IBM Advanced Interactive Executive/370 (AIX/370) Administration Guide Version 1.2.1

Document Number SC23-2088-01

IBM Advanced Interactive Executive/370
(AIX/370)

Administration Guide

Version 1.2.1

Document Number SC23-2088-01

Edition Notice

Second Edition (March 1991)

This edition applies to Version 1.2.1 of the IBM Advanced Interactive Executive for the System/370 (AIX/370), Program Number 5713-AFL, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department 52QA MS 911
Neighborhood Road
Kingston, NY 12401
U.S.A.

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

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Subtopics

Trademarks and Acknowledgments

Trademarks and Acknowledgments

The following trademarks apply to this book:

AIX is a registered trademark of International Business Machine Corporation.

IBM is a registered trademark of International Business Machine Corporation.

NFS is a registered trademark of Sun Microsystems, Inc.

PS/2 and Personal System/2 are registered trademarks of Internationa Business Machines Corporation.

RPC is a trademark of Sun Microsystems, Inc.

RT is a registered trademark of International Business Machine Corporation.

System/370 is a trademark of International Business Machine Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc. i the USA and other countries.

Portions of the code and documentation were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

About This Book

This book provides instructions for personnel responsible for supervising AIX/370 systems. AIX/370 systems can consist of System/370 or 370 Extended Architecture computers running IBM AIX/370 for System/370 and System/370XA (hereafter referred to as AIX/370). This book should be used by the system administrator to oversee file system maintenance and preserve data integrity. It should also be used for performance tuning on the VM and AIX levels and for overseeing the VM File Transfer facility.

Subtopics

Who Should Read This Book

This book is written for the AIX/370 system administrator or anyone who is responsible for the planning and installation of AIX/370.

What You Should Know

This book assumes an understanding of:

- Computer hardware and softwar
- Communication
- The UNIX operating syste
- Currently installed hardware and software

Because AIX/370 is a guest on a VM system, being familiar with VM is helpful.

As a prerequisite, you should have read *AIX/370 General Information* and *AIX PS/2 General Information* to familiarize yourself with the requirements and capabilities of AIX. You should have also read the *AIX/370 Planning Guide* that identifies important areas to be explored before installation and evaluated during system use.

The AIX Operating System features Multibyte Character Set (MBCS) support. For information on MBCS, see the material in this book as well as the *AIX Guide to Multibyte Character Set (MBCS) Support*.

How to Use This Book

This book should be used to maintain control of the AIX/370 Operating System.

Chapter 1 presents a general description of the Transparent Computin Facility, some of the commands that the administrator is likely to need, and some specifics regarding the running of AIX/370 as a guest under VM. Material on using the IUCV driver is covered as well as information on printing in a TCF environment. MBCS considerations are also explored.

Chapter 2 presents detailed information on AIX file systems and TC replication.

Chapter 3 presents procedures for backing up file systems. It als examines file restoration on a multibyte character system and provides guidelines when mixing versions of AIX.

Chapter 4 explains the steps involved in using the **updatep** command to apply, commit, uncommit, or reject files.

Chapter 5 presents additional administrative concerns such as stoppin a runaway process, removing hidden files, and restoring free space.

Chapter 6 presents detailed information on system tuning, includin the tuning of main storage at the VM level and the AIX/370 level.

Chapter 7 describes the VM File Transfer facility including th software requirements, VM, AIX, and RSCS administrative tasks, and several examples of the procedures used to send and receive files. Appendix A lists messages related to VM File Transfer.

This book also contains a glossary of terms and abbreviations.

It will be necessary to refer to other manuals in the AIX library for certain administrative tasks. For example, information on the Network File System (NFS) can be found in *Managing the AIX Operating System*.

Subtopics
Highlighting

Highlighting

This book uses different type styles to distinguish among certain kinds of information. General information is printed in the standard type style (the type style used in this sentence). The following type styles indicate other types of information:

Boldface type indicates AIX commands, names of files and directories, site names, options, and fields. For example, **mount** is an AIX command.
UPPERCASE BOLDFACE type indicates environment variables. For example, when working in a specific mode, **NOTRANS** should be on.
Boldface italic indicates new terms introduced in the text. For example, the directory on which a file system is mounted is called the ***mount point***.
Italicized type indicates variable information. For example, the */<sitename>* directory is the mount point for a particular file system.
Monospace type indicates information the user enters or information generated by the system. For example, to find an unwanted core file, enter

```
find /u -name core -print
```

Related Publications

For additional information, you may want to refer to the following publications:

AIX Access for DOS Users Administrator's Guide, SC23-2042, explains how to install and administer the AIX Access for DOS Users program on the IBM PS/2, RT, and System/370 computers running the AIX Operating System with the AIX DOS Server. It covers the responsibilities for installation, daily operation, and maintenance of the AIX Access program.

AIX Access for DOS Users User's Guide, SC23-2041, describes the AIX Access for DOS Users program and shows how to use the file services of an AIX host while running DOS applications.

AIX C Language Reference, SC23-2058, describes the C programming language and contains reference information for writing programs in C language that run on the AIX Operating System.

AIX C Language User's Guide, SC23-2057, describes how to develop, link, and execute C language programs. This book also describes the operating dependencies of C language and shows how to use C language-related software utilities and other program development tools.

AIX Commands Reference, SC23-2292 (Vol. 1) and SC23-2184 (Vol. 2), lists and describes the AIX/370 and AIX PS/2 Operating System commands.

AIX Guide to Multibyte Character Set (MBCS) Support, GC23-2333, explains the basic concepts of AIX multibyte character set support and refers to other AIX books that contain more detailed information.

AIX Library Guide, Glossary, and Master Index, SC23-2324, describes the publications in the AIX Operating System library and contains a glossary of terms used throughout the library. This book also includes a master index to the contents of each of the publications in the library.

AIX Messages Reference, SC23-2294, lists messages displayed by the AIX Operating System and explains how to respond to them.

AIX Programming Tools and Interfaces, SC23-2304, describes the programming environment of the AIX Operating System and includes information about operating system tools that are used to develop, compile, and debug programs.

AIX TCP/IP User's Guide, SC23-2309, describes the features of TCP/IP and shows how to install and customize the program. It includes reference information on TCP/IP commands that are used to transfer files, manage the network, and log into remote systems.

AIX Technical Reference, SC23-2300 (Vol. 1) and SC23-2301 (Vol. 2), describes the system calls and subroutines a programmer uses to write application programs. This book also provides information about the AIX Operating System file system, special files, miscellaneous files, and the writing of device drivers.

AIX VS FORTRAN Reference, SC23-2050, describes the FORTRAN programming language as implemented on AIX RT, AIX PS/2, and AIX/370. This book

describes all of the standard features of VS FORTRAN as well as the enhanced functions and capabilities incorporated into IBM AIX VS FORTRAN.

AIX VS FORTRAN User's Guide, SC23-2049, shows how to develop and execute FORTRAN programs on AIX RT, AIX PS/2, and AIX/370. This book also explains how to compile and execute programs that contain sections of code written in the VS Pascal and C programming languages.

AIX VS Pascal Reference, SC23-2054, describes the VS Pascal programming language as implemented on the IBM PS/2 or RT with the AIX Operating System installed. This book describes all of the standard features of Pascal as well as the enhanced functions and capabilities incorporated into IBM AIX VS Pascal.

AIX VS Pascal User's Guide, SC23-2053, shows how to develop and execute Pascal programs on the IBM PS/2 and RT using the AIX Operating System. This book also explains how to compile and execute programs that contain sections of code written in the VS FORTRAN and C programming languages.

AIX X-Windows Programmer's Reference, SC23-2118, describes the X-Windows licensed program and provides information on X-Windows library functions, FORTRAN subroutines, protocols, and extensions.

AIX X-Windows User's Guide, SC23-2017, describes the X-Windows licensed program and shows how to start, run, install, and customize this program.

AIX PS/2 DOS Merge User's and Administrator's Guide, SC23-2045, shows how to use DOS in the AIX environment, including running DOS and AIX programs simultaneously and running AIX commands from the DOS environment. It also shows how to install the DOS Merge software and how to perform essential system maintenance activities, such as adding user accounts, backing up the file system, and setting up terminals.

AIX PS/2 General Information, GC23-2055, describes the AIX PS/2 Operating System's functions and capabilities and the product's position in the AIX family of products.

AIX PS/2 INed, SC23-2001, shows how to use the INed editor to create, access, and store files. This book also includes reference information on INed commands and a listing of INed error messages.

AIX PS/2 INmail/INnet/INftp User's Guide, SC23-2076, describes the INmail/INnet/INftp/Connect programs and shows how to use these programs to send mail to and receive mail from local and remote computer systems. This book also shows how to transfer files to and from other computer systems installed on the network.

AIX PS/2 Interface Library Reference, SC23-2051, contains information about the library of system calls available with IBM AIX VS Pascal and IBM AIX VS FORTRAN as implemented for use with the IBM AIX PS/2 Operating System.

AIX PS/2 Keyboard Description and Character Reference, SC23-2037, describes the characters and keyboards supported by the AIX PS/2 Operating System. This book also provides information on keyboard position codes, keyboard states, control code points, code-sequence processing, and non-spacing character sequences.

AIX PS/2 Text Formatting Guide, SC23-2044, describes the text formatting utilities available on the PS/2 and shows how to format text with NROFF and TROFF. This book also shows how to use the **vi** editor to create, revise, and store files.

AIX PS/2 Usability Services Reference, SC23-2039, lists and describes Usability Services commands.

AIX PS/2 Usability Services User's Guide, SC23-2038, shows how to create and print text files, work with directories, start application programs, and do other basic tasks with Usability Services.

AIX/370 Diagnosis Guide, SC23-2090, describes procedures and tools that can be used to define and categorize symptoms of problems that may occur during daily operation.

AIX/370 General Information, GC23-2062, describes the functions and capabilities of AIX/370 and its position in the AIX family of products.

AIX/370 Planning Guide, GC23-2065, describes the functions and capabilities of the AIX/370 Operating System and lists the requirements for all supported hardware and software. This book also includes information to assist with planning for installation and customization of the operating system.

Installing and Customizing the AIX PS/2 Operating System, SC23-2290, provides step-by-step instructions for installing the AIX PS/2 Operating System and related programs. This book also shows how to customize the operating system to suit the user's specific needs and work environment.

Installing and Customizing the AIX/370 Operating System, SC23-2066, provides step-by-step instructions for installing the AIX/370 Operating System and related programs. This book also shows how to customize the operating system to suit the user's specific needs and work environment.

Managing the AIX Operating System, SC23-2293, describes such system-management tasks as adding and deleting user IDs, creating and mounting file systems, backing up the system, repairing file system damage, and setting up an electronic mail system and other networking facilities.

Using the AIX Operating System, SC23-2291, shows the beginning user how to use AIX Operating System commands to do such basic tasks as log in and out of the system, display and print files, and set and change passwords. It includes information for intermediate to advanced users about how to use communication and networking facilities and write shell procedures.

Other Publications

The following IBM publications are useful references for the AIX/370 system administrator:

VM/XA System Product: Administration, SC23-0353, provides information on how to manage your system. Topics include setting up virtual machines for accounting, error recording, and CMS batch. It also includes material on setting up the programmable operator, redefining command privilege classes, defining and managing saved segments and moved saved systems, tuning the system as well as reference information on the VM/XA SP monitor.

VM/XA System Product: General Information, GC23-0362, introduces and describes the features of the IBM VM/XA SP program product. It provides information for installation managers and system programmers.

VM/SP System Product: General Information, GC20-1838, introduces and describes the features of the IBM VM/SP program product. It provides information for installation managers and system programmers.

VM/XA System Product: Virtual Machine Operation, SC23-0377, provides a task-oriented source for virtual machine operations. In step-by-step format, it describes the procedures and commands used to perform each virtual machine task.

VM/XA System Product: Planning, GC23-0378, is for system programmers and administrators who plan for the installation of a VM/XA SP system. It contains information on preparing statements for the system directory and on planning storage allocation.

VM/SP System Product: Planning, SC19-6201, is for system programmers and administrators who plan for the installation of a VM/SP system. It contains information on preparing statements for the system directory and on planning storage allocation.

VM/XA System Product: System Product Interpreter Reference, SC23-0374, provides statement syntax, format, and usage notes for using Restructured Extended Executor (REXX), the command language for writing procedures to run under VM/XA SP System Product Interpreter, the command processor for CMS.

VM/SP System Product: Interpreter Reference, SC24-5239, provides statement syntax, format, and usage notes for using Restructured Extended Executor (REXX), the command language for writing procedures to run under VM/SP System Product Interpreter, the command processor for CMS.

VM/XA System Product: CMS User's Guide, SC23-0356, provides information on using CMS.

VM/SP System Product: CMS User's Guide, SC19-6210, is for general VM/SP users. It contains information describing the interactive facilities of VM/SP and includes examples that demonstrate how to use the system.

VM/XA System Product: CP Command Reference, SC23-0358, is a reference manual for users who run systems such as OS, OS/VS, DOS, DOS/VS, VSE, CMS, and networking systems in virtual machines under VM/XA SP. It describes all CP commands.

VM/SP System Product: CP Command Reference, SC19-6211, is a reference manual for class G users who run systems such as OS, OS/VS, DOS, DOS/VS, VSE, CMS, and networking systems in virtual machines under VM/SP. It describes all class G CP commands, which are these commands available to general users.

VM/XA System Product: Interpreter User's Guide, SC23-0375, discusses and provides examples of how to use Restructured Extended Executor (REXX), the command language for writing procedures to run under VM/XA SP System Product Interpreter, the command processor for CMS.

VM/SP System Product: Interpreter User's Guide, SC24-5238, discusses and provides examples of how to use Restructured Extended Executor (REXX), the command language for writing procedures to run under VM/SP System Product Interpreter, the command processor for CMS.

System/370 Extended Architecture Principles of Operation, SA22-7085, provides a detailed definition of the machine functions performed by System/370 Extended Architecture.

System/370 Principles of Operation, GA22-7000 provides a detailed definition of the machine functions performed by System/370.

VM/SP: Remote Spooling Communications Subsystem Networking Program Reference and Operations Manual, SH24-5005, provides information on installing and operating the RSCS program product. It presents an overview of RSCS and includes operation procedures.

VM/SP: Remote Spooling Communications Subsystem Networking Planning and Installation, SH24-5057, contains a description of how to plan for and install RSCS and includes migration considerations for current RSCS customers.

Table of Contents

Figures

Tables

1.0 Chapter 1. Introduction

Subtopics

1.1 Contents

1.2 About This Chapter

This chapter describes the AIX/370 Operating System and some of its capabilities. It defines the major responsibilities of the system administrator, including MBCS considerations, and explores the tasks that should be performed after the AIX/370 Operating System is installed.

1.3 AIX/370 Overview

AIX/370 is part of the AIX family of products, which includes the AIX PS/2 and AIX/RT. AIX/370 is an IBM operating system that is compatible with UNIX System V and the 4.3 Berkeley Software Distribution (4.3BSD). The AIX/370 system is a transparent, distributed UNIX operating system allowing multiple computer users to be linked together, and implements a transparent, distributed environment for installations that run IBM Virtual Machine (VM). With the AIX/370 operating system, machine boundaries are essentially invisible to users and application programs; therefore, users can access files even though those files are not stored on their own computers.

A ***TCF cluster*** is a group of computers operating under the AIX Operating System with the Transparent Computing Facility (TCF). One or more AIX/370 computers or AIX PS/2 computers can function as a cluster. Members of the cluster communicate via a local area network (LAN) or S/370 channel-to-channel adapter (CTCA). The user interacts with the machines in a cluster as if they were a single computer, even though the actual files, disk space, and memory resources being used may be distributed all across the cluster. The AIX/370 and AIX PS/2 products with the TCF function are transparent, distributed operating systems which are compatible with one another.

The AIX Operating System features Multibyte Character Set (MBCS) support. For information on MBCS, refer to "MBCS Considerations" in topic 1.5 in this chapter as well as the *Guide to Multibyte Character Set (MBCS) Support*.

Subtopics

1.3.1 Transparency

Transparency allows users to issue the same commands to access data regardless of where it is stored. The data may be stored on any site no matter what computer the user is logged in to.

There are actually five types of transparency in the AIX/370 environment: data transparency, process transparency, name transparency, location transparency, and semantic transparency.

Subtopics
1.3.1.1 Data Transparency
1.3.1.2 Process Transparency
1.3.1.3 Name Transparency
1.3.1.4 Location Transparency
1.3.1.5 Semantic Transparency

1.3.1.1 Data Transparency

Data transparency is the ability to access and control file data using the same system calls or commands from anywhere in the cluster. For example, to copy a local file, the **cp** command is used. To copy a remote file to either a local or remote location, the same **cp** command is used.

