarily increase charges for IBM System z software running on general-purpose (standard)

processors in the server. IFLs were first introduced in the Software Announcement dated May 29, 2001.

System z Application Assist Processors (zAAPs) are attractively priced specialized processors that provide an economical Java execution environment under z/OS and z/OS.e on the System z platform. zAAPs were announced in the Hardware Announcement dated April 7, 2004.

System z9 Integrated Information Processors (zIIPs) are the latest specialty processors, designed to help improve resource optimization and lower the cost for eligible workloads. z/OS and z/OS.e exploit zIIPs to offload software system overhead from standard Central Processors (CPs). This includes certain DB2 processing, enhancing the role of the mainframe as the data hub of the enterprise. zIIPs were announced in the Hardware Announcements dated April 27, 2006.

z/VM V5.3 is designed to provide new guest support for zAAPs and zIIPs and includes the following enhancements.

Simulation support

z/VM can simulate specialty processors for guest virtual machines by dispatching the virtual specialty processors on real CPs. The use of simulated specialty processors eliminates the cost associated with purchasing and installing new real specialty-processor hardware.

Simulating specialty processors provides a test platform for z/VM guests to exploit mixed-processor configurations. This allows users to assess the operational and CPU utilization implications of configuring a z/OS system with zIIP or zAAP processors without requiring the real specialty processor hardware. This simulation also supports the continuing role of z/VM as a disaster-recovery platform, because a virtual configuration can be defined to match the real hardware configuration even when real zIIP or zAAP processors are not available on the recovery system.

z/VM simulates specialty processors using real CPs if the underlying hardware is capable of supporting the real specialty processor. zIIPs can be simulated only on System z9 (z9 EC and z9 BC) servers. zAAPs can be simulated only on z9 EC, z9 BC, z990, and z890 servers.

Virtualization support

z/VM can create virtual specialty processors for virtual machines by dispatching the virtual processors on corresponding specialty processors of the same type in the real configuration. Guest support for zAAPs and zIIPs may help improve your total cost of ownership by allowing available zAAP and zIIP capacity not being used by z/OS LPARs to be allocated to a z/VM LPAR hosting z/OS guests

running Java and DB2 workloads. zAAPs and zIIPs cost less than standard CPs, so this support might enable you to avoid purchasing additional CPs, thereby helping to reduce your costs both for additional hardware and for software licensing fees.

Enhanced virtual switch and guest LAN usability

z/VM V5.3 provides the following usability enhancements for the virtual switch and guest LAN environments.

Enhanced ease-of-use for Virtual LAN (VLAN) and promiscuous mode configuration changes

Changes to the authorized VLAN ID (VID) set and to promiscuous mode authorization are now effective immediately instead of requiring a revoke, a grant, and an uncouple/couple in order for the changes to take effect.

New capability to configure a native VLAN ID

This support provides the ability to specify a native VLAN identifier for untagged traffic and a default VLAN identifier for guest ports. The DEFINE VSWITCH command now supports the specification of a native VLAN identifier.

New virtual NIC monitor domain

Existing counts maintained for the virtual NIC, such as inbound packets, outbound bytes, and frame counts per MAC/VLAN, are now included in records in a new Virtual Network monitor domain. These new monitor records provide data for a virtual NIC that is coupled to any guest LAN or VSWITCH.

MIDAWs for guests

z/VM V5.3 supports guest use of Modified Indirect Data Address Words (MIDAWs), which is a hardware feature available on the IBM System z9. MIDAWs can allow more flexibility and performance in certain channel programs as an alternative to data-chained channel-command words (CCWs).

MIDAWs accommodate noncontiguous data areas that cannot be handled by the predecessor indirect-data-address words (IDAWs). z/VM support for guest use of MIDAWs can allow operating systems such as z/OS to use this new aspect of z/Architecture without regard to whether the operating systems are running in a logical partition or a virtual machine. This allows guest operating systems to exercise their code-paths just as they would on the real machine during, for example, preproduction testing of z/OS systems.

Likewise, the provision of the function in a virtual machine allows guest operating systems to benefit from the real machine's added-value function just as though the guests were running directly on the machine.

Guest ASCII console support

The system ASCII console is a facility that comes with all System z models and is presented by the Hardware Management Console (HMC). z/VM V5.3 provides guest access to the system ASCII console.

By dedicating the system ASCII console to a Linux guest, customers can facilitate recovery of the guest during an emergency situation, using an environment that provides tools (such as vi and emacs) that are familiar to Linux support staff. This can be particularly useful when normal network access to a guest operating system is not available. The system ASCII console (and hence the guest ASCII console) supports a VT220 data stream.

This function can help lower system costs by helping to reduce the need to provide alternative facilities, such as duplicate network resources, to achieve desired guest-recoverability characteristics. Because this function provides guest access to the one system ASCII console by one guest at a time, use of the console can be transferred from guest to guest as required.

Enhanced SCSI support

z/VM V5.3 provides additional enhancements for Small Computer System Interface (SCSI) disk support for Linux users, including:

- Point-to-Point Fibre Channel links, which may provide a lower-cost installation than the current requirement for a Fibre Channel switched fabric
- Dynamically determined preferred paths for emulated FBA devices (EDEVICEs) on SCSI disks in an IBM System Storage DS6000, instead of the current need to specify which paths are preferred in a SET EDEVICE command or an EDEVICE configuration file statement
- Faster formatting of EDEVICEs on SCSI disks in an IBM Enterprise Storage Server (ESS) or IBM System Storage DS8000
- Display of additional SCSI device characteristics when using the QUERY EDEVICE DETAILS command
- Checking for erroneous mapping of multiple EDEVICE definitions onto the same SCSI disk when bringing emulated disks online

Network virtualization

This section describes enhancements to z/VM network virtualization.

Improved virtual network management

z/VM V5.3 helps network administrators manage virtual network performance, find and solve virtual network problems, and plan virtual network growth. z/VM V5.3 establishes a method for providing Simple Network Management Protocol (SNMP) data for virtual networking devices.

Specifically, it provides an SNMP subagent that runs in a separate virtual machine from the SNMP agent and extends the functionality of the agent by supporting a specific set of Management Information Base (MIB) variables. A preconfigured subagent and exit routine are provided in z/VM V5.3 to supply bridge Management Information Base (BRIDGE-MIB) data, as documented in RFC 1493, for the z/VM virtual switch.

This subagent, through the use of a Network Management System client, can acquire BRIDGE-MIB data for the z/VM virtual switch. In addition, this support provides a programming interface to obtain information about virtual networks.

Enhanced failover support for IPv4 and IPv6 devices

Failover support for Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) devices has been improved in z/VM V5.3. When the z/VM TCP/IP stack has two (or more) Queued Direct Input/Output (QDIO) or LAN Channel Station (LCS) Ethernet devices on the same network and one device is stopped or fails, another device takes over responsibility for traffic destined for the failing device (or any devices the failing device had previously taken over). This failover support includes OSA-Express devices (in QDIO Ethernet or LCS Ethernet mode), Virtual IP Addresses (VIPAs), and addresses for which PROXYARP services are being provided through a takeover-eligible device.

In addition to the basic failover support, one takeover-eligible device on that network will be responsible for informing other nodes on that network which hardware (MAC) address should be used to reach VIPA addresses on the TCP/IP stack, both when the stack initializes and when an IP takeover event occurs.