1.3.1.2 Process Transparency

Process transparency is the ability to execute and control tasks on any site in the cluster, regardless of where the user program is currently executing. The same system calls and commands are used and do not depend on the location of the process. In other words, local and remote processes may be created and controlled, and they may cooperate with each other as if they were on the same machine.

1.3.1.3 Name Transparency

Name transparency allows the user to access an object (a file, device or process) from any site in the cluster. For example, a user can view **/u0/marianne/glossary** from any site in the cluster by using the transparent name, **/u0/marianne/glossary**. Name transparency requires that there be a name for an object which uniquely identifies that object, regardless of where the name is used in the cluster.

1.3.1.4 Location Transparency

Location transparency allows the user to access an object without having to know the site at which the object is stored. This type of transparency is possible when an object's location is not encoded in its name. For example, **/u0/marianne/glossary** may have been a file on the cluster site named **eyore** last week but is a file on the site **pooh** this week. The user is not required to know that the file was on either **eyore** or **pooh**.

1.3.1.5 Semantic Transparency

Semantic transparency allows the same command to provide the same service at all times and from all cluster sites. For example, if the **grep** command is issued from different sites, the results are the same.

1.3.2 File Systems

In the AIX/370 Operating System, the distributed file system is organized to support the transparent nature of the cluster. When given the proper path name to a file or directory, the AIX/370 system makes the file available, as needed, regardless of its storage location.

A TCF cluster presents a single, tree-structured naming facility to users on every cluster site. There is a single root (/) directory for the entire distributed system. Each cluster site has at least two file systems: a root file system and its own local file system. The AIX/370 Operating System also allows file systems to be replicated. The system administrator can set up cluster sites which have copies of the file systems. There are primary copies and non-primary copies (backbone and secondary). Primary, backbone, and secondary replicated files are automatically updated by the AIX/370 Operating System. Every replicated file system has one primary copy site where a commit operation may update a file. Other backbone (exact copies) and secondary (subset copies) then asynchronously update their copies. The root file system is a replicated file system maintained on all cluster sites and contains system files critical to normal, general cluster operation but not specific to individual cluster sites. In contrast, the local file system contains system files that are specific only to an individual cluster site, or system files that experience a high rate of update.

Note: Other UNIX operating systems have normally referred to the replicated root file system and the local file system as the root file system.

For example, the root file system contains directories such as **/usr/bin** which contain executable binary files, **/usr/adm** which contains administrative files, and **/lib** which contains some portions of the C compiler, along with many of their associated files. The local file system contains directories such as **/dev** for devices, **/usr/spool** and other site-specific files such as **unix** which contains the AIX kernel. The local file system must be mounted on to the replicated root file system.

In addition to the root file system and the local file system, each cluster site may also contain user directories and files.

To illustrate the concept of clusters, assume the existence of a sample cluster called **Five-acre Wood**. This cluster is comprised of eight sites called **eyore, kanga, owl, piglet, pooh, rabbit, roo,** and **tigger**. Each site communicates through a LAN. Since standard UNIX applications expect standard names for some files which AIX places in each site's local file system, AIX uses symbolic links and <LOCAL> aliases to link standard names of the root file system to cluster-site specific names in the local file system. For example, the root file system has a **/unix** file which is a symbolic link to the **<LOCAL>/unix** (**/eyore/unix** on site **eyore**, **/pooh/unix** on site **pooh**, and so forth, on any other site in the cluster).

1.4 Considerations for the System Administrator

The administrator should be familiar with the following concepts:

Command syntax of the Bourne shell and the C shell. For information refer to the Bourne shell **sh** and the C shell **csh** entries in the *AIX Operating System Commands Reference*.

File systems, which are discussed in Chapter 2, "Distributed File Systems" in topic 2.0.

File protection, which is discussed in Chapter 3, "File System Backup" in topic 3.0.

The wide-range general user AIX commands, such as

cp copy

ls list contents of directory

For information on these and other commands, refer to the *AIX Operating System Commands Reference*.

A file editor, such as **vi**.

All of the system administration commands such as

mount mount a file system

fsck file system check

mkfs make a file system

fsdb file system debugger

adduser add, delete, or change user or group information

recmstr recovery daemon that calls **primrec**

primrec a utility that performs user-level reconciliation for replicated file updating

minidisks add or delete minidisks or file systems.

For information on these and other commands, refer to *AIX Operating System Commands Reference* and *AIX Operating System Technical Reference*.

New processing commands such as

onsite run a command on a specified cluster site

fastsite find the least-loaded site

loads display the load average of each site

site get site database information or display current partition.

For information on these and other commands, refer to the *AIX*

Operating System Commands Reference.

Configuration software, including the use of configuration files suc as **/etc/ports** and commands such as **stty** (set terminal options). For information on **ports** and **stty**, see the "File Formats" section of the *AIX Operating System Technical Reference*. For information on AIX clusters, refer to *Installing and Customizing the AIX/370 Operating System* and the "Tailoring the User Environment" section of *Managing the AIX Operating System*.

For teleprocessing issues related to **uucp**, **connect**, and **INftp**, as well as information on INnet and INmail, refer to *Using the AIX Operating System, Managing the AIX Operating System,* and *AIX/370 Planning Guide*.

AIX input conventions that explain line editing and how to interrup running programs. For information on these topics, refer to *Using the AIX Operating System*.

The administrator must understand how peripheral devices work. Peripheral devices include terminals, Personal Computer AT and Personal Computer XT workstations running DOS with AIX Access for DOS Users, and IBM Personal Computers running a terminal emulator.

The administrator should consider the following points:

System Us

- Which users can be logged on
- How the system will be used.

Involves privileges and login issues.

System startu

- Checking file integrity (**fsck**)
- Tailoring the initialization sequence (**inittab**)
- Running processes
- Choosing the most efficient time to run processes

System shutdow

- Scheduling system down-time (**makemotd** and **wall**)
- Establishing the shutdown procedure (**shutdown**)

User access to the system (accounts and **adduser**).

- Specifying telnet destination
- Specifying which users get access to the system (create and remove accounts and specify privileges)
- Specifying the password file (**passwd**)
- Specifying the group file (**group**)
- Handling problems and messages

Adding new file system

- Determining the location of files (which disks and on which sites)
- Determining whether files have been replicated onto multiple sites (and if so, where)

Site configuratio

- Setting up or changing system defaults
- Adding terminals
- Control issues
- AIX PS/2 (with TCF) as a front-end processor
- Handling problems

Personal Computer use, configuration, and management

- Setting up file protections
- Defining the user environment
- Using AIX Access for DOS Users or terminal emulation
- Using RS-232, Ethernet, or Token-Ring

Tailoring the environmen

- Adapting prototype files that control the environment to needs of the installation
- Understanding performance tuning
- TCP/IP **/etc/hosts** and configuration files.
- Adding a new PS/2 (or PS/55) For information, see *Installing and Customizing the AIX PS/2 Operating System*.

Updating the syste

- Placing programs and other applications in a separate subdirectory on the root, such as the **/lbin** subdirectory. Before the system is replaced, making a copy of this entire subdirectory. This allows easy replacement of the system.
- Keeping a record of the changes made to the system.
- Inspecting the disk on which AIX/370 is to be installed and saving whatever files need to be kept.

1.5 MBCS Considerations

The system administrator on an AIX System that supports MBCS should be familiar with the material in the *Guide to Multibyte Character Set (MBCS) Support*. This book is delivered as part of the AIX document set which accompanies AIX/370 Version 1.2.1. It contains information essential to an understanding of the way the operating system handles multibyte character sets and the files created using them.

In particular, the system administrator will be involved with the details of which interactions with the operating system can and cannot be performed by entering multibyte characters. To some extent, an administrator is unavoidably cast in the role of trainer; it will be his or her job to clean up the errors made by ill-informed users, and the easiest way to avoid costly and time-consuming cleanup efforts is to anticipate problems that may occur.

In general, MBCS permits almost any interaction which could be handled in singlebyte Roman characters to be performed in multibyte characters. There are significant exceptions to this rule, as noted below.

Items which CAN be expressed in multibyte characters are:

file name

the contents of text file

path name

aliases for shell scripts and command names (C Shell users only

comments and literal strings in a C program

Items which CANNOT be entered in multibyte characters are:

sitenames (the names of cluster hosts

names of remote machine

user login name

user password

group name

names of environment variable

pathname delimiter

the contents (stanzas) of system attribute file

the coding in a C progra

telephone numbers and dialer strings

All of the above must be entered in ASCII characters only.

Note the following:

Filenames and pathnames can be entered in Japanese characters, but th

pathname delimiter must be entered as the traditional ASCII slash (/).

Only C shell users can create Japanese aliases for their commands Bourne shell users must enter command names and shell script names in ASCII.

The comments and literal strings in a C program can be in Japanes characters, but the coding itself must be in ASCII.

A telephone number can be written in Japanese characters if it is par of a text file, but must be in ASCII if it is to be passed to a modem.

All system attribute files (like **/etc/ports**) must be in ASCII, but a user could write a program in C, and that program could process its own attribute file with Japanese contents.

The names of environment variables must be in ASCII characters but th values to which they are set can be Japanese strings.

These interactions can become quite complex on a system which includes both Japanese users and users operating in the singlebyte Roman characters of a European language. Other information which may be helpful includes the sections entitled "Working in a Japanese Locale" in *Using the AIX Operating System* and "How to Write Programs for an International Environment" in *AIX Programming Tools and Interfaces*.

Subtopics
1.5.1 Creating Commands for Japanese Users
1.5.2 Stanzas in Attribute Files

1.5.1 Creating Commands for Japanese Users

There are a number of ways in which Japanese users can interact with AIX in Japanese characters. One way is for a Japanese user operating under the C shell to alias frequently-used command names with Japanese names; however, option flags must be entered in ASCII characters. A Japanese user operating under the C shell or the Bourne shell can accomplish the same thing by linking a command name to any string of Japanese characters; however, this again requires option flags in ASCII. Another way for a Japanese user to interact with AIX involves the C shell user. By practicing commands in ASCII to discover which flags are used most often, the user can alias the entire ASCII string (commands and flags) to some Japanese name. Many users may find these tasks beyond their technical ability and, even if they succeed, the result is a different set of Japanese names for each user. The simplest approach is for system administrator to determine a set of links and aliases that best suit the needs of the general user. Each user is then set up with an established set of commands. The standard collection of Japanese names is gathered into a single directory, and the path name of that directory is placed at the beginning of each user's path statement. The standard aliases are given to each user in the **.cshrc** file. Each user can modify and expand the set, but at least all users start with the same base set of commands.

Note: Roman and Japanese characters are available in single and in double widths, and the system does not treat them the same way. For example, if a user names a link with single-width Roman or Japanese characters and then tries to invoke it with double-width characters, the shell does not recognize the name. Double-width Roman cannot be given to the shell in place of ASCII. For more information, refer to *Using the AIX Operating System*.

1.5.2 Stanzas in Attribute Files

The stanzas in an attribute file usually contain only ASCII characters. Japanese characters can be used for the values in stanzas of an attribute file, but this must be done carefully. As long as all processes that use the values operate in Japanese locales and use the same character code in files, there is no problem; however, any process that operates in a non-Japanese locale or uses a different code in files is unable to use the values. An example would be a cluster where all users are operating in Japanese locales, their files and directories are named with Japanese characters, and the system administrator operates in a Japanese locale. Mount points within the cluster for remote file systems are named with ASCII characters and the names of the remote file systems to be mounted are also in ASCII. The remote file system can be mounted because the system administration process doing the mounting is operating in a Japanese locale, which recognizes both Japanese and ASCII characters. However, if the remote file system name contains extended ASCII characters used in some European locales, the mount operation will be unsuccessful because a Japanese locale cannot recognize the extended characters. The system administrator must also be careful about the names of file systems within the cluster that are to be mounted outside the cluster. If an administrator operating in a non-Japanese locale on a remote cluster tries to mount a file system with a Japanese name, the attempt fails because non-Japanese locales cannot recognize Japanese characters. File systems that are to be mounted outside the cluster should be named with ASCII characters.

Some clusters are devoted to users who operate in the same language and locale; others support more than one locale. Many clusters may support a single locale but have extensive contact with systems supporting other locales. Some organizations use global clusters that span continents and languages. The system administrator must determine how frequently attribute files are accessed by users operating in non-Japanese locales. Japanese characters can usually be used freely for the values in attribute files if the cluster is devoted exclusively to Japanese users. However, if the cluster has users who operate in non-Japanese locales or if it has contact with remote systems that operate in non-Japanese locales, ASCII characters should be used for values.

1.6 After the Installation of AIX/370

Installing AIX/370 is not an administration task; however, the installer may need instructions and information from the administrator to perform the installation. AIX/370 installation is explained in *Installing and Customizing the AIX/370 Operating System*. Even though actual administration tasks start after AIX/370 has been installed, the administrator should have a clear understanding of all cluster configuration and planning.

Subtopics
1.6.1 Time Synchronization in the Cluster
1.6.2 Administration Tasks

1.6.1 Time Synchronization in the Cluster

The clocks should be synchronized so that all systems and all cluster sites are consistent. For more information, refer to the "Additional Systems Management Topics" chapter in *Managing the AIX Operating System*.

1.6.2 Administration Tasks

The following tasks should be performed by the administrator after AIX/370 has been installed:

Define AIX/370 for the environment by configuring the hardware an software system files.

Initialize all direct access storage devices (DASD) that are to b used by AIX/370.

Configure AIX/37

- Define the system layout
- Define new devices
- Make new file systems
- Define the paging, minidisk, swapping, and user areas to the system.

For more information on these topics, refer to *Managing the AIX Operating System*.

1.7 Starting AIX/370

Under normal circumstances, a System/370 machine is not powered down at night as one often does with smaller machines like the PS/2 (or PS/55). The system runs all night. On many systems, the need for a multi-user, interactive operating system is minimal during off hours, so AIX/370 may be de-activated during this period. In many installations, this is the time devoted to running large batch jobs, and a simpler operating system like CMS may be in use.

VM, which supports both of these operating systems, is typically up and running 24 hours a day. Starting up AIX/370 again requires logging on to one of the VM virtual machines and bringing AIX up on that particular virtual machine. Meanwhile, other virtual machines may also be running on the same hardware supporting CMS, other copies of AIX/370, or any of a number of other possibilities.

The actual procedure for bringing up AIX/370 varies according to the arrangement designed by the VM System Administrator. The customary procedure is for that particular virtual machine to come up initializing AIX/370 automatically. In this case, AIX normally comes up in multi-user mode immediately.

If the system comes up automatically initializing CMS, you need to IPL (Initial Program Load) the system disk containing the AIX/370 system files.

In either case, when the system successfully initializes from the system disk, it displays the copyright notice and the **login** prompt. At this point, AIX/370 is ready for normal use.

1.8 AIX/370 Interactions With VM

AIX/370 operates as a guest under several different versions of IBM VM. VM provides AIX/370 a limited number of services beyond the basic support of dividing a single physical machine into multiple virtual machines. This section addresses various topics to consider when establishing an interface between VM and AIX/370.

VM provides great flexibility with respect to device configuration. *Installing and Customizing the AIX/370 Operating System* provides information on the correct configuration of a virtual machine to run the distribution version of AIX/370. AIX/370 contains a default system that allows for expansion by using the **devices** command. It is up to the administrator to determine whether or not to continue to use the conventions set by the distribution for device configuration. For more information regarding interactions with VM, contact your VM administrator or refer to your VM reference library.

Subtopics

1.8.1 Using the IUCV Driver

The Inter-User Communication Vehicle (IUCV) is a communication facility that allows a program running in a virtual machine to communicate with other virtual machines, with a CP system service, and with itself. With IUCV, it is possible to connect an AIX/370 guest machine to VM Transmission Control Protocol/Internet Protocol (TCP/IP) or to connect an AIX/370 guest to another AIX/370 guest.

Note: You will need to add the following two lines to the VM USER DIRECT file for each guest communicating with IUCV:

```
IUCV ANY PRIORITY
IUCV *CSS PRIORITY MSGLIMIT 255
```

For specifics on tuning, refer to "Tuning Main Storage at VM Level" in topic 6.4.2.

Subtopics
1.8.1.1 AIX/370 to TCP/IP Connection
1.8.1.2 AIX/370 to AIX/370 with IUCV Driver

1.8.1.1 AIX/370 to TCP/IP Connection

The connection between AIX/370 and TCP/IP can be made when both are on different machines, or when both reside on the same machine.