VIPA support for IPv6

Virtual IP Address support in the TCP/IP stack has been extended in z/VM V5.3 to support IPv6 addresses. It is now possible to enable and configure a virtual device for IPv6, as well as to associate real IPv6-capable network adapters with a specific IPv6 virtual link for determining the source address used in outgoing packets. Support for VIPA is designed to improve the capability of the TCP/IP stack to maintain connections in the event that a real network device fails.

Support for OSA-Express2 IEEE 802.3 and link aggregation

Aggregating links with a partner switch box allows multiple OSA-Express2 adapters to be deployed in transporting data between the switches. This configuration provides the benefits of increased bandwidth and near-seamless recovery of a failed link within the aggregated group.

Security

This section describes enhancements to the security characteristics of z/VM.

Delivery of LDAP server and client

z/VM V5.3 introduces new user authentication, authorization, and auditing capabilities with the inclusion of a Lightweight Directory Access Protocol (LDAP) server and associated client utilities. The z/VM LDAP server has been adapted from the IBM Tivoli Directory Server for z/OS, delivered in z/OS V1.8. Executing in a CMS virtual machine, LDAP is integrated in the base of z/VM V5.3 as a subcomponent of TCP/IP. The z/VM LDAP server provides:

- Multiple concurrent database instances (referred to as “backends”)
- Interoperability with LDAP Version 2 or Version 3 protocol-capable clients
- LDAP V2 and V3 protocol support
- Native authentication using Challenge-Response Authentication Method (CRAM-MD5), DIGEST-MD5 authentication, and simple (non-encrypted) authentication
- Root DSE information master/slave and peer-to-peer replication
- The ability to refer clients to additional directory servers
- The capability to create an alias entry in the directory to point to another entry in the directory
- Access controls on directory information
- Change logging
- Schema publication and update

- SSL communication (SSL V3 and TLS V1)
- Client and server authentication using SSL/TLS

The LDAP client utilities provide a way to add, modify, search, and delete entries in any server that accepts LDAP protocol requests.

The new RACF Security Server for z/VM feature, available with z/VM V5.3, has also been updated to interoperate with the new z/VM LDAP server.

Enhanced system security with longer passwords

Working together, z/VM V5.3 and the RACF Security Server for z/VM FL530 feature support the use of passwords that are longer than eight characters, called password phrases (also known as *passphrases*). A password phrase may contain mixed-case letters, numbers, blanks, and special characters, allowing for an exponentially greater number of possible combinations of characters than traditional passwords.

To utilize password phrases, an external security manager (ESM) that supports password phrases, such as RACF, is required. To ease migration from passwords to password phrases, the RACF Security Server for z/VM continues to support traditional 8-character passwords.

A new callable services library (CSL) routine, DMSPASS, allows authorized CMS applications to authenticate passwords or password phrases. The z/VM LOGON command, the z/VM TCP/IP File Transfer Protocol (FTP), Systems Management API, Remote Execution Protocol (REXEC), and Internet Message Access Protocol (IMAP) servers, and the Performance Toolkit for VM have been updated to support password phrases.

For environments in which password phrases cannot be used, but where additional password complexity is required, the RACF Security Server for z/VM also provides support for mixed-case 8-character passwords.

Support for password phrases and mixed-case passwords enables a z/VM system to meet the enterprise password requirements imposed by many companies, governments, and institutions.

Conformance with industry standards

z/VM V5.3 adds Secure Sockets Layer/Transport Layer Security (SSL/TLS) support for industry-standard secure FTP (RFC 4217), Telnet (draft specification #6), and SMTP (RFC 3207) sessions. This support includes new socket APIs to permit a Pascal or Assembler client or server application to control the

acceptance and establishment of TCP sessions that are encrypted with SSL/TLS.

Data transmission on a connection can now begin in clear text and at some later point be made available in secure text, thus helping to reduce the need to dedicate a separate port for secure connections.

In order to enable enforcement of enterprise requirements for strong encryption on network connections (128 bits or higher), the z/VM SSL server has been enhanced to more easily allow weak cipher suites to be excluded.

SSL server enhancements

Previous releases of z/VM provided Red Hat Package Manager (RPM) packages for various Linux distributions. z/VM V5.3 supports:

- Novell SUSE Linux Enterprise Server (SLES) 9 Service Pack 3 (64-bit)
- Novell SUSE Linux Enterprise Server (SLES) 9 Service Pack 3 (31-bit)
- Red Hat Enterprise Linux (RHEL) AS 4 Update 4 (64-bit)
- Red Hat Enterprise Linux (RHEL) AS 4 Update 4 (31-bit)

The z/VM SSL server has been enhanced to allow the host Linux guest system to remain active after a critical error is encountered during server operations.

Also, the SSLADMIN command has been enhanced to:

- Allow the specification of the number of days that a self-signed certificate is valid
- Improve the management of the SSL server LOG files, by providing the ability to:
  - Maintain log information in a file named other than SSLADMIN LOG
  - Specify a maximum size to be established for the SSL server log
  - Purge log information accumulated by the SSL server

Tape data protection with support for encryption

z/VM now supports drive-based data encryption with the IBM System Storage TS1120 Tape Drive (machine type 3592, model E05). The TS1120 encryption capability and its subsystem-integration support provide a flexible tape-data-encryption solution that provides data encryption and key management across a variety of environments with a single point of control for all encryption keys. Most importantly, this solution can help protect data on tape in a cost-effective way.

Encryption of tapes by z/VM itself requires that the IBM Encryption Key Manager be running on another operating system, using an out-of-band (such as TCP/IP) connection to the tape control unit.

z/VM native support includes encryption for DDR and SPXTAPE, as well as transparent support for guests that do not provide for their own encryption (for example, Linux and CMS).

z/VM also enables encryption of tapes by guests (such as z/OS) that have the ability to control the tape-encryption facilities themselves and to optionally run the Encryption Key Manager. Key management for such guests can use either an out-of-band or an in-band (such as an ESCON or FICON channel) connection between the Encryption Key Manager and the tape control unit.

With the PTF for APAR VM64063 for z/VM V5.1 and V5.2, only the Encryption Key Manager default keys are supported for use by z/VM and by guests that do not provide for their own encryption. z/VM V5.3 expands this support to allow any key label to be used, with key labels being accessible through a key alias that is defined to z/VM.

DFSMS/VM FL221 with the PTF for APAR VM64062 supports locating encryption-capable 3592 tape drives in an Enterprise Automated Tape Library. This DFSMS/VM support provides tape-encryption capabilities for a z/VSE guest running on z/VM.

For additional information on the IBM System Storage TS1120 Tape Drive encryption support, refer to the Hardware Announcement dated August 29, 2006.

Systems management

This section describes z/VM systems management improvements that help to provide self-configuring, self-managing, and self-optimization facilities.

Enhanced management functions for Linux and other virtual

Images

The z/VM virtual systems management application programming interface (API), first introduced in z/VM V4.4, is provided for System z platform provisioning applications (such as IBM Director and programs developed by non-IBM solution providers) for ease of use in creating and managing large numbers of Linux and other virtual images running on z/VM.

With z/VM V5.3, a new sockets-based server supports the z/VM virtual systems management API. The sockets-based server is multitasking-capable and supports both AF_INET and AF_IUCV socket requests.