VM/SP TCP/IP Profile for Different Machines: The connection is made between AIX/370 and TCP/IP on a different machine, and PVM service machines are used as a bridge. The profile of the VM/SP TCP/IP service machine has to be modified to include a DEVICE and a LINK statement for the IUCV path. The following example shows the definition where the local TCP/IP service machine on node YKTVMDPA will connect to PVM, which in turn will connect to the YKTVMXA1 machine, userid AIXDP2 (which is running AIX/370). At the end of the example, the **start** command begins the operation of IUCV. This is an undocumented feature of VM/SP TCP/IP Version 1.2.

```
* Reduce pool sizes to allow running in 4M virtual machine
envelopepoolsize 50
databufferpoolsize 50


DEVICE IUCV1 IUCV YKTVMXA1 AIXDP2 PVM A
LINK PVM IUCV 1 IUCV1

port     * values from rfc 900, "assigned numbers"
    23 tcp intclien          * the telnet port

home     * the local host's internet addresses
    192.9.214.2 PVM

gateway
*  network      first hop     driver  packet size  subn mask  subn value
    192.9.214   =            PVM             2000     0

start iucv1
```

AIX/370 Definition of IUCV Driver: To use the IUCV driver, you need to complete the following tasks. Note that step 5 must be completed for connection through the PVM service machine. The other steps need to be completed whether or not the connection is through PVM.

1. The **/etc/master** and **/etc/system** files must have appropriate entries to define IUCV. In the **/etc/master** file, check the following entry (the **niucv** stanza should already be there) and make sure that it has the unique major device number.

   ```
   niuc:  type = dev
       routines = init,ioctl
       character=TRUE
       devhdrreqd=TRUE
       devtable=TRUE
       subunits=TRUE
       prefix = niuc
       major = 31
       maxminor = 4
   ```

   In the **/etc/system** file, add the following entry for one interface and make sure it has a unique address. Other interfaces, for example **niuc1**, **niuc2**, and **niuc3** could be added the same way).

```
niuc0:
    driver = niuc
    address = 460
```

2. A new kernel must be made using **newkernel**.

3. An appropriate pair of internet numbers must be created in the **/etc/hosts** file corresponding to the AIX/370 machine and the VM/SP TCP/IP service machine.

   Using the example given above in the profile of VM/SP TCP/IP service machine, the **/etc/hosts** file should have entries like the following:

```
192.9.214.1         aix370
192.9.214.2         vmtcpip
```

4. Verify that each AIX/370 guest has an entry for IUCV in the VM USER DIRECT file. For more information, refer to "Tuning Main Storage at VM Level" in topic 6.4.2.

5. Execute the **iuconfig** command to connect the AIX/370 machine to the VM/SP TCP/IP service machine. This is placed in the **/local/rc.tcpip.local** file. The following example matches the VM/SP TCP/IP profile above, where the IUCV connection goes to the TCP/IP machine on YKTVMDPA through PVM.

```
iuconfig niuc0 PVM YKTVMDPA TCPIP
```

6. An appropriate point to issue the **ifconfig** command must be added to the **/local/rc.tcpip.local** file. By using the example given above for the **/etc/hosts**, the following command should be used.

```
ifconfig niuc0 aix370 vmtcpip
```

VM/SP TCP/IP Profile For Same Machine: This configuration is intended for connection made when VM TCP/IP and AIX/370 reside on the same machine. The examples below show the profile of the TCP/IP service machine which defines how the TCP/IP service machine on YKTVMDPA machine connects to userid AIXSP4 (running AIX/370) on the same machine.

```
DEVICE IUCV1 IUCV IGNORED IGNORED AIXSP4 A
LINK AIXSP4 IUCV 1 IUCV1

port     * values from rfc 900, "assigned numbers"
    23 tcp intclien          * the telnet port

gateway
*  network      first hop    driver    packet size        subn mask  subn va
    192.9.214   =            AIXSP4    DEFAULTSIZE            0

start iucv1
```

AIX/370 Definition of IUCV Driver: The only difference between making a connection through PVM and going directly through IUCV is the **iuconfig** command as shown below.

```
iuconfig niuc0 TCPIP ignored ignored
```

1.8.1.2 AIX/370 to AIX/370 with IUCV Driver

The IUCV driver can also be used to connect an AIX/370 guest machine to another AIX/370 guest machine.

AIX/370 Definition of IUCV Driver: Generating new AIX/370 kernels should be done on both AIX/370 systems. This can be accomplished by following steps 1 and 2 beginning on page 1.8.1.1.

1. An appropriate pair of internet number should be added to the **/etc/hosts** file:

   ```
   192.9.214.1        aixmach1
   192.9.214.2        aixmach2
   ```

2. Verify that each AIX/370 guest has an entry for IUCV in the VM USER DIRECT file. For more information, refer to "Tuning Main Storage at VM Level" in topic 6.4.2.

3. An **iuconfig** command should be presented in the **/etc/rc.tcpip** file.

   For connection made through PVM, enter the following commands:

   ```
   iuconfig niuc0 PVM NODE2 MACH2  ----- for machine 1
   iuconfig niuc0 PVM NODE1 MACH1  ----- for machine 2
   ```

   where **MACH2** and **MACH1** are the VM userids of AIX/370.

   Note: PID 1.4 Release of PVM or above should be used.

   For direct IUCV connection, enter the following commands:

   ```
   iuconfig niuc0 MACH2 xxxxxx yyyyyy --- for machine 1
   iuconfig niuc0 MACH1 xxxxxx yyyyyy --- for machine 2
   ```

   where **MACH2** and **MACH1** reside on the same node and **xxxxxx** and **yyyyyy** could be anything as long as they are specified the same way in both systems. For more information on the **iuconfig** command, refer to the *AIX Operating System TCP/IP User's Guide*.

4. An appropriate entry for **ifconfig** command should be added in both systems' **/local/rc.tcpip** files. Using the example given above, the entry in both system should be

   ```
   MACH1 : ifconfig niuc0 aixmach1 aixmach2
   MACH2 : ifconfig niuc1 aixmach2 aixmach1
   ```

1.8.2 VM Requirements for AIX/370

The *AIX/370 Planning Guide* describes the minimum requirements for AIX/370. This includes the correct level and version of VM in addition to physical and virtual resources. For information on minimum requirements, refer to the *AIX/370 Planning Guide*.

1.8.3 Using the AIX/370 Console

Installing and Customizing the AIX/370 Operating System provides information on using the 3270 console. Refer to the appendix section of the manual for details on help screens and key mapping.

1.8.4 Logging AIX/370 Device Errors

When a hardware error is detected by AIX/370 software, it logs this error to the VM-based error logging support. In the case of disk support, AIX/370 uses the CP level disk-reading services to actually retry any fault detected in accessing a disk. Other devices have errors logged using the VM-provided interface.

1.8.5 VM Performance Tuning

There are many performance considerations to be addressed when AIX/370 operates under VM. One of the critical concerns in the execution of any operating system is the amount of physical memory available to the system. In the case of VM, this would translate to the amount of physical memory available to the AIX/370 virtual machine.

VM/SP can execute a virtual machine in ***virtual equals real*** (**V=R**) memory mode and ***virtual equals virtual*** (**V=V**) memory mode. VM/XA has an additional memory mode of ***virtual equals fixed*** (**V=F**). For more information on memory modes, refer to "Performance Considerations" in topic 6.4.

Note: **V=F** machines are only available on 3090 Series E processors with PR/SM installed. In addition, VM/XA SP Release 1 enhancement for Multiple Preferred Guests which is available on a program update tape (PUT) must be installed unless VM/XA Release 2 is used.

1.8.6 Attaching Disks to a Virtual Machine

There are three different ways that a DASD device (a disk) can be attached to an AIX/370 virtual machine. They are:

Dedicating the devic
Full disk VM minidis
Partial disk VM minidis

Each of these options has slightly different characteristics. Before addressing the differences between the three options, a short discussion on the administration strategy of the disk space for AIX/370 follows.

In AIX/370, as in most UNIX V systems, there is a mechanism to partition a physical device logically into several subpieces called ***minidisks***. A minidisk is a contiguous portion of disk space. VM also provides a mechanism to partition physical disks into logical subpieces called VM minidisks. AIX/370 does not require that all disks be partitioned with an AIX/370 minidisk structure (a bootable disk must be, however). AIX/370 can create one or more AIX/370 minidisks on a VM disk. To create one minidisk, AIX/370 simply puts the minidisk structure on the VM device. When creating multiple minidisks, AIX/370 queries VM about the space available on the VM device and partitions that space into minidisks.

An administrator has the option of choosing how to allocate and maintain free space for the overall system and the AIX/370 system in particular. If the VM administrator wants to let the AIX/370 administrator allocate the disk space, then the DASD devices would be appropriately attached to the virtual machines and the AIX/370 administrator will use the AIX minidisks mechanism to manage the disk space. Similarly, if disk space needs to be tightly allocated among various virtual machines on a physical machine, then using the VM minidisk mechanism may be desirable (though the additional AIX/370 minidisk structures can still be used in addition). The attachment of disk devices differs among the three cases. If a disk is dedicated, it can only be used by the virtual machine to which it is dedicated. This has an advantage in that it eliminates errors when two virtual machines are trying to use the same portion of the same disk. Dedicating a disk also has a performance advantage in that it is not necessary to do as much translation of disk channel programs (such as adding offsets). Furthermore, the disk scheduling algorithms function optimally because they are really aware of all the device action on the disk. A VM full minidisk has the performance advantages of the dedicated disk (only if not shared). However, it is also possible to set up a second virtual machine from which backup personnel are able to do full image DDR backups of the volume. A VM minidisk (not a full one) can also be shared but has the additional overhead to access in that VM must add a base offset to all block numbers. Much more significantly, the AIX disk I/O scheduler is not able to order activity based on real knowledge of where the disk heads are really located.

Performance can be tuned depending upon memory modes and the way in which a disk device is attached to an AIX/370 guest virtual machine. These performance factors are determined by the needs of the configuration and the preference of the administrator.

1.8.7 Installation/Maintenance System (I/M System)

AIX/370 is distributed with a small single site system which is used for emergency maintenance and installation. Administrators are encouraged to make sure that the system is backed up when new service materials are applied to it. A virtual machine should be created which has all the devices of the real AIX/370 system but also boots the I/M system. This will allow operators to easily boot the I/M system so it can be used if problems arise in the base system.

1.8.8 Console Logging

The system regularly prints, on a defined line printer, all output that has been written to the system console. The line printer (at virtual address 501 in 370 mode or 00A in XA mode) can be spooled not to print under normal conditions, but in critical circumstances it can be used to determine what occurred.

1.8.9 Printing in a TCF Environment

The queuing system running on a TCF cluster prints files on the device specified in the **/etc/qconfig** file. There is only one **/etc/qconfig** in the TCF cluster, so any user logged on to any site in the cluster may access any printer in the cluster. The queue and device entries in the **/etc/qconfig** file determine the selection of printer.

A print job may be done on a VM spooled printer or on a printer attached to a PS/2 (or PS/55) workstation by specifying the queue to that printer. If the node in the printer queue stanza is the name of an AIX/370 site, it prints using the CP spooling system. When the node is an AIX PS/2 site, the print job is done on a workstation printer.

Note: AIX/370 supports only printers attached to the AIX/370 virtual machine. All printing is done by spooling the files to VM and then printing under the control of the VM spooling system.

The **/etc/qconfig** file is part of the root replicated file system. The queue files and copies of the files to be printed are kept in the **/usr/lpd/qdir** and **/usr/lpd/stat** files, which are the symbolic links into the local filesystem of the TCF site serving the printer.

Subtopics

1.8.9.1 Printing Procedure

There are several steps in the AIX/370 printing procedure:

1. The user selects a printer by executing the **print** command and specifying the **argname** parameter corresponding to the printer to be used. The **argname** parameter is found in the **etc/qconfig** file. Available options explained in the *AIX Operating System Commands Reference* are specific to backend operation.

2. The queuing daemon, **/etc/qdaemon**, handles the print request and starts the backend program specified in the **/etc/qconfig** file.

3. The backend communicates with the **lp** device driver through standard **open**, **write** and **ioctl** system calls. AIX/370 provides several **ioctl** options to change the spooling parameters and printing attributes of the printer.

4. The **lp** device driver communicates with VM and provides the print file with the correct VM spool file attributes.

5. VM processes the print file according to the spooling parameters and printing attributes provided. Refer to the *VM/SP System Product CMS User's Guide* or the *VM/XA System Product CP Command Reference* for information on the CP SPOOL and CP TAG commands.

Files spooled to a network communication virtual machine, normally running RSCS, require TAG information with the spool file to specify the remote destination where the file is to be printed.

1.8.9.2 Printer Interfaces

The AIX **print** and **lp** columns in Table 1-1 describe the options used for providing the VM spooling information. The **lp driver** column describes the corresponding CP commands issued from the device driver.

Table 1-1. AIX/370 Printing Process

print	**lp backend**	**lp driver**	**CP COMMAND**
	-tag="tag"	PRTsetting	CP TAG DEV dev tag
	-route=userid	PRTto	CP SPOOL dev TO userid
	-class=class	PRTclass	CP SPOOL dev CLASS class
-nc=copies		PRTcopy	CP SPOOL dev COPY copies
	-form=form	PRTform	CP SPOOL dev FORM form
-to=user		PRTdist	CP SPOOL dev DIST user
-tl=name		PRTfname	CP SPOOL dev NAME name
		PRTpurge	CP CLOSE dev PURGE
	-translate=table	PRTxlate	
	-trtable=table	PRTxlate	
	-fcb=fcb	PRTfcb	CP SPOOL dev FCB fcb
	-char=char	PRTchar	CP SPOOL dev CHARS chars
	-ascii	TASCII	
	-dest=dest	PRTdest	CP SPOOL dev DEST dest
	-spool="....."	PRTspool	CP SPOOL dev "....."

Note the following information regarding the options seen in Table 1-1:

The **lp -route** option is different from the **print -t** option. The **-route** option causes the file to be spooled to the userid following the option. The **print -to** option changes the distribution code for this print file.

The **print** options are passed to the backend through the queue entry for the **-nc=copies** and **-tl=name** options.

The **-tl=name** option changes the filename of the spool file to be printed.

The **-translate=table** option allows you to use a different translate table.

The **-ascii** option allows you to print in ASCII on an IBM 3800 printer using the VM Print Services Facility.

The **-dest=dest** option is not supported under VM/XA SP, and the backend prints an error message when used with this system.

The **-spool** option allows use of a more general spool command. After a file is transferred to CP, the **lp** device driver resets the CP SPOOL information to the following:

```
SPOOL OFF CLASS A COPY 1 FORM OFF DIST OFF
```

This command also resets the **to=user** information. If any TAG has been set, the TAG information is deleted.

1.8.10 Using the VM Spooling System

In a local VM printing environment printers are organized in print classes. Each print class defines printer options like the following:

```
locatio
paper siz
security leve
prioritie
```

Starting a printer in a specific class is done at VM IPL time or by the VM operator later. Only print files of the designated class are printed. The printer for the AIX/370 virtual machine is spooled to a default class (CLASS A) during installation unless the class is changed by the directory or in the PROFILE EXEC for this virtual machine.

To print a file on a remote printer, the print file is spooled to a network communication service machine (normally running RSCS) and the information on the printer destination for the file is provided in the TAG information. The commands for remote printing are:

```
CP SPOOL PRT dev TO userid CLASS class FORM form
TAG DEV PRT nodeid system
```

The remote printer must be defined in the VM RSCS network. It can be attached to another VM system or even to a remote MVS or DOS/VSE system. The print request is started on a system printer for the remote machine.

For more information on printing, see your VM administrator or refer to one of the following manuals: *VM/XA System Product: Virtual Machine Operation*, *VM/XA System Product: Planning*, *VM/XA System Product: Administration*.

Subtopics

1.8.10.1 Customizing Printing from AIX/370

Printing from AIX/370 may be customized in the following ways:

The **/etc/qconfig** file may be customized on the system level. The system administrator defines the print queues in the **/etc/config** file. Several print queues which serve different printers or the same printer with different options may be configured.

The **lp1** stanza defines a printer which the user accesses by entering

```
print -lp1 filename
```

Requests to each printer are handled by the backend defined in the lpdev1 stanza. By adding additional print queues and providing the backend with different options for each queue, the backends can provide printers with a variety of options.

The following is an example of a stanza in the **/etc/qconfig** file used to set up a class C printer.

Note: The options specified in **/etc/qconfig** file have preference over the user options.

```
lp4:
 argname = -lpC
 node = carmen
 device = lpdev4

lpdev4:
 file = /dev/lp1
 align = TRUE
 header = always
 trailer = never
 feed = never
 backend = /usr/lpd/lp -fw=80-fl=64 -class=c
```

To send a file to this printer, enter

```
print -lpC filename
```

The following is an example of a stanza in the **/etc/qconfig** file used to set up a remote VM printer. This set up routes the request through RSCS to NODE and prints on the printer **prtname** with PRIORITY=50.

```
lp5:
 argname = -prtname
 node = carmen
 device = lpdev5

lpdev5:
 file = /dev/lp1
 align = TRUE
 header = always
 trailer = never
 backend = /usr/lpd/lp -fw=80 -fl=64 -route=rscs -tag='NODE
PRTNAME 50'
```

To send a file to this printer, enter

```
print -prtname filename
```

The spool option for a printer may be customized on the system leve using the **devices** command. The system administrator can add or change the virtual printers defined for each AIX/370 site by defining new virtual printers or changing spool information for a virtual printer. For more information on the **devices** command, refer to *Installing and Customizing the AIX/370 Operating System*.

The spool options for a printer may be changed in the PROFILE EXE when the AIX/370 virtual machine is installed. For example, to change options for a remote printer, enter

```
CP spool prt dev TO userid CLASS class FORM form
TAG DEV PRT nodeid [printerid]
```

To change options for a local printer, enter

```
CP SPOOL PRT  [dev]  CLASS . . .
```

The VM administrator can change the directory entry for the AIX/370 virtual machine. For example, to change the spool printing to a 1403 and set class=C, enter

```
SPOOL 00E 1403 C
```

The user can route print requests to a specific printer.

1.8.10.2 ASCII to EBCDIC Translation

Some printers need a special print chain and a special translation table to print AIX files. The translation is done in the **lp** device driver.

AIX/370 provides a set of pre-defined ASCII/EBCDIC translation tables for printing ASCII files with National Language Support (NLS) characters. The system administrator defines the table to be used for his system by setting the environment variable NLPRINT when **/etc/qdaemon** is invoked in **/etc/rc**.

The NLPRINT variable contains the name of a country-specific translation table. If NLPRINT is not set, the built-in translation table in the **lp** device driver is used. If users want to specify a translation table, they can use the **translate** option to the **lp** backend.

The new option **ascii** permits printing on a 3800 printer in line mode and directly in ASCII. No translation to EBCDIC is necessary.

1.8.10.3 Banner Pages

The queuing system provides options to define burst pages, AIX headers and trailers in addition to the standard VM banner and trailer pages. For more information refer to the header, trailer, and feed parameters in the **/etc/qconfig** file.

When header and trailer parameters are defined, it is possible to print an AIX banner and trailer page beside the standard VM banner page, with information about the time printed, the time queued, the last time the file was modified, and who printed the file and to whom it was delivered.

1.8.10.4 AIX Print Queues

The status of an AIX file can be checked by entering

```
print -q
```

1.8.10.5 VM Print Queues

The local VM print queue can be checked as soon as the file is spooled to CP by entering the following command on the AIX console for the site where the VM printer is attached:

```
cpcmd q prt all
```

If the file is sent to a remote printer, there are CP commands for checking the status of the file. The RSCS network sends messages back to the virtual machine indicating where the spool file is sent. The CP SMSG facility allows you to query remote printers, and responses are sent as CP messages to the virtual machine running AIX/370.

1.8.10.6 Modifying CP SPOOL Files

As long as the spool file is on the local VM, the AIX superuser can use the **cpcmd** command and issue CP commands.

If the file is spooled to a remote printer, there are limited possibilities for changing or purging the spool file. Refer to the SMSG command in the *VM/SP System Product: CMS User's Guide, VM/XA System Product: CP Command Reference*, or in related RSCS manuals.

The normal AIX user can only query the CP SPOOL for information by entering

```
cpcmd query prt . . .
```

or

```
cpcmd query virtual prt
```

Refer to the CP QUERY command for valid options.

1.8.10.7 Queue Requests in TCF Clusters

Because a queue is related to a node in the TCF cluster, requests for queues currently not in the cluster are rejected and the following message appears:

```
Site sitename is down. Cannot access queue queuename
```

When printing, the spool file is always copied into the **/local/spool/qdaemon** file so that the server site always has access to the print file, even if the storage site for the original file is no longer available.

2.0 Chapter 2. Distributed File Systems

Subtopics

2.1 Contents

2.2 About This Chapter

Distributed file systems are an integral part of the TCF cluster architecture. This chapter shows the system administrator how to create, monitor, and maintain file systems in the AIX/370 distributed processing environment.

2.3 File Systems and Administration Responsibilities

This section lists some of the tasks and commands important to the system administrator regarding file systems.

Among the administrator's primary responsibilities are:

Allocating disk space for file system
Creating file systems on logical disk
Making file space available to user
Monitoring file space usag
Backing up files to guard against their loss in the event of use error, or system or disk failures
Ensuring that file systems remain in a consistent stat
Managing data replication

Some of the AIX/370 system management commands for dealing with file systems are:

mkfs Makes a file system of a specified size on a specified logical disk.

df Reports the amount of used and free space on a copy of a file system.

mount Attaches a file system or a copy of a file system to the cluster-wide naming structure so that files and directories residing on the file system can be referenced.

umount Removes a file system or a copy of a file system from the cluster-wide naming structure, making the files and directories on it inaccessible.

fsck Checks file systems and repairs inconsistencies.

backup Backs up files in backup format to a backup medium, such as magnetic tape or diskette.

restore Copies back files created by the **backup** command.

minidisks Adds, deletes, shows or changes the characteristics of a minidisk.

fsdb Permits interactive exploration and patching of a file system. A tool for the expert or student of file system structure, rather than a maintenance utility.

mdrc Provides access to user created minidisks by configuring these minidisks at system startup.

Most of the utilities for dealing with a file system actually operate on the special file which serves as an interface to the file system (for example, **/dev/diskux22**).

2.4 Understanding the AIX/370 Directory Structure

A TCF cluster presents a single, tree-structured naming facility to users and applications on every cluster site. There is a single root directory (/) to the entire cluster.

The user and the application program's view of object names in an AIX/370 Operating System is analogous to a single, centralized, UNIX operating system environment; all objects appear with globally unique names in a single, uniform, hierarchical name space. Each object is known by its path name in a tree. When given a proper path name to a file or directory, the TCF cluster makes the file or directory available as needed, regardless of its storage location.

A path name is a sequential list of directory names separated by slashes and may end with a file name. If the path name begins with slash (/), it is called a ***full path name*** or ***absolute path name***. Full path names specify the desired directory or file regardless of the current working directory. ***Relative path names*** specify the path from the current working directory. In Figure 2-2 in topic 2.4.1, if the current working directory is **/u2** and the subdirectory **jane** is to be referred to, then **/u2/jane** is the full path name and **jane** is the relative path name.

Note: A path name can also have implicit hidden directory references and references through symbolic links to the **<LOCAL>** system. A path name that does not have these references is a ***context-free path name***. It is a name that evaluates to the same object under all cases. These topics are discussed in greater detail in "Symbolic Links and <LOCAL> Aliases" in topic 2.4.2.1.

Subtopics

2.4.1 File Systems

A file system is a complete directory structure, including a root directory and any subdirectories and files underneath the root directory. It is a hierarchically-structured organization of files and directories residing locally or remotely on a contiguous section of secondary storage (disk or diskette). For example, Figure 2-1 illustrates a hierarchical directory.

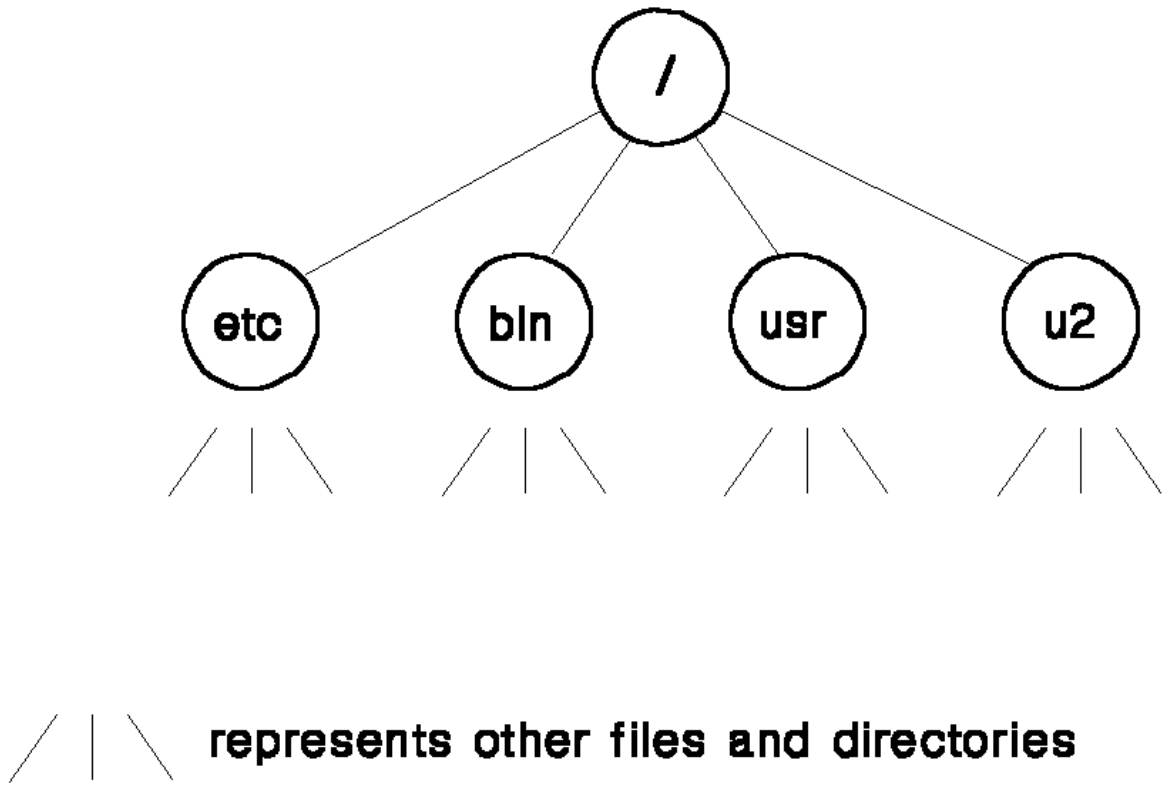

Figure 2-1. Sample Naming Hierarchy

Each file system consists of three main parts:

- The ***superblock*** is a file system header which uniquely identifies the file system and contains its size, type, and current space utilization statistics.

- The ***inodes*** are file and directory descriptors. They store such information as the file's owner, group, and permissions, and the pointers to the blocks which hold the file's data.

- The ***data blocks*** are used to hold file and directory data. A data block can be either ***allocated***, which means it is in use and is pointed to by some inode in the file system, or ***free***, which means it is available for use when someone creates a new file or alters an existing file.

Note that each file system has its own collection of data blocks. For this reason, one file system can be out of free space, while another file system can have an abundance of space. There is no way to give one file system's free space to another file system without restructuring the disk partitions and rebuilding the file systems from scratch.

File systems are created with the **minidisks** command. This command takes a section of disk storage which is referred to by a device node in the **/dev** directory and initializes a file system onto that disk. That is, it writes out a superblock and a set of inodes, and places all of the data blocks into the file system free list. To make the files in a file system available for use, the file system must be mounted onto the global system naming hierarchy. This is done with the **mount** command. The **mount** command takes as arguments a disk device node (like the **minidisks** command) and a directory. The **mount** command tells AIX/370 to take the file system which is on the specified disk device, and splice it onto the naming hierarchy at the specified directory. Any directory which has a file system mounted to it is called a ***mount point***.

Assuming that **/dev/diskuser** refers to a device which has a file system containing user files, Figure 2-2 reflects what the naming hierarchy looks like after the command **mount /dev/diskuser /u2** is issued.

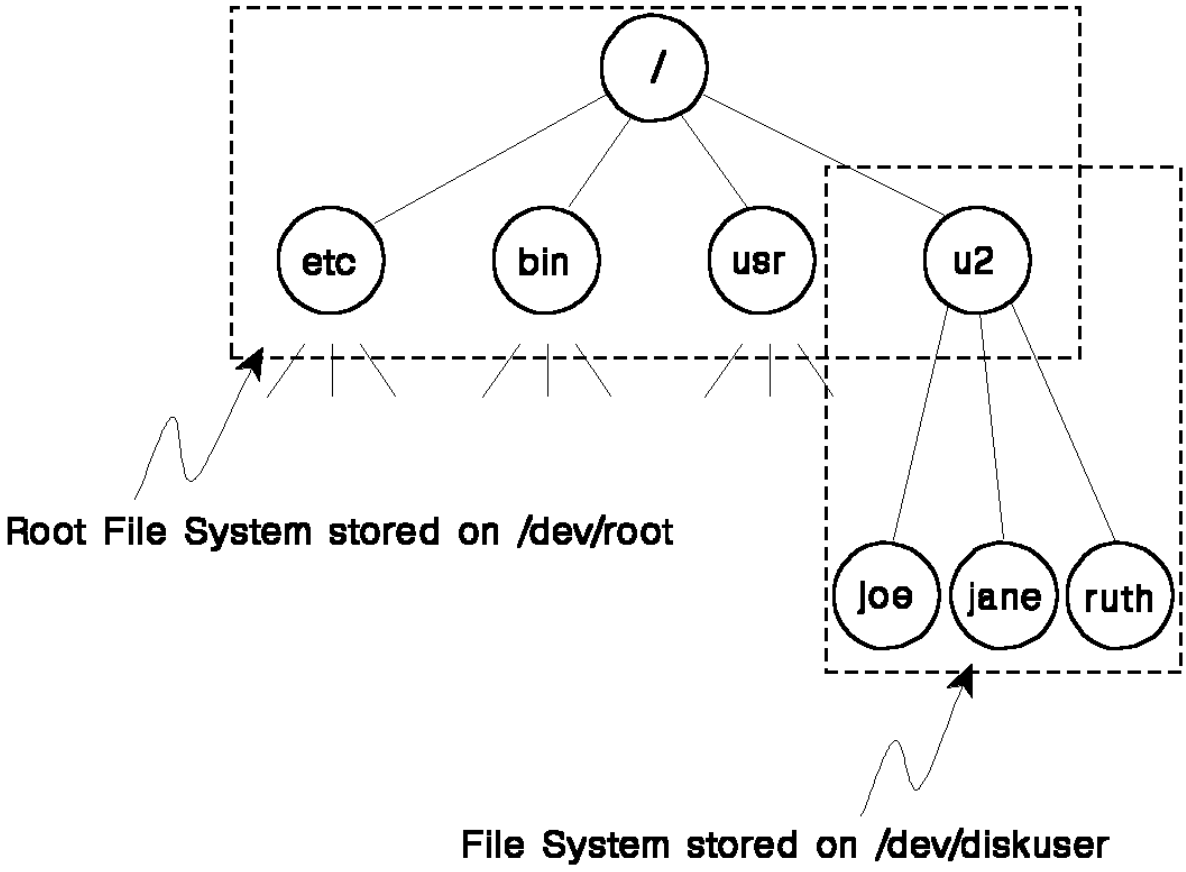

Figure 2-2. Sample Naming Hierarchy After Mount Command

2.4.2 Classes of File Systems

An understanding of the classes of file systems that can exist in the AIX/370 distributed processing system is important. AIX/370 defines file systems as being either replicated or non-replicated. A file system that is replicated can have full or partial copies of itself on other sites in the cluster. This replication allows cluster sites to run independently while presenting a uniform view to all users. AIX/370 uses replication to allow each site to operate on its own, if necessary. It also allows users to continue to access files if the site that normally stores them becomes unavailable, or if the media links that connect the cluster sites together should fail.

A file system that is non-replicated only has one copy of itself on one site. A non-replicated file system is different from a replicated file system that only has its primary copy. A primary copy of a replicated file system is allowed to have other copies of itself created with **minidisks**. On the other hand, I/O to non-replicated file systems can be more efficient than to replicated file systems. This is discussed in greater detail in the section "Replicated File System Copies" in topic 2.4.5.