In addition to the new server, enhancements provided in z/VM V5.3 include:

New functions to:

- Create, delete, and query the IPL statement in a virtual image's directory entry
  - Create and delete virtual switches and guest LANs
  - Obtain processor, memory, and device information for active virtual images
  - Check the validity of a given user ID/password (or passphrase) combination
- Enhancements to existing functions to:
  - Provide values of specific attributes for selected query functions, rather than return a buffer containing QUERY command output
  - Exploit the new Asynchronous CP Command function available in z/VM V5.3
  - Accept passphrases and forward them to the external security manager to be set, changed, or deleted
  - Provide a list of active virtual images

The new and enhanced API functions for z/VM V5.3 have been implemented using the new sockets-based server. Functions provided in earlier releases of z/VM can also be invoked through the new server.

The sockets-based server replaces the Remote Procedure Call (RPC) server and CSL routines that were used to call the virtual systems management API in previous releases of z/VM. The RPC server is still available in z/VM V5.3, with all of the functions that were available in z/VM V5.2. However, the enhancements provided in z/VM V5.3 are not available through the RPC server, for which no future enhancements are planned.

Documentation on the use of the API with the RPC server and CSL routines will not be updated and will not be included in the V5.3 bookshelf. IBM intends to remove the RPC server from a future z/VM release.

New function level for DirMaint

The IBM Directory Maintenance Facility (DirMaint) has been upgraded to a new function level (FL530) in z/VM V5.3. In addition to service being applied since FL510, DirMaint FL530 includes:

- Supporting new and changed directory statements in z/VM V5.3

- Eliminating indefinite wait times when a DATAMOVE machine cannot access all required resources for a DASD management function
- Providing more detailed information about the causes of errors returned from the z/VM virtual systems management API

Enhancements to the Performance Toolkit

In addition to being upgraded to a new function level (FL530), the Performance Toolkit for VM feature has been enhanced for z/VM V5.3 to:

- Support passphrases when accessing the Performance Toolkit using the Web interface
- Change the service process for Performance Toolkit from a full-part replacement MODULE to service by individual object parts, reducing the size of the service deliverable
- Provide new or updated displays and reports to support the following new z/VM V5.3 functions:
  - Linux monitor data for virtual CPUs and steal time counters
  - Monitor data for virtual network devices and virtual switches
  - Monitor data for guest simulation of zAAPs, zIIPs, and IFLs
  - Monitor data for up to 32 processors in a z/VM image

Enhanced guest configuration

z/VM V5.3 helps improve the guest LOGON process by providing a new COMMAND directory statement in a virtual machine definition or profile to configure the virtual machine. Any form of a CP command may be invoked using this capability, including privileged class commands (such as SET RESERVED), on behalf of the virtual machine, thus eliminating the need to provide some other method to configure it.

z/VM Integrated Systems Management

z/VM integrated systems management support uses the Hardware Management Console (HMC) to help enable administration of z/VM guests without having to establish additional network connections and reducing complex configuration of the system. The HMC will automatically detect z/VM images and provide integrated GUI-based basic management of z/VM guests. The z/VM integrated systems management capability supports the following image management functions: activate, deactivate, and display guest status.

z/VM provides a local management interface to allow basic z/VM systems management functions to be managed from the HMC. A new SCLP system

service (*SCLP) will allow you to receive and transmit HMC events. A new proxy server will exploit this service and direct requests to the Systems Management API server to perform the desired function, and send the results back to the HMC using *SCLP.

A new VM event system service (*VMEVENT) is provided that gives notification about certain events that occur in the VM system, such as some virtual machine status changes. The proxy server will receive notification when these events occur, and will report these back to the HMC through the *SCLP interface.

To manage guests with the HMC on the z9 EC and z9 BC, the HMC and Support Element (SE) must be at level 2.9.2. Refer to the z/VM subsets of the 2094DEVICE and 2096DEVICE Preventive Service Planning (PSP) buckets prior to installing a z9 EC or z9 BC, because a minimum MCL level may be required.

To manage guests with the HMC on the z990, z890, z900, and z800, the HMC must be at level 2.9.2, and the SE must be at the following level:

- 1.8.2 for the z990 and z890
- 1.7.3 for the z900 and z800

A minimum MCL level is also required. Refer to the z/VM subsets of the PSP buckets for your particular server.

Installation, service, and packaging changes

This section describes changes to the z/VM installation and service processes and how z/VM is packaged (what facilities are provided with z/VM or offered as features).

Additional DVD installation options

z/VM V5.3 provides additional capabilities for installing z/VM from DVD. The second-level DVD installation process now supports moving the contents to an FTP server directory or a second-level CMS minidisk, and then installing from the server or minidisk. This provides more options for customer environments and can facilitate the electronic delivery of z/VM.

Enhanced status information

The automated service command SERVICE has been enhanced to display the service and production levels for preventive service (RSU), and to display an applied, built, and production status for corrective service. This can provide a quicker and easier way to determine service status.

RSCS repackaged as an optional feature

Remote Spooling Communications Subsystem (RSCS) V3.2.0 (5684-096) has been repackaged and is now available for licensing under International Program License Agreement (IPLA) terms and conditions. RSCS Networking for z/VM, Function Level 530 (FL530), is available as a priced, optional preinstalled (installed disabled) feature of z/VM V5.3. Pricing is based on engine-based Value Units and is available for both IFL and standard processor configurations.

RSCS FL530 provides dynamic command authorization support through a new server, RSCSAUTH, that runs as a disconnected z/VM server and is authorized for all RSCS commands. This can eliminate the need to recycle RSCS when changing system and link authorizations.

New RACF Security Server for z/VM

With z/VM V5.3, the standalone RACF for VM V1.10.0 (5740-XXH) product has been repackaged with new function added and is now called the RACF Security Server for z/VM, function level 530 (FL530). It is a priced, optional preinstalled (installed disabled) feature of z/VM V5.3 and will operate only with z/VM V5.3.

The new RACF Security Server feature includes support for mixed-case passwords and password phrases. A password phrase is a string of up to 100 characters, including blanks, and can be used in addition to, or in place of, the traditional 8-character password. An installation exit is provided to help enable customers to define rules governing the length and content of password phrases.

Additional password management enhancements have been added, including:

- Validation of a password and password phrase using DIAGNOSE code X'88' and the new DMSPASS CSL routine
- Operation with the new industry-standard LDAP server included in z/VM V5.3 to enable remote management of passwords and selected user attributes, and to enable remote applications to perform authorization and auditing using RACF for z/VM

- Access by password phrase, allowing the replacement of the 8-character password from a user ID
- Enhanced security of password reset operations, which now removes the password completely, rather than resetting the user password to the default group name, as in prior releases
- Adding the user's password to the password history list when the password is reset by an administrator
- Providing the capability for passwords to be set by administrators or authorized password management applications without the need for the user to immediately change the password on its first use, improving the auditing of password changes

To simplify analysis of the security audit trail, the RACF SMF Unload utility has been updated to store the unloaded data in industry-standard XML format, making it suitable to be examined by a variety of applications, including XML browsers and spreadsheets.

This new feature will be the base for all future RACF enhancements on z/VM and works with the existing functions and features of z/VM to provide improved discretionary and mandatory access controls, separation of duties, and auditability capabilities of z/VM.

U.S. daylight saving time effect on z/VM

A provision of the U.S. Government's Energy Policy Act of 2005 and similar legislation enacted by the governments of Canada and Bermuda extends daylight saving time (DST) by four weeks, beginning in 2007. Starting in March 2007, daylight saving time in the United States, Canada, and Bermuda will begin on the second Sunday in March and end on the first Sunday in November.