Subtopics
2.4.2.1 Symbolic Links and <LOCAL> Aliases

2.4.2.1 Symbolic Links and <LOCAL> Aliases

A symbolic link is a special mechanism used to create a second logical pathname to a file or directory. As seen in Figure 2-5 in topic 2.4.3, certain directories in the replicated root file system are symbolic links to other directories in the **<LOCAL>** file system. For example,

/dev is a symbolic link to **<LOCAL>/dev**
/tmp is a symbolic link to **<LOCAL>/tmp**
/unix is a symbolic link to **<LOCAL>/unix**.

Hard Links and Symbolic Links: There are differences between a conventional hard link and a symbolic link. A hard link permits multiple names to be created for a single file (not a directory). For example, if the command **ls -li /bin** is entered, it can be seen that the files **mv**, **cp**, and **ln** all have the same inode number. These files are different names for the same program and they are linked together with a hard link. A hard link is only possible between files on the same file system. A symbolic link permits multiple names to be created for a file or a directory and can be used to link files and directories on different file systems. In other words, a symbolic link can cross file system boundaries.

In Figure 2-3 and Figure 2-4, the slash (/) after a name indicates a directory. As seen in Figure 2-3, the **<LOCAL>** file system is the one named for a particular cluster site. In this example, the site name is **/tigger**. When logged into **tigger**, a user may access the **/dev** directory on the replicated root file system. When the user changes to this directory, only the devices for **tigger** are seen because **/dev** points symbolically to the **<LOCAL>/dev** directory (in the root of the local file system) which is **/tigger/dev**.

In the following figures, directories and symbolic links to directories are indicated by a trailing slash (/).

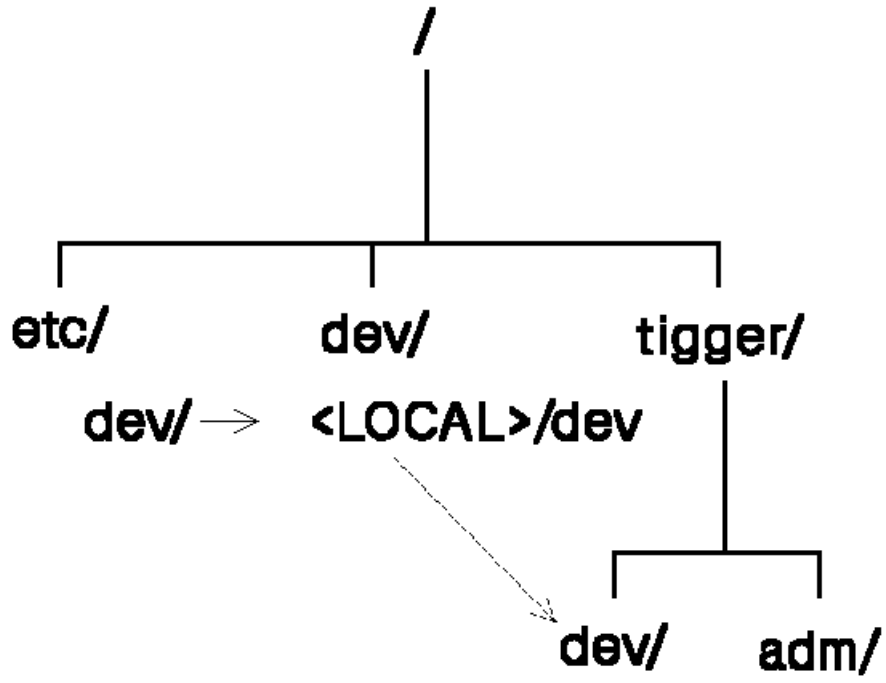

Figure 2-3. One Example of Symbolic Links and Local Aliases

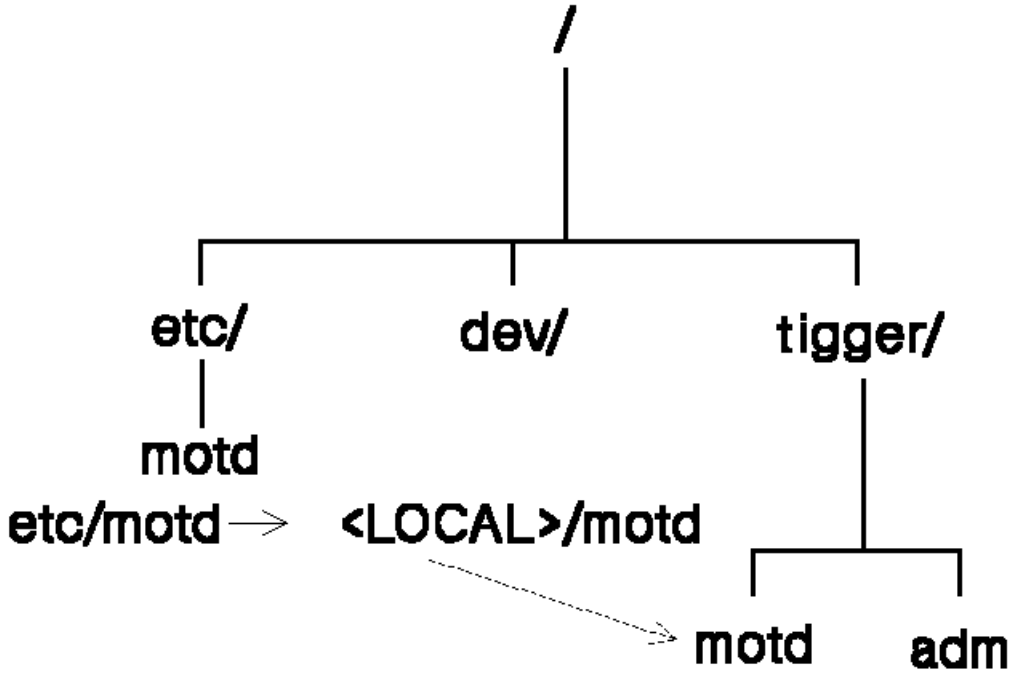

Figure 2-4. Another Example of Symbolic Links and Local Aliases

The <LOCAL> Alias: A separate **<LOCAL>** file system is mounted onto the replicated root file system for each site in the cluster. The **<LOCAL>** alias is used with symbolic links to create an exception to name transparency. This exception to name transparency can allow a given cluster site to operate differently from the rest of the cluster. The **<LOCAL>** alias is translated into different strings on different cluster sites (for different processes). When **<LOCAL>** is the first component of the destination name for a symbolic link, it is replaced with its alias string, normally **/<sitename>**. Each process has a **<LOCAL>** alias manipulated with the **getlocal** and **setlocal** system calls. The **<LOCAL>** alias is preserved by the **fork** and **exec** system calls. For information about these system calls, refer to the *AIX Operating System Technical Reference*.

AIX/370 uses the **<LOCAL>** alias and symbolic links to link site-specific files and directories into their standard names. For example, in Figure 2-4, **/etc/motd** is the standard path name for the file that might contain the message-of-the-day file, but since this file is site-specific, it is really stored in the **<LOCAL>** file system.

2.4.3 File Systems on Cluster Site

Each cluster site is pre-configured to have at least the root file system and the local file system in a root file system hierarchy. Once the system administrator sets up these file systems on the cluster site, the user file systems can be created and added to the cluster site. The file systems seen from a particular cluster site before joining the cluster are the replicated root file system, the **<LOCAL>** file system, and the user file system (or systems).

Root File System: The root file system contains files that are critical to the normal operation of all sites in the cluster and are not specific to any individual cluster site. The root file system is replicated and maintained on all sites in the cluster.

The major directories and files found in the root file system are:

etc/ System administration programs and data files (includes profile and login files)

bin/ AIX programs (includes **sh** and **csh**)

usr/ System administration, **bin**, **lib** files (no user files)

lib/ Portions of the C compiler and major C libraries

unix Symbolic link to **<LOCAL>/unix**, the system kernel

dev/ Symbolic link to **<LOCAL>/dev/** devices

tmp/ Symbolic link to **<LOCAL>/tmp/** temporary files

lbin/ Place to put local programs, files and directories that need to be globally available within the cluster

Local or <LOCAL> File Systems: The */<sitename>* directory is the mount point where the cluster's **<LOCAL>** file system is mounted. The **<LOCAL>** file system contains directories and files that are an extension of the root file system, but are site dependent or have a high update rate. The following list shows some of the major files and directories in the local file system:

inittab System initialization table

dev/ Devices for the cluster site

adm/ Administration files for the cluster site

tmp/ Temporary files for the cluster site

spool/ Spool files for the cluster site

unix Kernel for the cluster site

uts/ Kernel configuration directory for cluster site

ddi/ Files that describe system devices

ports File containing terminal characteristics

lpd/ Files of commands used to send data to system printers and collect accounting statistics on printer usage

filesystems File system characteristics for the site

The **<LOCAL>** file system is mounted on to the root file system at the mount point called */<sitename>*.

User File Systems: User file systems are files containing users' home directories and associated directories and files. These files can be stored on one cluster site or on several cluster sites. The user file systems are usually mounted onto the root file system. The system administrator sets up the user file systems and can give them any names.

Notes:

1. User accounts do not belong under **/usr** since it is part of the root file system.

2. There may not be any user file systems on the local cluster site.

Figure 2-5 shows the root file system, the **<LOCAL>** file system (*/<sitename>*), and the user file system, as viewed from a particular cluster site (called *<sitename>*).

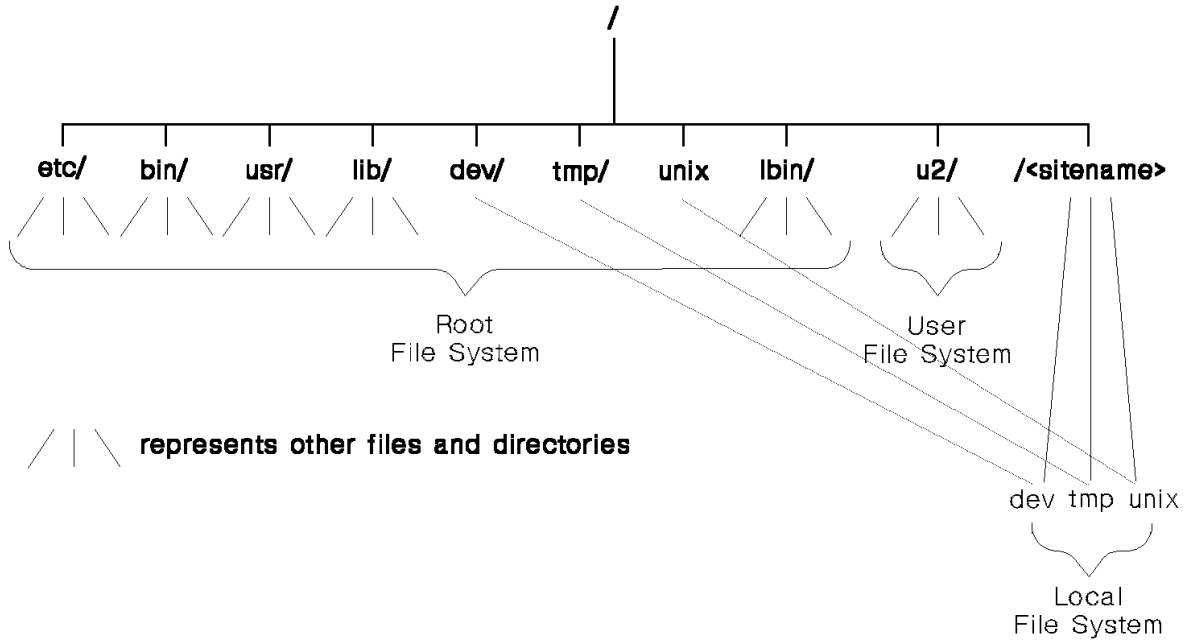

Figure 2-5. All File Systems Viewed from One Cluster Site

2.4.4 Replication Principles

Replication, in simplest terms, provides system support for a file or directory to be duplicated in several locations, or on several sites. On almost any operating system, a file can be replicated by simply creating a second copy. However, TCF replication goes far beyond this simple concept of replication.

In TCF replication, the system makes sure that all copies are up to date with respect to each other. The user need only change one copy. In fact, most of the time the user need not be aware that the files are replicated. The primitives operate similarly on replicated and non-replicated files.

TCF replication also allows the system administrator to specify a minimum and maximum number of copies that users keep. Between these limits, users can control the number of copies kept by the system on their behalf. The implementation of file replication permits applications which are not replication knowledgeable to function properly with the users controlling replication factors external to the application.

Note: With TCF replication, all copies of a file or directory have the same name from the system's point of view. All copies are logically manipulated as a unit. If a replicated file is deleted, all copies of the file are effectively deleted at that time. If one of the copies is to be deleted, use the **store** or the **chfstore** command to reduce the number of copies rather than the **del** command.

To control concurrent updates to the replicated files, TCF replication uses a primary copy-based synchronization scheme. In this context, the term synchronization is defined slightly differently from the common definition. Normally, an operating system synchronizes access to a file when two users on the same system are trying to write the file. In this case, the system needs to serialize these operations to make sure each operation sees a consistent file from the system's point of view.

There is another level of synchronization involved in TCF replication. Consider the following example:

User A on Site S1 is accessing File F

User B on Site S2 is accessing File F

Assuming S1 and S2 are different, the following cases can occur:

Case 1: File F is not replicated and is stored only on Site S1.

If Sites S1 and S2 are communicating through TCF, both Users A and B can operate on the file using normal UNIX and AIX primitives. Once each has saved a version of the file, both users will see each other's changes. If Sites S1 and S2 are not communicating through TCF because, for example, the LAN is down, only User A has access to the file. User B must make changes at a later time.

Case 2: File F is replicated and is stored on both Sites S1 and S2.

If Sites S1 and S2 are communicating through TCF, the replicated file appears to both users as if it were a single file. As long as the primary site is on the net, all normal UNIX and AIX primitives behave in the expected manner. Both Users A and B see each other's changes.

If Sites S1 and S2 are not communicating through TCF at this time because, for example, the LAN is down, the system needs some strategy to either prevent the creation of inconsistencies or resolve inconsistencies at a later time.

To resolve inconsistencies, TCF replication uses a primary copy mechanism to synchronize the changes made by two users. The primary copy mechanism designates one copy of the file system as the primary copy. The user only makes changes to the primary copy. The system guarantees that all changes are first done to the primary copy and then the changes are propagated to the other copies of the file system.

The copies of a file system fall into three classifications:

Primary — The copy where all changes are originally done. This copy stores every file in the file system. Updates to any file in a file system can only occur when the primary copy is available for modification. This guarantees that no inconsistent updates are performed to the data in the file. Note that a file system's primary site is the cluster site which maintains the primary copy of that replicated file system.

Backbone — A read-only copy. A backbone stores all files in the file system independent of the replication control factor on the files. However, since this copy is not the primary, all changes to data in the file system are propagated to the backbone copy from the primary copy.

Secondary — This copy contains only a subset of the files and directories contained in the primary copy. It, too, is read-only. It is useful in cases where specific files are not needed on every cluster site. For example, text processing files are large and use a large amount of space. Only certain sites may elect to have copies of text-formatting files (such as **troff**), so certain sites may exclude these files from its secondary copy. All changes to files in a secondary copy are propagated from the primary copy of the file system.

To determine when a copy of a file is out of date, TCF replication uses commit counts and ***high*** and ***low water marks***. Each file in a replicated file system has a commit count which is basically a version number and is stored in the inode of the changed file. Every time a file is modified at the primary site, the commit count is increased for the entire file system. High and low water marks track the state of a copy of the file system regarding changes to its files, as follows:

high water mark — Indicates that this copy is aware of changes up to the indicated commit count.

low water mark — Indicates that this copy incorporates all changes up to and including the indicated commit count.

These two numbers are always the same on a primary copy of a file system. When a secondary or backbone copy of a file system has been out of communication with the primary copy for some period, the system determines if it is up to date by checking if its low water mark is equal to the low water mark of the primary copy. If two copies of a file have different commit counts, the copy with the larger commit count is considered more recent.

2.4.5 Replicated File System Copies

Replicated file systems are file systems that have copies on multiple cluster sites. Each copy is mounted on the same mount point. Files and directories within different copies of the mounted replicated file system are referenced with the same pathnames. During normal operation, the user is not aware of the existence of multiple copies.

The significance of types of replicated file system copies should be understood in light of file system classifications. Replicated file systems can be system or user file systems. A ***system-replicated file system*** contains files and directories that are used by all sites in the cluster and are not specific to any one cluster site. The root file system is a good example of a system replicated file system. The root file system contains files critical to the normal operation of all sites in the cluster. A ***user-replicated file system*** contains files and directories that are important only to specific users or applications.