New sample system configuration file statements will be shipped with z/VM V5.3. System programmers should change the dates that are specified on TIMEZONE_BOUNDARY statements in the existing system configuration files that their systems use.

z/Architecture CMS shipped as a sample program

z/Architecture CMS is shipped as a sample program with z/VM V5.3, with no formal support available. This version of CMS runs in z/Architecture 31-bit addressing mode and enables the use of z/Architecture instructions, including those that operate on 64-bit registers, by CMS programs, while permitting most existing ESA/390-architecture CMS programs to continue to function without change.

z/Architecture CMS does not exploit or explicitly support 64-bit addressing mode, but it does not impose serious restrictions on programs that enter 64-bit addressing mode themselves. For additional z/Architecture CMS details and usage restrictions, refer to the z/VM Web site at:

www.ibm.com/eserver/zseries/zvm/related/zcms/

Withdrawal of ROUTED and BOOTP servers

The ROUTED and BOOTP servers have been removed from z/VM V5.3. This satisfies the Statement of General Direction made in the Software Announcement dated July 27, 2005.

MPROUTE is the only dynamic routing server supported by TCP/IP for z/VM FL530.

Additional changes

This section describes additional changes included in this z/VM release.

Support for searches across PDF files in the z/VM Library

On the z/VM V5.3 edition of the IBM Online Library: z/VM Collection, SK2T-2067-24, and IBM Online Library: z/VM Collection on DVD, SK5T-7054-01, you can now do IBM BookManager-style searches across the PDF files in the z/VM V5.3 PDF library (directory). This enhancement has also been retrofitted to the z/VM V5.2 PDF library on this edition of the CD-ROM and DVD.

B

Answer key

This answer key corresponds to the questions for review and topics for further discussion at the end of each chapter.

© Copyright IBM Corp. 2007. All rights reserved.

Chapter 1 Introduction to the mainframe hardware systems

1. ***Multiprocessor*** **means several processors (and that these processors are used by the operating system and applications). What does** ***multiprogramming*** **mean?**

   Multiprogramming means several programs are run at the same time on a uniprocessor. The operating system executes part of one program for some time, then another program for some time and so on. This way, to the user, it appears that all programs are running at the same time.

2. **Usually, each read or write operation on a non-shared DASD device is one I/O operation. How many I/O operations will be involved when the DASD is in a shared mode?**

   When the DASD is shared, each read or write will essentially involve three operations since the operating system will have to issue the RESERVE and RELEASE commands in addition to the read or write command.

3. **What is the role of PR/SM?**

   PR/SM contains the functions that are used to create and manage the LPARs on the hardware.

4. **What changes are needed for z/VM applications to work in an LPAR?**

   No changes are required.

Chapter 2 Introduction to virtualization and z/VM

1. **Which method of virtualization is used in grid computing?**

   Aggregation.

2. **How do you compare physical hypervisors and software hypervisors?**

   Physical hypervisors are implemented at a hardware level and the software hypervisors are implemented at a software level.

3. **What could be the reason for removing the “basic mode” of operation option on the latest mainframes?**

   Mainframes are typically used for infrastructure consolidation and for hosting multiple operating systems with varying workloads simultaneously.

   These kinds of requirements use the LPAR mode of operation. So, the “basic mode” is, practically, never used to run any mainframe operating system.

4. **True or false - The processor weight can be adjusted in the IOCDS.**

   False. Input/Output Configuration Data Set (IOCDS) does *not* contain the processor (CPU) information. The processor information including the processor weight should be adjusted in the Partition Image Profiles.

5. **What is needed (and not how) to create an LPAR? Where does an LPAR get it?**

   LPAR name, partition number, partition identifier, logical processors with their weight and capacity, central and expanded storage and I/O devices and paths to them.

   LPAR name, partition number, and I/O devices and paths to them are defined in the IOCDS. Logical processors with their weight and capacity and central and expanded storage are defined in the Partition Image Profiles.

6. **What are the three types of operating environments we introduced in this chapter?**

   Native mode (also called basic mode), LPAR mode, and the guest mode.

7. **List at least three operating systems that can run as z/VM guests.**

   Linux, z/OS and z/VM are three examples. Refer to Chapter 13, “Guest operating systems” on page 387 for more details about this topic.

Chapter 3 z/VM history

1. **What was the major enhancement in XA architecture that was used to dispatch guests more efficiently?**

   Start Interpretive Execution (SIE).

2. **When were the first virtualization ideas presented?**

   1959.

3. **What is the EBCDIC representation of the letter D?**

   C4.

4. **What is SFS, and which decade saw its introduction?**

   Shared File System, 1980s.

5. **What was a major difference between z/VM 3.1 and z/VM 4.1?**

   Licensing terms and conditions.

Chapter 4 z/VM overview

1. **Which components of z/VM do you know? List at least five components.**

   Base products:

   – Control Program (CP)
   – Conversational Monitor System (CMS)
   – Transmission Control Protocol/Internet Protocol (TCP/IP) for z/VM
   – Advanced Program-to-Program Communication/Virtual Machine (APPC/VM)
   – Virtual Telecommunications Access Method (VTAM) Support (AVS)
   – Dump Viewing Facility
   – Group Control System (GCS)
   – Hardware Configuration Definition (HCD) and Hardware Configuration
   – Manager (HCM) for z/VM
   – Language Environment
   – Open Systems Adapter Support Facility (OSA/SF)
   – Restructured Extended Executor/Virtual Machine (REXX/VM)
   – Transparent Services Access Facility (TSAF)
   – Virtual Machine Serviceability Enhancements Staged/Extended (VMSES/E)

2. **What are the main responsibilities of a z/VM system programmer?**

   The z/VM system programmer installs, tailors, and maintains the operating system. The system programmer also installs and customizes applications or products on z/VM.

Chapter 5 Control Program for new users

1. **Explain the CP user directory and its purpose.**

   The CP user directory defines all of the users and guests that exist on the system and defines their virtual machine configuration and virtual device configuration.

2. **This question has two parts:**

   a. **What important virtual machine parameter do you need to remember to check (or set) before disconnecting from a production virtual machine?**

   SET RUN ON

   b) Why is it important to check or set the parameter mentioned in part a?

If SET RUN is not ON when you disconnect, then depending on certain factors, your virtual machine could be placed in CP mode during the disconnect or a subsequent reconnect.

It is important to avoid this on a production machine because it will pause the guest operating system, thus preventing it from running until a BEGIN command is issued.

3. **You are a system administrator for a virtual machine that is providing Web hosting services for a very popular company that uses their Web site to process customer order information.**

   You are asked to make a few hardware configuration changes to the z/VM guest running the Web server, but you obviously cannot simply shut down this customer's very busy Web site.

   How would you issue the needed commands to CP without shutting down the Web server?

   Issue the needed commands preceded by #CP. This will cause CP to interpret the command and then immediately resume the guest operating system.

   This method is certainly preferable to issuing a #CP STOP command or issuing a PA1 key press from your terminal emulator because those methods will temporarily pause the guest operating system running the Web server.

4. **What is the difference between a TDISK, VDISK and a regular minidisk? In what situations would you prefer one over the other two?**

   – A TDISK is a temporary disk that is mapped to a section of a real DASD device on the system.
   – A VDISK is a temporary disk that is mapped to a section of storage.
   – A regular minidisk is a section of DASD reserved for a specific virtual machine and is permanent.

   Here are the usage situations for each disk:

   – Because a regular minidisk stores files permanently, use it for general file storage.
   – A TDISK only stays around while you are logged on to the system, so use it when you need a large amount of disk space, but only temporarily.
   – A VDISK exists in storage which is extremely fast compared to normal disk based devices, so it is preferable to use when you need a small but fast temporary disk.