For example, consider **/user/projectdoc** to be a file system that contains project documentation files. The main copy of this file system (the primary copy) is stored on cluster site **tigger**, and other copies reside on **eyore** and **pooh**. This file system is replicated on more than one cluster site to allow the **projectdoc** files to be referenced (for example, read or printed) even when the primary copy is not available. All replicated file systems, whether system-replicated or user-replicated, must have a primary copy. They may optionally have one or more backbone or secondary copies as well.

A file in a replicated file system is only stored on a secondary copy of the file system if it has been specially marked for storage on the secondary copy. For system-replicated file systems, this is accomplished with the **chfstore** command. For user-replicated file systems, the **store** command is used instead. Once a file or directory has been marked for storage on a secondary site copy, it is automatically updated as needed by the kernel to maintain consistency with the primary copy.

For example, suppose a file system **/u2** has a primary copy on site **tigger**, a backbone copy on site **roo**, and a secondary copy on site **pooh**. Also suppose that a file called **newdoc** is to be added to this file system. The primary copy and backbone copies are automatically updated by the kernel, but the secondary copy is not. In order to instruct the system to store **newdoc** on the secondary copy, enter the command **store +pooh /u2/ruth/newdoc**. Then, whenever **newdoc** is updated, the secondary copy on **pooh** is changed also.

Subtopics
2.4.5.1 The Significance of Replicated File Systems and Copies

2.4.5.1 The Significance of Replicated File Systems and Copies

To continue with the example from the previous section, assume that **/u2/ruth** is in a user-replicated file system. The primary copy is on site **tigger**, a backbone copy on site **roo**, and a secondary copy on site **pooh**. Ruth creates a file in **/u2/ruth** called **newdoc** and stores it on **pooh**. The file **newdoc** is now stored on all three copies of the file system. The cluster consists of the sites **tigger**, **roo**, **pooh**, and **eyore**.

If Ruth wants to change the file **newdoc**, the system accesses the file from **tigger**, because **tigger** contains the primary copy of **/u2/ruth/newdoc** and only the primary copy can be modified. If, however, Ruth only wants to print (read) the file, then when she logs into **roo**, the system accesses the copy on **roo**. When she logs into **tigger**, the system accesses the copy on **tigger**. If, however, Ruth logs into **eyore**, the system accesses a copy of the file remotely from either **tigger**, **pooh** or **roo** because **eyore** does not have a copy already available.

If **tigger** is not available, the primary copy of Ruth's file system is not available. Even though the primary copy is not available, Ruth can still log into **roo** or **pooh** or any other site in the cluster, and read (or print) her files. Ruth can not change her files, however, because the primary copy of her file system is not available. Ruth can create a temporary copy by copying **newdoc** into another file, for example, **newdoc2** in the **/tmp** directory. When **tigger** is available again, Ruth can copy **/tmp/newdoc2** to **/u2/ruth/newdoc**.

Replication can be viewed as a way to reduce cluster communication load and increase performance by making files locally accessible for those files that are read more than they are written. Replication can also be viewed as a means of increasing the availability of a file. Notice how Ruth's file is still available when **tigger** is not available. Furthermore, replication provides increased reliability for critical data. Should there be a storage media failure which impacts one of the file's copies, the other copies can be used to recover the file. Replication, however, should not be used as an alternative to file system backup. A file inadvertently deleted or changed is deleted or changed in all copies of the file systems.

2.4.6 File System Structure

The structure of a file system is completely storage-device independent. A file system is represented by exactly the same combination of bits on any direct access device. The only restriction on moving a file system from one device to another is that the destination device must have enough space to accommodate it. The AIX/370 and AIX PS/2 file system structures are not identical because of the different ways the hardware architectures represent binary data. File system utilities such as **fsdb** and **fsck** must be run on an AIX/370 site when accessing an AIX/370 file system, and on an AIX PS/2 site when accessing an AIX PS/2 file system. Generally, these utilities should be run directly on the cluster site which stores the file system. This file system difference also means that a file system can be moved only between cluster sites of the same type.

AIX/370 and AIX PS/2 allocate disk space in units of 4096 bytes each; however, AIX/370 commands (for example, **mkfs** and **fsck**) expect block numbers or block counts as arguments to be specified in terms of 1024-byte blocks. This is done to provide a uniform way for users to view file sizes when moving between systems in the AIX family.

A file system is implemented as a section of mass storage. It is composed of a header or superblock, a small set of file descriptors called inodes, and a large number of data blocks. These three terms are described in detail in the following section.

Superblock: The second block of every file system is called a superblock. It is the most critical part of the file system containing information about every allocation (or de-allocation) of a block in the file system. The superblock resides in block one (counting from zero) of the logical disk. Following are some of the important fields in the superblock:

s_magic A number indicating that the file system has been initialized with the **mkfs** command.

s_fpack The volume-ID of the disk pack. This ID is optionally created by **mkfs**. For more information, refer to the **mkfs** entry in the *AIX Operating System Commands Reference*.

s_gfs The global file system (gfs) number. This is a number set up by the system administrator that is between 0 and the **NMOUNT** system parameter. It is the unique number for the file system on the cluster. A gfs number of 0 is typically used in removable media such as diskettes, and causes the system to choose an unused gfs number in a reserved range. A gfs of 0 cannot be used on a replicated file system.

Note: Different file systems must have different gfs numbers. Replicated copies of the same file system have the same gfs number, but a different gfs pack number.

s_gfspack The gfs pack number. Each copy of a replicated file system has a unique gfs pack number. It is a number between 1 and 31 and can be set up by the system administrator to correspond to the cluster site number, if so desired. The primary copy need not be pack number 1.

s_flags File system flags. This field is used to identify the type of file system created. Each type of file system can be replicated or non-replicated. If the file system is replicated, then

s_flags indicates whether it is primary, and whether it is a system or user-replicated file system.

s_fsize The total size (in blocks) of the file system.

s_isize The number of blocks reserved for inodes (refer to "Inode File Descriptors"). Included in **s_isize** are also the two blocks used for the boot block (**block0**) and the superblock (**block1**). There are eight inodes in each inode block for AIX/370 and AIX PS/2.

s_tfree Total free blocks. This is useful for reporting the number of available disk blocks. For more information, refer to the **df** command in the *AIX Operating System Commands Reference*.

s_free[0] This is the block number of the head of the freelist.

s_free[1] - s_free[s_nfree -1] These are block numbers for free blocks.

s_tinodes Total free inodes. This is useful for reporting the number of available inodes. For more information, refer to the **df** command in the *AIX Operating System Commands Reference*.

s_inode A list of free inodes.

s_time The time and date of the last disk update of the superblock.

s_fstore The **fstore** value is only meaningful in secondary copies of a replicated file system. It identifies which subset of the files should be stored in this copy of the file system.

s_hwm, Commit counters. These are special counters used for replicated file
s_lwm, systems. The commit operation incorporates all data changes made to a
s_llst file into a new version of the file. When a file is modified, a commit operation is automatically performed when the file is closed.

The **s_hwm** field (high-water mark) contains the commit sequence of the last commit that has occurred on the primary copy and is known to this copy.

The **s_lwm** field (low-water mark) on a primary copy is always equal to the high-water mark. On a backbone copy or secondary site, all commits with sequence numbers less than or equal to the low-water mark have been incorporated into this copy.

The difference between the high-water mark and the low-water mark indicates how closely the backbone and secondary copies resemble the primary copy and whether updated versions of files need to be propagated over to the backbone and secondary copies. System and user-level replicated file systems in AIX automatically propagate primary copies of files over to the backbone and secondary copies based on these commit counts.

Note: The primary, backbone and secondary copies must be mounted and part of the same network partition in order for file propagation to occur.

The **s_llst** field (commit list mark) indicates the commit count

for the oldest entry in the commit list. For more information about the commit mechanism, refer to "The Commit Operation and File Update Propagation" in topic 2.5.1 and the *AIX Operating System Technical Reference*.

s_cmtlst The recent commit list which contains the commit counts for the last 200 file changes known to the cluster site.

Note: Superblock information is slightly different, depending on whether the file system is replicated or non-replicated. In non-replicated file systems, the commit count information is not needed, so AIX/370 is able to store a larger free block array. For this reason, a file system cannot be easily converted from replicated to non-replicated and from non-replicated to replicated.

For more information about the superblock, refer to the **fs** entry in the *AIX Operating System Technical Reference*.

Inode File Descriptors: An inode file descriptor serves as a low level internal directory and the internal structure for managing files in the system. After the superblock, there are a number of blocks containing inodes. The specific number of inode blocks is a parameter that varies with the file system size and is specified in the superblock. There is one inode for each possible file on the file system. Up to 2,147,483,647 inodes can be allocated.
For replicated file systems, it is required that all copies have the same number of inodes. Each inode, if allocated, describes a file and contains the following type of information:

di_mode The file type and permissions. Possible file types are ordinary file, directory, block device, character device, FIFO (named pipe), and socket. For information on changing permission codes, refer to the **chmod** command in the *AIX Operating System Commands Reference*.

di_nlink The link count. A directory entry (link) consists of a name and the number of the inode that represents the file (its *i-number*). The link count specifies the number of directory entries that reference the inode. When this count goes to zero and all processes having the file open close it, the file is deleted. That is, its inode is de-allocated and its associated blocks are freed.

di_uid, di_gid The user-ID and group-ID associated with the file. The user associated with the file is considered its owner.

di_inogen The inode generation number is a count of the number of times this inode has been used for different files.

di_dflag The disk flags. The **di_dflag** data structure contains information which augments the **di_mode** field to yield certain file types. For instance, the **DIHIDDEN** flag distinguishes a hidden directory from a regular directory, the **DISOCKET** flag distinguishes a socket from a FIFO, and the **DILINK** flag distinguishes a symbolic link from a regular file.

di_size The size of the file (in bytes).

di_mtime The date the file was last modified.

di_atime The date the file was last accessed. The access time on each copy of a replicated file system is maintained independently, except when the file is modified. Under some circumstances, non-primary copies of files may have their access time set backwards. When a file is modified, its access times are set up to be the same as its modification time.

di_ctime The time the inode was created or last modified. This records the time of last modification to the inode or to the data associated with the inode and is used to determine whether it should be backed up. Changes to the inode header information (mode, owner, and group changes) cause **di_ctime** to change, but do not affect **di_mtime**.

di_blocks Number of blocks allocated to an inode.

di_cmtcnt The gfs commit sequence number for the last commit done on this inode.

di_fstore The **fstore** bits. This shows where a file is stored for replicated file systems. It is ignored for non-replicated file systems. For more information on the use of this field, refer to "fstore Values and Propagation" in topic 2.5.2.

di_version The inode version number. This shows the number of commits done to this file.

di_sbflag Small block flag. If the **di_sbflag** is non-zero, the data for the file is completely contained within the inode in the **di_sbbuf** buffer. The inode has space for 384 bytes of file data.

di_addr Block numbers of the blocks that contain the actual data in the file (or in the case of a device special file, the major, minor, and clustersite device designations.

di_sbbuf The small block buffer. If **di_sbflag** is non-zero, **di_sbbuf** contains the entire contents of the file.

For more information about inodes, refer to the "Introduction to System Management" chapter in *Managing the AIX Operating System* and also the *AIX Operating System Technical Reference*.

Data blocks: The remaining blocks of the file system are used as data blocks. These blocks contain the data stored in the files and directories, and indirect blocks that point to other data or indirect blocks for large files.

The first inode (inode 1) on every file system is an unnamed and unusable file that historically contains the addresses of bad (physically flawed) blocks. The second inode (inode 2) corresponds to a directory. That directory is the root directory for the file system. All other files in the file system are somewhere in the hierarchy below the root directory. Beyond inode 2, any inode can be assigned to any file. Similarly, any data block can be assigned to any file. There is no requirement that inodes or blocks be allocated contiguously or be assigned in any particular order.

2.4.7 Mounting and Unmounting Files

Distinct file systems are totally separate in that a file in a given file system contains only data blocks from that file system. A directory in a given file system can make references to files in the same file system (using a hard or symbolic link) or make references to files in another file system (using a symbolic link). In general, because file systems are separate, damage to one file system cannot affect any other file system. File systems can be backed up, restored, and repaired separately. However, modifications (additions, deletions, or changes) to files on a copy of a replicated file system are automatically reflected in backbone and secondary copies by AIX/370.

AIX/370 has a **mount** command which temporarily links disjointed file systems into a logical whole. Essentially, **mount** creates a mapping from one file system to another. The **mount** command sets up a mapping between the named directory and the root directory of the file system on the logical disk device. For example, the command **mount /dev/diskuser /u2** sets up a mapping between the named directory called **/u2** and the root directory of the file system on the logical disk device called **diskuser**. After a **mount** command is used, references to the named directory are translated into references to the root directory of the mounted file system, bridging the gap between the two file systems. In effect, the root directory on the mounted file system is spliced into the system-wide name space on top of the named directory. The directory on which a file system is mounted is called the mount point for the mounted file system.

This mapping has no effect on the mount point directory. It merely becomes inaccessible while the mount is in effect. The **mount** operation does not change the data on either file system. It only affects the system's name-to-file mapping mechanism. If the file system is subsequently unmounted (with the **umount** command), or if AIX/370 is taken down and rebooted, the system returns to a state prior to the mount and the mapping is lost.

The **mount** operation allows the integration of several disconnected file systems into a seemingly continuous directory structure. Most users never know that all the files in the system are not really part of a single hierarchy. The mount operation gives the administrator a very powerful tool for managing disks. A particular file system can be physically moved from one place to another on a drive, on a directory with the same name, or on a different cluster site, and users never know the difference.

If a file system is replicated, it is possible to unmount one of the available copies without affecting the other copies. The data becomes unavailable when the last copy available is unmounted. (Of course, this may not be true for secondary copies, as all files on the primary copy are not necessarily represented there.) For more information on the **mount** command, refer to the *AIX Operating System Commands Reference*.

Note: The **mount** operation is usually performed during system startup and **unmount** during system shutdown based on the **/etc/filesystems** file.

2.5 How AIX/370 Updates Replicated File Systems

This section deals with the commit mechanism and **fstore** value comparison used to maintain data integrity.

Subtopics

2.5.1 The Commit Operation and File Update Propagation

The non-primary, replicated files are kept up-to-date in an asynchronous fashion by using the commit operation and file update propagation. The commit operation incorporates all data changes made to the specified file into a new version of the file. This operation moves changed data pages and the in-core file descriptor (inode) to the disk. Changes made to a file are ordinarily made permanent when the file is closed (close includes an implied commit) and thus, an explicit commit is not usually necessary.

To simplify the protocols necessary to maintain file system consistency in a dynamic cluster environment, the system allows file modification of only one copy of the file system (the primary copy). When a replicated file is updated, the primary copy is updated first, then the backbone and secondary copies. When the primary copy is closed, a commit operation is performed. The commit operation assigns the next available commit sequence number to this change, places that sequence number in the inode, and writes out the data pages and inode. Also, the inode number is recorded in the commit list in the superblock. The primary copy site then sends out messages to the backbone sites and the secondary sites to tell them that a new version of the file has been created. Asynchronous to the writer, sites storing backbone and secondary file copies of the file system then copy the newly updated file onto their copy.

The current commit level is determined by two separate counters in the superblock. One counter is the low-water mark counter; the other is the high-water mark counter. On the primary site for the replicated file system, the low-water mark is always equal to the high-water mark. On a backbone or secondary site, all commits with sequence numbers less than or equal to the low-water mark have been incorporated into this copy.

The high-water mark contains the commit sequence number of the last commit that has occurred on the primary copy or is known to this copy. Thus, the difference between the high-water mark and low-water mark indicates how closely the backbone or secondary copy resembles the primary copy and whether updated versions of files need to be propagated over to the backbone or secondary copy.

For example, suppose the primary copy of a file system has a low-water mark of 100 and a high-water mark of 100. A change is made to the primary copy of a file. When the file is closed, the commit count is increased by 1 so the high-water mark and low-water marks on the primary copy become 101. The primary copy site tells the backbone copy site that the new file has been updated and its commit count is 101. The backbone copy site has a low-water mark of 100 and a new high-water mark of 101. Since they do not match, the backbone site propagates a new version of the file from the primary copy site and increments its own low-water mark to 101.

Sometimes the primary copy site is separated from the backbone and secondary sites. Perhaps the LAN connection is broken somehow, so that each site can no longer respond to the other. For example, suppose that there are eight cluster sites connected on an Ethernet. The connection breaks down (a repeater fails) so two partitions are formed; for instance, one partition may have 3 sites and the other may have 5 sites. Each partition has full communication between each site, but there is no communication between the two partitions. Now if each site has a backbone copy of **/u2**, but only one site (in the five-cluster-site partition) has the primary copy, then only the five-cluster-site partition has the updated primary copy and the other copies will have changes propagated to them. Since it does not have the primary copy, the three-cluster-site

partition can not have any file updates and corresponding propagation.