5. **Assume that you issue the HELP command, only to discover that HELP does not work. You later learn that HELP failed because you do not have access to the CMS help disk, which is MAINT's 19D disk.**

   What CP command (the full command, not just the command name) would you need to use before you could access this disk to use the HELP command within CMS?

   LINK MAINT 19D 19D RR

6. **You need 6 GB of disk space to perform a Linux install. However, you do not have disks that are large enough for this task.**

   Based on your knowledge of how real DASD devices are typically sized and how they are partitioned and given to users, what type of DASD will your administrator likely give you access to, so you can perform your install?

   Your administrator could give you an entire 3390-mod9 DASD pack, which is roughly 9 GB in size. A 3390-mod3 is approximately 3 GB in size, which is too small, thus making the mod9 the logical next choice because it is the next size up.

   If the system administrator did not want to give an entire pack, then the administrator could create a minidisk for you on a 3390-mod9 (or larger) pack at least 7200 cylinders in size, because 7200 cylinders would be approximately equivalent to 6 GB.

Chapter 6 Conversational Monitor System

1. **Which virtual address is typically accessed as A in a CMS environment?**

   191

2. **What is the maximum number of minidisks that can be accessed concurrently by a single CMS user? What command would you use to show how many minidisks are accessed right now by your CMS guest?**

   26

3. **Describe which command you would use to list files interactively, and which command you would use to list files statically. Explain a situation in which you would prefer each.**

   Interactive: FILELIST, whenever you want to perform a sendfile or other interactive command without typing the entire file name.

   Static: LISTFILE, ideally used when you want to receive static information such as a list of files matching a predetermined pattern.

4. **Provide a command that would list all the files with filename PROFILE on your disk accessed with file mode A.**

   listfile PROFILE * A

   or

   filelist PROFILE * A

5. **List two or more ways of creating a file using CMS tools. Provide a use case for each.**

   Methods include copying an existing file with the **`copy`** command (or some network protocol like ftp, and so on), or using XEDIT to create new content.

6. **Review your understanding of XEDIT's file-closing mechanisms. Make a distinction between the `qquit`, `quit`, and `file` commands used in an XEDIT session, and their use cases.**

   – FILE will save the file and exit normally.

   – QUIT will close the file without saving, but will prompt you if edits have been made during the session that will be lost.

   – QQUIT will quit *immediately* without saving even if edits had been made during the current editing session.

7. **Describe the role of your PROFILE EXEC A and why it is important.**

   The PROFILE EXEC A configures your CMS session and loads preferences and custom configuration options at CMS startup. It controls many fundamental aspects about your CMS environment.

8. **Examine your PROFILE EXEC A. How would you set up your CMS environment to automatically list files accessed with mode C interactively upon each IPL?**

   ```
   Insert the line FILELIST * C as the last option in the file.
   ```

9. **As an optional experiment, customize your PROFILE EXEC as you deem appropriate. Be sure to make a backup before you begin editing.**

   This exploratory lab has no single correct answer.

Chapter 7 REXX basics

Question 1 Indicate whether the following questions are accurate. (Y/N)

- Yes.

  It does *not* matter if the code is upper case or lower case and spaces are ignored.

- Yes

  Although REXX compilers are available, the interpreted version remains the most popular.

- No.

  REXX will run in a variety of environments, but not in *every* environment.

- No.

  REXX is useful to advanced programmers *and* general novice programmers.

- Yes.

  REXX language syntax is easy to use and quite “English-like”.

Question 2

Answer: The correct answer is 3. mm = 12/ dd = 25/ yy = 90

Explanation:

- Assign from the beginning of the string, up to but *not* including the fourth character, to the variable mm.
- Then starting in position 4, scan to position 7 and assigns the three characters, not including the one in the seventh position, to dd.
- Finally, beginning at position 7, the remainder of the string is parsed into yy.

Question 3

The PARSE statement, as shown, is looking for blanks as delimiters between words. Since there are none m1 would get the entire string and m2 through m6 would be set to NULL strings.

Question 4

Answer: W1 = A; W2 = TEST STRING; W3 = IS A TEST; W4 = STRING

Explanation:

- Assign characters beginning at position 9 for a length of 1 to w1.
- Assign characters beginning at position 10 (9 + 1) for a length of -4. Because there is a negative length indicated, assign to the end of the string to w2.
- Assign characters beginning at position 6 (10 - 4) for a length of 10 characters to w3.
- Assign characters beginning at position 16 (6 + 10) to the end of the string to w4.

Question 5

The following would be displayed on the screen: `steelpegs $100`

Question 6

1. SUBROUTINE
2. FUNCTION
3. TASK

Question 7

MARKS.FRED

Explanation:

marks.fred has not been assigned any value so it has the initial value of the variable name, all in upper case.

Question 8

Use the ADDRESS instruction, followed by the name of the new environment.

Question 9

Use the ADDRESS built-in function as follows:

```
x = ADDRESS()
```

Question 10

This EXECIO instruction would read 6 lines beginning at line 50.

```
/* READ EXEC */
"EXECIO 6 DISKR test file A 50"
 ...
 ......
 EXIT
```

Chapter 8 Pipelines

1. **Answer:**

   ```
   pipe literal copy PROFILE EXEC A profile save A | Command
   ```

2. Answer:

```
/* Exec to copy in mixed case */
'pipe literal COPY PROFILE EXEC A profile save A',
'| Command'
exit rc
```

3. Answer:

```
/*list all the logged on users alphabetically */
'PIPE cp q names ',
     '| split , ',
     '| nlocate /- DSC/ ',
     '| strip ',
     '| sort ',
     '| stem users.'
say 'USER       SCREEN ADDRESS'
say '----------------------'
do i=1 to users.0
         say users.i
end
```

4. Answer:

```
/*list all users in alphabetically sorted list */
'PIPE (endchar ?) cp q names ',
     '| a: fanout',
     '| split , ',
     '| nlocate /- DSC/ ',
     '| strip ',
     '| sort ',
     '| stem users.',
     '?',
     'a:',
     '| split , ',
     '| locate /- DSC/ ',
     '| strip ',
     '| sort ',
     '| stem users1.'
say 'USER       SCREEN ADDRESS'
say '----------------------'
do i=1 to users.0
         say users.i
end
do i=1 to users1.0
         say users1.i
end
```

5. **Answer:**

   ```
   HELP PIPE MENU
   ```

   Use the PREFACE stage to invoke another stage or subroutine pipeline, write the records produced by that stage or subroutine pipeline to its primary output stream, and then copy all records from its primary input stream to its primary output stream.

Chapter 9 System administration tasks

1. **List at least three privileged CP commands.**

   There are several privileged CP commands, including:

   ```
   shutdown, cpaccess, attach
   ```

2. **Which CP utility is used to format and allocate DASDs?**

   CPFMTXA

3. **List the Named Saved Segment (NSS) used by CMS users.**

   CMS

4. **Which z/VM system component accesses the PARM disk?**

   Control Program (CP)

5. **What is the file name where you define your resource addresses, DASD lists and CP-owned disks?**

   SYSTEM CONFIG file

6. **If you want to back up your system, which programs you have to use in order to back up DASDs and SPOOL files?**

   To back up DASD, use DDR.

   To back up SPOOL, use SPXTAPE.