When the cluster sites are all united again on the same Ethernet line, the question may arise regarding how the backbone and secondary file system copies can be brought up-to-date with respect to their primary copy. The two operations for handling this situation are ***kernel-level reconciliation*** and ***user-level reconciliation***. The primary copy keeps a history of the last 200 consecutive commits (including the inode numbers). If the difference between the high-water mark and low-water mark is less than 200, the backbone and secondary copies automatically get the last 200 updates. If the difference between the high-water mark and low-water mark is greater than 200, the backbone and secondary copies automatically invoke a user-level reconciliation process for the copies to be brought up-to-date. There is a recovery daemon called **recmstr** that runs on each site and works with the **primrec** and **comlist** utilities to perform this user-level reconciliation. User-level reconciliation utilities run as a user process (not a kernel process). They check the commit counts in the inode information on the primary copy and compare them to the commit counts in the local inodes. If the commit counts are not the same, then they instruct the kernel to propagate the primary copy files over to the backbone and secondary copies and change the commit counts to be the same. For more information about these utilities, refer to the *AIX Operating System Commands Reference*.

2.5.2 fstore Values and Propagation

The **fstore** values consist of the values found in the **s_fstore** field in the superblock and the **di_fstore** field in the inode. These values are used to determine whether a secondary file system copy gets a modified file from the primary copy.

Note: The backbone copies get copies of any new or modified files no matter what the **fstore** value is.

When the superblock and inode **fstore** values are compared, a special operation is done to determine if a particular copy of a file is stored on a particular secondary file system copy. The special operation only occurs when a primary copy of the file is modified or the secondary copy of the file is being synchronized with the primary copy (for example, the secondary copy is not synchronized because it has not been mounted or it has been disconnected from the primary copy).

This special operation is a logical, bit-by-bit AND operation done between the **fstore** value in the superblock for the file system and in the inode for the file. If the result of the AND operation is non-zero, the system stores the file.

For example, suppose there is a replicated user file system called **/u2** and there are three copies of this file system: a primary copy on site **tigger**, a backbone copy on site **roo**, and a secondary copy on site **pooh**. A new file is created in **/u2** called **glossary** and the **store** command is used to indicate which sites should store this file. When the file is created, it is given a unique inode number, and the **store** command sets a **fstore** value in this inode.

The superblock **fstore** value for the secondary copy of **/u2** on **poo** has a value of 0000010000 (in binary). The **fstore** value for **glossary** (stored in its inode) is 0100011101. A logical bit-by-bit AND of the two numbers is done as follows:

```
0000010000 Superblock value for /u2
0100011101 Inode value for glossary
----------
0000010000 Result of AND operation
```

Since the result of the AND operation is non-zero, **glossary** is stored on the secondary copy on **pooh**. If the result had been zero, the file would not be stored on the associated cluster site. This AND operation is done for each file on each secondary file system copy to determine if this file should be stored on this site.

There are two sub-classes of replicated file systems: system and user. The primary difference is in the definition of how files are chosen to be replicated. Each file system has an **fstore** value for the file system which is stored in the superblock. In addition, each file has an **fstore** value which is stored in its inode. The logical AND operation of the file system's **fstore** and the inode's **fstore** determines whether a file is stored on a particular file system.

There are 32 bits in an **fstore**. Each bit can be used to represent either of the following:

A disk pack number, which is usually the site number of the fil system for a user-replicated file system.

A class of files whose value is agreed upon, as in the case of system-replicated file system.

The **fstore** values of particular files are modified as follows:

For a user-replicated file system, the **store** command takes as arguments a site name or number and a list of files. The operations performed include adding and deleting sites to the inode **fstore** value.

For a system-replicated file system, the **chfstore** command takes as arguments an octal value or a system class type and a list of files. The **chfstore** command sets the **fstore** value of the indicated files to the specified value. The association of system class names to octal **fstore** values is kept in the file **/etc/fstore**. This file initially defines the following classes names:

i386 Causes the file to be stored on all AIX PS/2 system cluster sites

i370 Causes the file to be stored on all AIX/370 system cluster sites

none Causes the file to be stored on only the primary and backbone cluster sites

all Causes the file to be stored on all cluster sites.

The system administrator can add additional classes.

The choice of creating user-replicated or system-replicated file systems should be based on the following criteria:

A file system in which files are site-specific or specific to a subse of sites is a candidate for a user-replicated file system. An example could be a project directory primary on one or more sites.

A file system in which files have cluster-wide significance would b best handled by a system-replicated file system. Examples are defining system classes of i386 and i370 for 386 and 370 binaries or 386 development tools, 386 utilities and 386 typesetting tools.

The creation of a user-replicated file system does not require any effort beyond making a replicated file system on several sites, and setting the **fstore** value of a file to the desired site numbers. The **mkfs** command is used to set the pack number to the site number when the file system is created. The **mkfs** command automatically sets the superblock **fstore** value based on the gfs pack number in each secondary copy of the file system.

The creation of a system-replicated file system should be started by deciding whether to use the default set of file system classes in the **/etc/fstore** file, or to define new classes or subclasses in this file. Then **mkfs** can be used to create a replicated file system specifying the initial **fstore** value of the appropriate class.

For example, assume the contents of **/etc/fstore** is as follows:

```
020000000000:0:none
07000:0:i386
04000:0:tooli386
```

```
  02000:0:basei386
  01000:0:dvlpi386
  00007:0:i370
  037777777777:0:all

Site 1 has the following disks (cpu=370):

  disk1, pack=1, system replicated  fstore=00077
  disk2, pack=1, user replicated    fstore=all
```

Note: The 370 stores all i370 and 386 files.

```
Site 2 has the following disks (cpu=386):

  disk1, pack=2, system replicated  fstore=00070

Site 3 has the following disks (cpu=386):

  disk1,         system replicated  fstore=00030
  disk2, pack=3, user replicated    fstore=00040
```

Note: The 386 stores 386 base and 386 development files.

The following table lists files, their **fstore** values and the sites where the files are stored:

File	**Fstore value**	**Sites**
/bin/date@/i386	00020	1, 2, and 3
/bin/date@/i370	00007	1
/bin/troff@/i386	00010	1 and 2
/usr/project/	003000	1
/usr/proj/exp	020040	1 and 3

Subtopics
2.5.2.1 File System Updates

2.5.2.1 File System Updates

Whenever a file is created, modified, or removed, the AIX/370 operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. It is helpful to understand the order in which updates are done. There are five types of file system updates involving:

Superbloc
Inode
Indirect Block
Data Block
First Free-List Bloc

Superblock: The superblock contains information about the size of the file system, the commit counters, the size of the inode list, part of the free-block list, the count of free blocks, the count of free inodes, and part of the free-inode list.

The superblock of a mounted file system (the replicated root file system and local file systems are always mounted) is written to the disk whenever the file system is unmounted or a **sync** system call is issued. For more information on the **sync** system call, refer to the *AIX Operating System Commands Reference* and the *AIX Operating System Technical Reference*.

Inodes: An inode contains information about the type of file (directory, data, regular file or special file), the number of directory entries linked to the inode, the list of blocks claimed by the inode, and the size of the file.

An inode is written to the file system on closure of the file associated with the inode when:

an **fcommit** call is executed
if the process using the file is marked for periodic commit, o
whenever a **sync** system call is issued.

A file open for writing will be marked for periodic comment if it is opened without the **O_DEFERC** flag. For more information, see the **fcommit** system call in the *AIX Operating System Technical Reference*.

Indirect Blocks: There are three types of indirect blocks - single indirect, double indirect, and triple indirect. A single-indirect block contains a list of some of the data block numbers claimed by an inode. Each one of the 1024 entries in an indirect block is a data-block number. A double-indirect block contains a list of single-indirect block numbers. Triple-indirection blocks are not used in AIX/370 or AIX PS/2.

Indirect blocks are written to the file system whenever they have been modified. More precisely, they are queued for eventual writing. Physical I/O is deferred until the buffer is needed by the system, until a **sync** system call is issued, or until a commit operation is performed on the file to which the indirect blocks belong.

Data Blocks: A data block may contain file information or directory entries. Each directory entry effectively consists of a component name and an inode number.

Data blocks are also queued to be written to the file system whenever they

have been modified.

First Free-List Block: The superblock contains the first free-list block. The free-list blocks contain a list of all blocks that are not allocated to inodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in this free-list block, a pointer to the next free-list block, and a partial list of free blocks in the file system. Free-list blocks are also queued to be written to the file system whenever they have been modified and released by the operating system.

2.5.3 Checking for File System Integrity

AIX includes the **fsck** command which checks for (and usually repairs) corrupted file systems. This command uses the redundant structural information in the AIX file system to perform consistency checks. If an inconsistency is detected, the **fsck** command reports the condition. Then, **fsck** corrects the condition or asks the operator to fix it. The operator uses the **fsdb** command to fix a problem that the **fsck** command is not able to resolve.

A ***quiescent*** (inactive or unmounted) file system may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multipass nature of the **fsck** program. Also note that the **fsck** command executed on a secondary file system copy can not check all aspects of the consistency on that copy because some of the directories may not be present on the secondary copy. Running the **fsck** command on a secondary file system copy, as with all file systems, is still required after a system crash.

When AIX is loaded, a file system check using the **fsck** command is performed automatically. This measure ensures a reliable environment for file storage on disk.

The root and **<LOCAL>** file systems are checked when the system comes up in single-user mode. All other file systems are checked when the system changes from single-user to multi-user mode, depending on **/etc/filesystems**.

Automatic running of the **fsck** command is often performed with the **-p** and **-f** arguments. The **-p** argument instructs **fsck** to repair those inconsistencies which have an obvious method of repair (for example, to reconstruct the free block list) without asking for user confirmation. The **-f** argument instructs **fsck** to observe the file system clean bit, and if set, to skip the check. The clean bit is set when a file system is successfully unmounted. The clean bit is not set if the system terminated abnormally, as in a system panic or power failure.

For more information on the **fsck** and **fsdb** commands, refer to the *AIX Operating System Commands Reference* and the **fsck** message description in the *AIX Operating System Messages Reference*.

2.5.4 Possible Causes for Corruption

A file system can become corrupted for several reasons. The most common reasons are:

Improper system shutdown or startu

File systems may become corrupted when proper shutdown procedures are not observed. Examples would be forgetting to shutdown the system before logging off the virtual machine, physically write-protecting a mounted file system, or taking a mounted file system off-line.

File systems may also be corrupted if proper startup procedures are not observed. Examples would be not checking a file system for inconsistencies and not repairing inconsistencies.

Hardware failur

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk pack or as blatant as a nonfunctional disk-controller.

2.5.5 Correcting Inconsistencies

The following section explains how to discover inconsistencies and how to correct inconsistencies for the:

- Superbloc
- Inode
- Indirect block
- Data blocks containing directory entrie
- Free-list blocks

The corrective actions can be performed interactively by using the **fsck** command.

Subtopics
2.5.5.1 Superblock
2.5.5.2 Inodes
2.5.5.3 Indirect Blocks
2.5.5.4 Data Blocks
2.5.5.5 Free-List Blocks

2.5.5.1 Superblock

One of the data structures most commonly inconsistent is the superblock. The superblock is prone to inconsistency because any changes to the file system's structure modify the superblock, but the disk version is updated with this new information only periodically.

The superblock and its associated parts are most often corrupted when the computer is halted and the last command involving output to the file system was not a **sync** system call. You must disable cluster traffic for the cluster site before you halt the system. For information on the **clusterstop** command, refer to the *AIX Operating System Commands Reference*.

The superblock can be checked for inconsistencies involving file system size, inode list size, free-block list, free-block count, and the free-inode count. Replicated file systems can have inconsistencies involving commit counts and superblock lists.

File System Size and Inode List Size: The file system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The file system size and inode list size are critical pieces of information to the **fsck** program. While there is no way to actually check these sizes, **fsck** can check that the sizes are within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

Free-Block List: The free-block list starts in the superblock and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count that is out of range, for block numbers that are out of range, and for blocks that are already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The first free-block list is in the superblock. The **fsck** command checks the list count for a value of less than 0 or greater than 700. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free-list block pointer is nonzero, the next free-list block is read in and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free-block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system. If anything is wrong with the free-block list, then **fsck** may rebuild the list, excluding all blocks in the list of allocated blocks.

Free-Block Count: The superblock contains a count of the total number of free blocks within the file system. The **fsck** command compares this count to the number of blocks it found free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock with the actual free-block count.

Free-Inode Count: The superblock contains a count of the total number of free inodes within the file system. The **fsck** command compares this count to the number of inodes it finds free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock with the actual free-inode count.

2.5.5.2 Inodes

An individual inode is not as likely to be corrupted as the superblock. However, because of the great number of active inodes, there is almost as likely a chance for corruption in the inode list as in the superblock.

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0) and going to the last inode in the file system. Each inode can be checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and size.

Format and Type: Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of six types:

Regular (including symbolic links
Directory (including hidden directories
Special bloc
Special characte
FIFO (or named pipe
Socket

If an inode is not one of these types, then the inode has an illegal type. Inodes may be found in one of three states: unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted inode. An inode can get in this state if bad data is written into the inode list through, for example, a hardware failure. The only possible corrective action is for the **fsck** command to clear the inode. For more information on inodes, refer to the *AIX Operating System Technical Reference*.

Link Count: Contained in each inode is a count of the total number of directory entries linked to the inode. The **fsck** command verifies the link count of each inode by traversing down the total directory structure, starting from the mount point, and calculating an actual link count for each inode.

If the stored link count is nonzero and the actual link count is zero, it means that no directory entry appears for the inode. If the stored and actual link counts are nonzero and unequal, a directory entry may have been added or removed without the inode being updated.

If the stored link count is nonzero and the actual link count is zero, the **fsck** command links the disconnected file to the **lost+found** directory. If the stored and actual link counts are nonzero and unequal, the **fsck** command may replace the stored link count by the actual link count.

Duplicate Blocks: Contained in each inode is a list or pointers to lists (indirect blocks) of all the blocks claimed by the inode. The **fsck** command compares each block number claimed by an inode to a list of already allocated blocks. If a block number is already claimed by another inode, the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number. If there are any duplicate blocks, **fsck** makes a partial second pass of the inode list to find the inode of the duplicated block. This is necessary because without examining the files associated with these inodes for correct content, there is not enough information available to decide which inode is corrupted and should be cleared. In most cases, the inode with the earliest modify time is incorrect and should be cleared. This condition can occur by using a file system with blocks claimed by both the

free-block list and by other parts of the file system. This can occur if a crash occurs and the inode for the file has been written to disk, but the superblock has not been written (with the indication that some blocks left the free list for the file). Also, note that duplicates between the free list and a file are not uncommon with the commit operation. (Note that using the **fsck** command with the **-p** option automatically fixes this condition.)

If there is a large number of duplicate blocks in an inode, it may be due to an indirect block has not being written to the file system. The **fsck** command will prompt the operator to clear both inodes.

Bad Blocks: Contained in each inode is a list or pointer to lists of all the blocks claimed by the inode. The **fsck** command checks each block number claimed by an inode for a value lower than the first data block or greater than the last block in the file system. If the block number is outside of this range, the block number is incorrect.

If there is a large number of incorrect blocks in an inode, it may be due to an indirect block not being written to the file system. The **fsck** command prompts the operator to clear both inodes.

Size Checks: Each inode contains a 32-bit (4-byte) size field. This size indicates the number of characters in the file associated with the inode and can be checked for consistency with the allocated blocks. Blocks cannot be allocated beyond the length indicated by the file size. Also, data in the last block which exceeds the file size must be all zero bytes.

The **fsck** command compares the number of blocks pointed to by the inode with the **di_blocks** field in the inode. If the actual number of blocks does not match the **di_blocks** field, **fsck** corrects the **di_blocks** field.

2.5.5.3 Indirect Blocks

Indirect blocks are owned by an inode. Therefore, inconsistencies in indirect blocks directly affect the inode that owns it. Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system. For a discussion of detection and correction of the inconsistencies associated with indirect blocks, refer to "Duplicate Blocks" in topic 2.5.5.2 and "Bad Blocks" in topic 2.5.5.2.

2.5.5.4 Data Blocks

The two types of data blocks are plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. The **fsck** command does not attempt to check the validity of the contents of a plain data block.