7. **Which command (or commands) do you have to issue in order to access the PARM disks?**

   ```
   CPRELEASE, LINK, CPACCESS
   ```

8. **List the major steps needed to apply a service using the VMSES/E commands.**

   Receive, apply, build

9. **Which command do you execute to access the VMSES/E disks to CMS component?**

   ```
   VMFSETUP zvm530 CMS
   ```

Chapter 10 Performance

1. **Which CP command should you use to find out paging space utilization?**

   `QUERY ALLOC PAGE`

2. **How could you find out if a user is using too much CPU?**

   `PTK exception, IND USER`

3. **What CP system service does a user need to access MONITOR data?**

   * MONITOR

4. **Which three lists are used by the scheduling and dispatching code?**

   Dormant, Eligible, Dispatch

5. **What could you use a DCSS for?**

   To share applications and data.

6. **Which command would you use to display information about real storage?**

   `QUERY FRAMES`

7. **List six displays that you could use with the `INDICATE` command.**

   IO, NSS, PAGing, Queues, SPaces, USER, ACTive, LOAD

8. **What types of information does the monitor collect?**

   Event and Sample

Chapter 11 Networking and connectivity

1. **In what release of z/VM was guest LAN support added?**

   Guest LAN support was added in z/VM 4.2.

2. **List three types of hardware network adapters supported by z/VM.**

   OSA-2, OSA-Express, OSA-Express2.

3. **The two types of NICs supported by z/VM are QDIO and _____?**

   HiperSockets.

4. **Name the two primary point-to-point network types supported by z/VM.**

   Channel-to-channel (CTC) and Inter-User Communications Vehicle (IUCV).

5. **What is the difference between a layer 2 network and a layer 3 network?**

   With a layer 2 network, z/VM guests act just like a standard PC on a network and use the media access control (MAC) address associated with the network card to send an receive data.

   With a layer 3 network, routing is performed based on the IP address associated with the network card.

6. **What is the primary advantage of a VSWITCH over a guest LAN?**

   A VSWITCH does not require a routing virtual machine.

7. **You have just added an OSA device to a guest. When you run QUERY NIC to verify it was added, nothing shows up. Why? Is there some other command you can use to verify it was added?**

   OSAs are real hardware network adapters, not virtual NICs. Use **QUERY OSA** to see what OSA devices have been made available to your guest.

8. **You want to couple a NIC on your guest to a guest LAN that was defined in the normal manner. What command do you need to run to get access to the LAN?**

   By default guest LANs are unrestricted, so no special permission is required.

9. **You now want to couple a NIC on your guest to a VSWITCH. What command do you need to run to get access to the LAN?**

   ```
   SET VSWITCH <switchname> GRANT <guest>
   ```

10. **What command can you use to view the access list for a VSWITCH?**

    ```
    QUERY VSWITCH ACCESSLIST or QUERY VSWITCH <switchname> ACCESSLIST
    ```

11. **If you wanted to temporarily add an OSA device to a guest, what command would you use?**

    ```
    ATTACH xxx-yyy TO <guest>
    ```

12. **Assume you wanted to add the same OSA device to the guest permanently. What steps would you need to perform?**

    Edit the guest's directory entry and use the DEDICATE statement to dedicate the three OSA devices to the guest.

13. **What command would you use to view how much data has been sent or received by an OSA device?**

    ```
    NETSTAT DEVLINKS
    ```

14. **Log on to a z/VM system as MAINT and examine the user directory. How many service virtual machines can you identify?**

    Some of the default service virtual machines include: TCPIP, IMAP, FTPSERVE, SMTP, NAMESRV, PORTMAP, TFTPD, DHCPD, MPROUTE,

and SSLSERV. There are a number of others we have not mentioned; refer to *z/VM TCP/IP Planning and Customization Guide* for the complete list.

Chapter 12 Security

1. **What kinds of resources can an external security manager be used to secure?**

   Minidisks, tape devices, virtual LANs, terminals, and shared user IDs.

2. **How many privilege classes are defined by default on a z/VM system?**

   There are seven privilege classes defined by default on a z/VM system (classes A - G).

3. **What user role does the class C privilege define?**

   Class C on z/VM defines a system programmer role.

4. **What type of encryption and hashing methods are supported by the CPACF feature of newer mainframes?**

   Symmetric ciphers such as DES, Triple DES, AES, MAC message authentication, and the SHA-1 and SHA-256 hashing algorithms.

5. **What is the name of the optional System z cryptographic hardware feature, and what two modes does it support?**

   Crypto Express2 (CEX2), accelerator mode (CEX2A) and coprocessor mode (CEX2C).

6. **What directory statement must be added to a guest in order for it to share access to a hardware cryptographic card?**

   APVIRT.

Chapter 13 Guest operating systems

1. **What is the difference in the memory utilization of the physical mainframe system when operating systems are running in LPAR mode and when they are running as a guest of z/VM?**

   When individual LPARs are activated and running on the physical system, the memory utilization may not be optimized. But in the case of a z/VM guest, z/VM automatically optimizes the total memory consumed.

   When a guest is not using the memory allocated to it, z/VM steals that memory and utilizes for useful work. When the guest needs it back, z/VM automatically allocates the required amount.

2. **True or False - At many sites, it is typical to see several hundreds of Linux guests hosted on a single z/VM LPAR.**

   True.

3. **True or False - To run a Parallel Sysplex under z/VM, special hardware is required on the physical mainframe system.**

   False.

4. **When the z/VSE Operating System is running as a guest under z/VM, what is the benefit in terms of “paging” requirements?**

   CP can be used to do paging for z/VSE.

5. **Is it possible to install a third level z/VM system?**

   Yes. As far as resources are available, you can go up to “n” level.

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

© Copyright IBM Corp. 2007. All rights reserved.

Trademarks

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

Redbooks (logo) ®
eServer™
z/Architecture®
z/OS®
z/VM®
z/VSE™
zSeries®
z9™
AIX®
AS/400®
BookManager®
DirMaint™
DB2®
DFSMS™
DFSMS/VM™
DS4000™
DS6000™
DS8000™
Enterprise Storage Server®
Enterprise Systems Architecture/390®
ECKD™
ESCON®
FlashCopy®
FICON®
HiperSockets™
IBM®
IMS™
Language Environment®
MVS™
NetView®
OMEGAMON®
OS/390®
Parallel Sysplex®
Print Services Facility™
Processor Resource/Systems Manager™
PR/SM™
PROFS®
Redbooks®
RACF®
RAMAC®
REXX™
RMF™
S/360™
S/370™
S/390®
System z™
System z9™
System Storage™
System/360™
System/370™
System/390®
SystemView®
SAA®
Tivoli®
TotalStorage®
VM/ESA®
VTAM®
WebSphere®

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

AMD, the AMD Arrow logo, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

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

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

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

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

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

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks” on page 438. Note that some of the documents referenced here may be available in softcopy only.

- *IBM System z Connectivity Handbook*, SG24-5444
- *Linux on IBM eServer zSeries and S/360: Performance Toolkit for VM*, SG24-6059
- *IBM TotalStorage Peer-to-Peer Virtual Tape Server Planning and Implementation Guide*, SG24-6115
- *The IBM Enterprise Information Portal: A Cookbook*, SG24-6125
- *Introduction to the New Mainframe: z/OS Basics*, SG24-6366
- *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES9*, SG24-6695
- *Introduction to the New Mainframe: Networking*, SG24-6772
- *Introduction to the New Mainframe: Security*, SG24-6776
- *Linux on IBM eServer zSeries and S/390: Performance Measurement and Tuning*, SG24-6926
- *IBM eServer zSeries 990 (z990) Cryptography Implementation*, SG24-7070
- *IBM z/VM and Linux on IBM System z: Virtualization Cookbook for Red Hat Enterprise Linux 4*, SG24-7272
- *Using Discontiguous Shared Segments and XIP2 Filesystems With Oracle Database 10g on Linux for IBM System z*, SG24-7285
- *IBM System Storage TS1120 Tape Encryption Planning, Implementation, and Usage Guide*, SG24-7320
- *Using z/VM for Test and Development Environments: A Roundup*, SG24-7355
- *Using the z/VM INDICATE Command*, TIPS0592

© Copyright IBM Corp. 2007. All rights reserved.

Other publications

These publications are also relevant as further information sources:

CMS

- *z/VM CMS Commands and Utility Reference*, SC24-6073
- *z/VM CMS File Pool Planning, Administration and Operation*, SC24-6074
- *z/VM CMS Pipelines Reference*, SC24-6076
- *z/VM CMS Pipelines User Guide*, SC24-6077
- *z/VM CP Commands and Utilities Reference*, SC24-6081
- *z/VM CP Planning and Administration*, SC24-6083
- *z/VM V5R2.0 Running Guest Operating Systems*, SC24-6115
- *z/VM Saved Segments Planning and Administration*, SC24-6116
- *z/VM Virtual Machine Operation*, SC24-6128
- *Linux on System z Device Driver, Features, and Commands*, SC33-8289

Installation and Service

- *z/VM V5R2.0 Guide for Automated Installation and Service*, GC24-6099
- *z/VM V5R2.0 Summary for Automated Installation and Service*, GA76-0406 (DVD) and GA76-0407 (tape)
- *z/VM Guide for Automated Installation and Planning*, GC24-6099
- *z/VM Service Guide*, GC24-6117
- *VMSES/E Introduction and Reference*, GC24-6130

Networking and connectivity

- *z/VM General Information*, GC24-6095
- *z/VM Connectivity*, SC24-6080
- *z/VM: Getting Started with Linux on System z*, SC24-6096
- *z/VM TCP/IP User's Guide*, SC24-6127

Performance

- *z/VM Performance Version 5 Release 3*, SC24-6109
- *z/VM Performance Toolkit Guide*, SC24-6156
- *z/VMCP Messages and Codes*, GC24-6119

REXX/VM

- *z/VM V4 R2: REXX/VM Reference*, SC24-6035
- *z/VM V3 R1: REXX/VM User's Guide*, SC24-5962

Security

- *Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6135
- *Directory Maintenance Facility Commands Reference*, SC24-6133
- *RACF: Security Administrator's Guide*, SC28-1340
- *RACF: General User's Guide*, SC28-1341
- *A Clear Key/Secure Key Primer*, WP100647

Online resources

These Web sites are also relevant as further information sources:

- AF_IUCV protocol support for Linux on System z

  `http://www.ibm.com/developerworks/linux/linux390/useful_add-ons_af-iucv-v1.html`

- OSI model

  `http://en.wikipedia.org/wiki/OSI_model`

- VM/ESA TCP/IP Performance

  `http://www.vm.ibm.com/devpages/bitner/presentations/tcpip`

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics
2105 17
3390 disk drives 16

A
active partition 47
address, I/O 6
application program 18, 52

B
base z/VM
  operating system 55
basic cage 9
basic mode 44
basic user 103
  Control Program 103

C
Cache 14
cache usage, port utilization (CPU) 109, 123
Cambridge Monitor System (CMS) 30
Capacity management 321
Capacity on Demand 13
CEC
  definition 4
CEC cage 9
Central Electronics Complex (CEC) 4
central electronics complex (CEC) 4
Central processing unit
  CPU
    definition 3
central processing unit
  definition 3
central processor (CP) 4, 12
Central Processor Complex (CPC) 4, 8
Central Storage 14
CF 390
Channel Path Identifier (CHPID) 7
channel path identifiers 7
channels, definition 5
CHPID (Channel Path Identifier) 7
CHPID address 21
class G 111
class G user 112
client representative 73
CMS command
  SENDFILE 136
COBOL
  language program 71
command line 103
  area 143
  environment 106
  interface 106
Concurrent Copy 17
Consolidation
  definition 2
  reasons to consolidate 2
Control Program (CP) 30, 103
control unit 5, 8
control unit, definition 5
Conversational Monitor System (CMS) 104
copy service 305
count key data (CKD) 18
coupling facility 390
Coupling Facility (CF) 13, 390
  service machine 391
CP 12–13, 26
CP command 105
CP-40 62
CP-67 63
CPC
  definition 4
CPU 4, 27
Creating LPARs 46
CTSS 62
customer engineer (CE) 72

D
DASD 16
DASD pack 111
Dedicated logical processors 47
desktop computer 114
detailed information 56
device number 8, 12, 21, 120, 122
  8FF 133

© Copyright IBM Corp. 2007. All rights reserved.

9FF 132
device number, definition 8
device type 120, 122
devices, I/O 5
Direct Access Storage Device (DASD) 15–16, 126
Direct Hardware Support Method 44
directory entry 105
disk drive 5
disk hardware 1
DS8000 17

E

Early architectures
overview 3
EDT 107
Emulation of function 35
Enterprise Storage Server 17
Enterprise Systems CONnection 7
ESCON 7
channels 21
director 22
ESS 17
Exclusive read (ER) 131
Exclusive write (EW) 131
Expanded Storage 15
expanded storage 15
Extended Remote Copy 17
external security manager (ESM) 57

F

Fast Ethernet 21
FC-AL 17
fiber channel arbitrated loop 17
FIber CONnection 7
Fiber Distributed Data Interface (FDDI) 21
Fibre Channel Protocol (FCP) 20
FICON 7
FICON channels 21
field technical sales support (FTSS) 73
file name 138
File System 108
first level
CP 117
CP instance 117
virtual machine 118
z/VM 55
z/VM operating system 55
z/VM system 55–56
fixed block architecture (FBA) 18
FlashCopy 17, 305
FlashCopy V2 306
FTSS (field technical sales support) 73

G

Gigabit Ethernet 21
graphical user interface (GUI) 2
complex code 2
Grid computing 30
guest mode 105–106
guest operating system 2, 39–41, 104, 106, 118
Working 114
guest OS 39
guest system 55, 388–389, 391–392

H

Hardware Management Console (HMC) 26
Hardware partitioning 37
Hardware Storage Area (HSA) 20
HCD 46
HELP command 106, 146
HiperSockets 49
History 59
host operating system 39
Host Page-Management Assist (HPMA) 49
How does virtualization work? 33
HSA 21
HyperPAV 307
hypervisor 30, 33, 44
hypervisor application 40
Hypervisor Call Method 43
Hypervisor technologies 41
Hypervisor-based partitioning 39

I

I/O channel 25
I/O connectivity, overview 20
I/O Control Data Set (IOCDS) 21, 46
I/O definitions 45
I/O device 5, 20, 30, 46
particular type 5
I/O operation 8
ICF (Integrated Coupling Facility) 13
IFL (Integrated Facility for Linux) 12
implemented on physical (IP) 35
independent software vendor (ISV) 72

Insulation 36
Integrated Console Controller (ICC) 15
Integrated Coupling Facility (ICF) 13
Integrated Facility for Linux (IFL) 12
IOCDS 21, 46
IOCDS (I/O Control Data Set) 21
IP address 51
IPL CMS 115
ISV (independent software vendor) 72

L

Layer 2 and Layer 3 network switching 49
LCSS 48
Linux 25, 30, 104, 389
Logical Channel Subsystems 48
logical partition 25, 44
logical partition (LPAR) 25
Logical partitioning 38
logically partitioned (LPAR) 34, 45
LPAR (logical partition) 26
LPAR mode 44
LPARs 25, 30
  I/O devices 25

M

mainframe consolidation 2
Mainframe system 1, 18
Mainframe terminology
  illustrated 3
Message Facility 390
message facility 390
MF 390
MIF 48
MIT 62
Multiple Allegiance 18
Multiple Image Facility 48
Multiple write (MW) 131
multiprocessor, terminology 10

N

Named Saved System (NSS) 114
network card 122
Network Interface Card (NIC) 140
network protocol 21
Network Virtualization 51
new user 103

O

Open System Adapter (OSA) 15
operating system 388
  channel subsystem 20
  new release 53
operating system (OS) 7, 30, 33, 49, 51, 104, 108
original equipment manufacturer (OEM) 72
OSA 15
OSA card 15

P

PA1 key 117–118
Parallel Access Volumes 17
Parallel Access Volumes (PAV) 307
Parallel Sysplex 390
Paravirtualization 43
Partition Image Profiles 46
partition weight 47
partitioning, introduction 23
PAV 18
PCHID 7
Peer-to-Peer Remote Copy (PPRC) 306
Performance 321
  Analyzing 342
  class 1 user 325
  class 2 user 325
  class 3 user 325
  CP Monitor 332
  CP QUERY 329
  DCSS 299, 336
  dispatching 323
  Event data 333
  MONDCSS 335
  Monitor Data Collection 334
  MONWRITE 336
  NSS 299, 336
  Omegamon 339
  Performance Toolkit 337
  Sample data 333
  Scheduling 323
  Tuning 349
  User Directory 332
performance 322
Peripheral Storage 15
personal computer 108
personal computer systems 1
physical channel identifiers 7
physical hypervisor 38

Physical partitioning 38
physical resource 25, 30, 32
  dynamic sharing 31
physical server 30, 37
physical system 25, 32, 37
  separate processors 25
Pipeline
  concepts 260
  developing 261
  device drivers 263
    264
    > 264
    >> 264
    CMS 264
    COMMAND 265
    CONSOLE 265
    CP 265
    FILE 264
    LITERAL 265
    PUNCH 266
    READER 266
    REXX 268
    REXXVARS 267
    STACK 265
    STEM 268
    TAPE 266
    VAR 268
    XMSG 265
  filters 268
    CHOP 269
    COUNT 270
    DROP 270
    FIND 269
    JOIN 270
    LOCATE 269
    NLOCATE 269
    PACK 270
    PAD 270
    SORT 270
    SPECS 269
    SPLIT 270
    STRIP 270
    TOLABEL 269
    UNIQUE 270
    XLATE 271
  Multistream 271
    FANIN 272
    FANINANY 272
    FANOUT 273
    Reference 273
Pipelines
  CMS 259
POR (power-on Reset) 26
power-on Reset (POR) 26
PR/SM 26
Preferred pathing 310
Principles of Operation
  reference 3
printed circuit (PC) 3
privilege class 111
  subset 111
Processor
  definition 3
Processor Resource/System Manager (PRSM) 26
processor unit (PU) 12
production system 388
PROFILE Exec 138
PRT 107
PU (processor unit) 12
punch cards 61

Q

QDIO Enhanced Buffer State Management 49

R

RDR 107
reader queue 138–139
Real DASD 127
  multiple types 128
real DASD
  device 146
  pack 127
  pack address 134
real device 104, 119
  abstracted view 119
real disc 129
real I/O
  configuration 54
real machine 52
real storage 13, 54, 124
Redbooks Web site 438
  Contact us xvii
Resource aggregation 34
Resource sharing 34
running Linux 389
running z/OS 390
running z/VSE 391–392

S

S/360 3
same time 7, 34, 49
  host computer 34
  multiple data transfers 23
  same disk data 7
SAN 15
SAP 12
Secure Sockets Layer (SSL) 57
SENDFILE command 136
serial storage architecture 16
server model 54
Server virtualization 37
service virtual machine 144
shared DASD environment 19
Shared logical processors 47
soft capping 47
Spool device 135, 137
  spool device types 135
SPOOL file 136
spool file 135
  disk space 137
Storage Area Networks 15
Support Center 73
Support Element (SE) 8, 26, 46
System
  definition 3
System 360 62
System 370 63
system administrator 25–26, 105, 109
System Assistance Processor (SAP) 10, 12
System operator 9
System programmer 4, 8, 51
system simulation 388
System z 1, 14, 44
  architecture 54
  configuration 9
  Connectivity Handbook 8
  environment 11, 23
  external I/O device 9
  hardware 8
  hardware architecture 2, 9
  hardware layout 9
  machine 9
  model 25
  processor 11, 24
  processor type 20
  server 8
System z9 Integrated Information Processor (zIIP) 13
System/360 diagram 5
Systems Adapter 122
systems engineer (SE) 73

T

tape drive 5, 52
  control unit 5
  real address 52
  virtual address 52
TDISK 127
TDISK allocation 132
TERM command 141–142
time slice 43
  yield rest 43
Translate, Trap and Emulate Method 42
Trap and Emulate Method 41
Type-1 Hypervisor 39
Type-2 Hypervisor 40

U

user directory 105
user ID 140

V

variable workload 31
VDISK 127
VDISK allocation 133
virtual device 103
  different types 119
virtual device number
  000C 136
  space 122
virtual environment 50
  backup copy 50
Virtual LAN 51
virtual machine
  directory entry 125
  user name 138
  virtual CPUs 122
  whole set 104
virtual machine (VM) 30, 105, 113, 392
virtual processor 36, 108
virtual resource 25, 32
Virtualization in action 49
Virtualization Technology 31, 55
virtualization to maintain outdated software 50

Virtualized resource 32
VLAN 51
VM/370 63
VM/ESA 65
VM/SP 64
VM/VSE Interface 392
VM/XA 64

W

Web server 146, 421
Web site 3

Z

z/OS system 390
z/VM 1, 29, 66, 103
  guest systems 390
z/VM guest 44–45, 116–117
  implementation 55
z/VM image 19
z/VM instance 24, 109
z/VM support 18
z/VM system 9, 45, 52, 137
zAAP processor 13
Zone relocation 48

Introduction to the New Mainframe: z/VM Basics

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Introduction to the New Mainframe: z/VM Basics

Understand introductory z/VM concepts

Learn basic system administration tasks to manage your system

Study z/VM performance, networking and security

This textbook provides students with the background knowledge and skills necessary to begin using the basic functions and features of z/VM Version 5, Release 3. It is part of a series of textbooks designed to introduce students to mainframe concepts and help prepare them for a career in large systems computing. For optimal learning, students are assumed to be literate in personal computing and have some computer science or information systems background. Others who will benefit from this textbook include z/OS professionals who would like to expand their knowledge of other aspects of the mainframe computing environment. This course can be used as a prerequisite to understanding Linux® on System z. After reading this textbook and working through the exercises, the student will have received a basic understanding of: the Series z Hardware concept and the history of the mainframe; virtualization technology in general and how it is exploited by z/VM; operating systems that can run as guest systems under z/VM; z/VM components; the z/VM control program and commands; the interactive environment under z/VM, CMS and its commands; z/VM planning and administration; implementing z/VM networking capabilities; tools for monitoring z/VM systems and guest operating systems; REXX language and CMS pipelines; and security issues when running z/VM.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7316-00 ISBN 0738488550