Each directory data block can be checked for inconsistencies involving the following:

Directory entry inode numbers pointing to unallocated inode
Directory entry inode numbers greater than the number of inodes in th file system
Incorrect directory entry inode numbers for . (dot) and .. (dot dot
Directories which are disconnected from the file system

In addition, the validity of the contents of a directory's data block is checked.

If a directory entry inode number points to an unallocated inode, then **fsck** may remove that directory entry. This condition may occur because the data blocks containing the directory entries were modified and written out while the inode was not yet written out.

If a directory entry inode number is pointing beyond the end of the inode list, the **fsck** command may remove that directory entry, but this occurs in non-replicated or primary file systems only.

Directory information is checked. The directory inode number entry for . (dot) should be the first entry in the directory data block. Its value should be equal to the inode number for the directory itself.

The directory inode number entry for .. (dot dot) should be the second entry in the directory data block. Its value should be equal to the inode number for the parent directory (or the inode number of the directory itself if the directory is the root directory).

The **fsck** command checks the general connectivity of the file system. If directories are found not to be linked into the file system, the **fsck** command links the directory back into the file system in the **lost+found** directory.

Note: This does not occur for secondary copies of replicated file systems.

2.5.5.5 Free-List Blocks

Free-list blocks are owned by the superblock. Therefore, inconsistencies in free-list blocks directly affect the superblock.

Inconsistencies that can be checked are a freelist count that is outside of range, block numbers that are outside of range, and blocks that are already associated with the file system.

2.6 System Management Files

Several files are available to help manage the system. They are:

/etc/motd (in the local file system)
/etc/inittab (in the local file system)
/etc/profile
/etc/filesystems (in the local file system)
/etc/fstore

The files that are in the local file system are tailored to the needs of a specific cluster site, not all cluster sites.

Subtopics
2.6.1 The /etc/motd File
2.6.2 The /etc/inittab File
2.6.3 The /etc/profile File
2.6.4 The /etc/filesystems File
2.6.5 The /etc/fstore File

2.6.1 The /etc/motd File

This file contains the message-of-the-day and is different on each specific cluster site. The **/etc/motd** file gets printed on the terminal when the user logs in.

The **/etc/motd** file is a symbolic link to **<LOCAL>/motd**. It is usually created by executing the **makemotd** command with the **-L** option at cluster site startup. Its contents are created by concatenating the information in **<LOCAL>/ident** with **/etc/MOTD**, and **<LOCAL>/MOTD**.

For information on the **makemotd** command, refer to the *AIX Operating System Commands Reference*.

2.6.2 The /etc/inittab File

This file indicates to the **/etc/init** command which processes should be created or terminated in each **init** state. By convention, state 3 is single-user and state 2 is multi-user. For example, the following stanza creates a sample single-user environment:

```
  sin1:
      id =  sin1
      level =  03
      action = wait
      command = "/etc/init.dir/shx Multi2singl"
```

2.6.3 The /etc/profile File

The Bourne shell (**sh**) can be invoked at login. When the **sh** command is executed at login time, it reads and executes the commands in the **/etc/profile** file before it executes commands in the user's **.profile** file. This allows the system administrator to set up a standard environment for all users (for example, executing the **umask** command, setting shell variables) and take care of other housekeeping details by executing the **news** command with the **-n** option. Note that in **/etc/profile** the shell variable $0 indicates the invocation: normal shell (-sh), restricted shell (-rsh), or **su** command (-su). There is no corresponding system-wide initialization script for C-shell users.

For more information on the **sh** command, refer to the *AIX Operating System Commands Reference*.

2.6.4 The /etc/filesystems File

This file contains a list of default devices to be checked for consistency by the **fsck** command and mounted when the system transitions to multi-user mode. The devices normally correspond to those mounted when the system is in multi-user mode. This file, normally maintained by the **minidisks** command, contains extensive information on file systems and is sorted by the gfs number.

To check all standard mounted file systems, use the **fsck** command with no file name arguments.

For more detailed information on **/etc/filesystems**, refer to the *AIX Operating System Technical Reference*.

2.6.5 The /etc/fstore File

This file contains important **fstore** information on system-replicated file systems and is composed of entries that are position-dependent. The entries have the following format:

```
pattern:0:CPU-type
```

For example, the entry **0007:0:i370** indicates the following information:

0007 Octal integer representing the **fstore** bit pattern

0 Reserved field

i370 CPU-type.

2.7 Maintaining File Systems

The system administrator has a number of responsibilities. Among these are user permissions, system security, sufficient free space, file system expansion, backbone or secondary file systems, and non-replicated file system conversion.

Subtopics

2.7.1 Permissions

The administrator (the superuser or root) has all permissions that apply to file systems and user files. The administrator can:

- Set and change the access modes of files (the **chmod** command)
- Manage user and group ownership of files (the **chown** and **chgrp** commands)
- Mount and unmount file systems
  Mount and unmount can be performed in read-only mode and in read-write mode (the **mount** and **unmount** commands)
- Remove unused files such as login files and core files with the **rm** command.

For information on these tasks and commands, refer to the *AIX Operating System Commands Reference*.

2.7.2 System Security

The system administrator is responsible for system security, which includes defining:

- Backup procedure
- File check procedures (the **fsck** command).

Subtopics
2.7.2.1 Physical Security
2.7.2.2 Access Security

2.7.2.1 Physical Security

Physical security can be achieved by:

- Keeping data on spare disk
- Using backup procedures

2.7.2.2 Access Security

Access security can be achieved by:

- Using permission
- Read only mountin
- Using non-trivial password
- Restricting permissions for groups and user
- Forcing users to reset passwords

2.7.3 Maintaining Free Space

The system administrator should ensure that the file systems have sufficient free space. On each file system, there should be at least 15% free space. A printout of the current free-space values can be obtained by using the **df** command. To check the local file systems, use the **df** command with the **-L** option.

To provide free space, do the following:

Find core and temporary files and remove them

For example, to find an unwanted core file in a user file system **/u2**, enter

```
find /u2 -name core -print
```

Note: Using the **find** command on the cluster site where the file system is mounted improves performance.

To find temporary files, look in the directories **/tmp** and **/usr/tmp**.

Find log files and truncate them

The following are examples of log files:

/usr/adm/messages	The log of administration messages.
/usr/adm/sulog	The **su** logs.

Find accounting file

For more information about accounting files, refer to *Managing the AIX Operating System*.

Clean up old service files with the **cleanup** command. For more information, refer to the *AIX Operating System Commands Reference*.

2.7.4 Expanding a File System

A file system can only be expanded in the following way:

1. Save the file system on tape or on a spare disk pack.

2. Make a new file system.

3. Restore the saved file system into the newly created file system.

File systems can be saved and restored by using the **backup**, **restore**, **find**, **cpio**, or **tar** commands.

2.7.5 Adding a Backbone or Secondary Copy to a Replicated File system

To create a backbone copy or a secondary copy from a primary copy of a replicated file system, do the following:

1. Identify a suitable site on which to store the copy.

2. Use the **rdf** command to display which sites already have a copy of the file system. Enter:

   ```
   rdf /filesystem
   ```

   Sites which already have a copy of the file system are listed. Any sites in the cluster which are not listed are candidates for the additional copy, as long as sufficient free disk space is available on the selected site. The **ptn** command can be used to list all sites currently available in the cluster.

3. When you have decided where the backbone or secondary copy is to be stored, a file system must be created on the selected site. For this operation, use the **minidisks** command. Refer to *Installing and Customizing the AIX/370 Operating System*.

4. If a secondary file system is being created, the storage attributes may need to be set for all files on the new file system. If the new file system is system replicated, use the **chfstore** command; if it is user replicated, use the **store** command.

For more information about the **rdf, ptn, chfstore,** and **store** commands, refer to the *AIX Operating System Commands Reference*.

2.7.6 Converting a Non-replicated File System to a Replicated One

To convert a non-replicated file system to be the primary copy of a replicated file system, do the following:

1. Save the file system on tape or on a spare disk by using the **backup**, **restore**, **find**, **cpio**, or **tar,** commands.

2. Use the **mkfs** command with file system name along with the **-r** option. The **-r** option causes the newly created file system to be the replicated type.

3. Restore the saved file system into the newly created file system.

For more information, refer to "File System Backup Utilities" in topic 3.5.

3.0 Chapter 3. File System Backup

Subtopics

3.1 Contents

3.2 About This Chapter

After the system is up and running, the next consideration is protecting system data by backing up the file systems. This chapter describes file system backups in detail. It shows you how to back up file systems and presents guidelines regarding when backups should be done.

3.3 General Backup Policies and Principles

The mechanism or combination of mechanisms adopted for backing up file systems is largely unimportant; all the mechanisms described in this chapter can provide reliable protection against catastrophic or accidental data loss. It is also possible for a user to write backup and restore utilities. What is vitally important is that the combination of mechanisms and procedures decided upon be viable in the particular environment and afford adequate protection. Different facilities have different modes of operation, and a backup system that works well in one environment may not be appropriate in another. There are, however, some general considerations that are applicable to all sites and situations:

The standard backup medium for System/370 is nine-track magnetic tape the device name is **/dev/rmt0rh**. The standard backup device can be reset by specifying a different value for **backupdev** in the **/etc/filesystems** file.

Make sure there is coverage against catastrophic failures. Can th system continue running (with diminished capacity) after having suffered the loss of any disk? Can the system be brought back up after the total loss of all the disks? What is to be done if a fire near the computer destroys all the disks and on-site backups? Although unlikely, these situations can occur. It is advisable that you anticipate these situations to ensure the system can actually be recovered.

Use the backups (even if they are not needed). Unfortunately, tap drives may require adjustment over time. It can be disconcerting to have a large library of backups and find that they cannot be read back in when needed. It is advisable to restore old backups periodically to make sure they are still readable. If multiple drives are available, attempt to read backups back onto a drive other than the one on which they were written.

Make duplicate copies of important backups. There is always possibility that a backup tape will be lost, overwritten, destroyed or just deteriorate with age. To protect system data, several complete backups should be done and the copies stored separately in different locations.

Keep old backups. Most sites reuse their backup tapes in some cycli fashion; however, do not reuse all the tapes. Keep old backups for long periods of time; it may be some time before users realize that a file has been deleted. One popular approach is to have three cycles of backup tapes. Daily tapes are recycled once a week. One set of daily tapes, for example, Friday's tapes, would be an exception. Friday tapes are recycled once a month so that the last four Friday backups are always available. The last Friday backup tapes from each month are kept for a quarter. The last monthly tapes from each quarter are kept (off-site) forever. Creating backbone copies of replicated files is also a feasible way to have backups, but are not a replacement for tape backups.

Check file systems before they are backed up. A backup is used t recover from problems, so if the file system was damaged at the time the backup was performed, the integrity of the data is affected. This problem can be avoided if the **fsck** command is run on each file system immediately before the backup is started.

Automate the backup procedures as much as possible. Using a standar command file to so daily backups simplifies the operator's task and helps to ensure that backups are done properly. Successful completion of system backup should be automatically logged. Ticklers or log files generated by the **cron** command can check to determine if backups have been done and generate appropriate warnings if they are not.

Fine tune backup procedures for the installation by determining wha works best for your particular configuration. Whether a particular backup procedure is faster than another depends on your configuration and needs to be determined over time by trying different procedures. For example, a VM DDR image copy may be faster than a minidisk backup but slower for a full backup of the entire file system.

In deciding when and how often to do backups, consider the amount of activity on the system involved. A file system that does not change frequently, for example, one that contains system documentation, does not need to be backed up daily. On the other hand, file systems on which users actively work should be backed up regularly.

When file systems are backed up, the system must be fairly quiescent. If a file system is backed up while it is in use, it is possible for files to be changed as they are being backed up. This can result in a backup of down-level files.

Another factor to consider when deciding to do a backup is the amount of time the file system should go unprotected. If backups are taken in the morning, a day's work is unprotected until the following morning. A system crash in the middle of the night can destroy the previous day's work. Evening backups, on the other hand, promptly protect each day's work.

Subtopics
3.3.1 When File System Backup Copies Should Be Made

3.3.1 When File System Backup Copies Should Be Made

It is best to establish your own policy for backing up files and file systems. However, here is a suggested file system backup procedure that can be used as a guideline for your installation:

Note: In the following plan, only primary copies need to be backed up (not backbone or secondary copies).

Frequency	**Activity**
Daily	Copy all user file systems to backup disk or tape volumes. Keep these volumes three to five days before reusing them.
Weekly	Copy each file system to tape. Keep weekly tapes for eight weeks.
Bi-monthly	Copy all file systems to tape. Keep bi-monthly tapes indefinitely. They should be recopied once a year.
Monthly	Back up the root and **<LOCAL>** file systems.

The most recent weekly tapes should be kept off the premises. Other tapes should be kept in a fireproof safe.

It is advisable to make daily backups of file systems that change every day. Either full or incremental backups can be made. Full backups can be made daily, but because they are so I/O intensive and can slow down system performance, it may be more effective to do a full backup of one file system and incremental backups of all other file systems every day. This combination plan requires less time to perform than full backups on all file systems every day. However, this method does require that both the full backup and incremental backup be copied in order to completely restore a lost or damaged file system. For more information on restoring files and file systems, refer to "Restoring File Systems from Backups" in topic 3.4.2 and "Restoring Files from Backups" in topic 3.4.3. In general, backups should be performed when the fewest number of users are on the system.

3.4 Types of File System Backups

The two basic types of backups are complete backups and incremental backups. Complete backups include all the data in a file system and can be further subdivided into volume image backups and file backups. Incremental backups include only those files that have been changed recently. Each type of backup has advantages and disadvantages.

Subtopics

3.4.1 Creating Backups

Complete backups are done using the **backup** command with the **-m** flag or the **0** level flag. Incremental backups are done with the **backup** by inode or **backup** by name commands. For either type of backup, disk-to-disk (keeping this copy on-line), and disk-to-tape (keeping this copy off-line) copies are suggested. The second copy is important in case the first copy is damaged accidentally (especially if it is mounted and thus, susceptible to user errors).

Files stored with the **backup** command can be recovered by executing the **restore** command. Basically, files can be restored from incremental backups on a file-by-file basis. An image (**backup** with the **-m** option) must be restored completely.

Subtopics
3.4.1.1 Example

3.4.1.1 Example

Assume there is a file system on **/dev/fhd00001** which has full backups done on Mondays and incremental backups done on Tuesday, Wednesday, Thursday, and Friday. On Monday, the following command is run:

```
backup -0 -u /dev/fhd00001
```

Then on Tuesday, the following command is run:

```
backup -1 -u /dev/fhd00001
```

The above command gets all files that have been modified within one day (one level) for Tuesday. On Wednesday, a -2 (for a level 2 backup) would be specified and on Thursday, a -3 (for a level 3 backup) would be specified. On Friday, the system administrator would execute the following:

```
backup -4 -u /dev/fhd00001
```

This command backs up all the files modified in the last four days (since Monday). For more information on the **backup** command, refer to the *AIX Operating System Commands Reference*.

3.4.2 Restoring File Systems from Backups

To restore primary copies or non-replicated file systems, do the following:

1. Locate the exact area where the restored file system will be placed. Make sure there is enough space for it.

2. Copy the full backup to the device using the **restore** command.

3. Copy the files in the most recent result of an incremental backup with **restore**.

After restoring the primary copy of the replicated file system, complete the following steps to restore backbone and secondary copies:

1. Invoke **mkfs** by executing the **minidisks** command. This creates a new backbone or secondary file system using the same options as used on the original.

2. Check the new copy of the file system using the **fsck** command.

3. Mount the file system on a site in the same cluster as the primary copy site. The TCF support in AIX/370 re-propagates the file system to the backbone or the secondary copies.

Failure to do this will result in inconsistent replication and unpredictable system behavior.

Note: Restoring files on an MBCS system requires some special cautions. For information, read the sections entitled "Accessing Files under Different Locales" and "File Restoration between Singlebyte and Multibyte Character Clusters" at the end of this chapter.

Subtopics
3.4.2.1 Example

3.4.2.1 Example

If you want to take the information on the backup tapes and write the entire file system information onto **/dev/rfhd00001**, restore the latest full tape onto the file system, and then restore the incremental backup , if any, on top of the file system. If this restoration is done on Friday, the following command should be executed first:

```
restore -rv /dev/rfhd00001
```

This command gets the original file system, as of Monday. For all files on the most recent incremental backup, execute the following command:

```
restore -rv /dev/rfhd00001
```

3.4.3 Restoring Files from Backups

To restore individual files or directories and files:

1. Determine where the file is and whether it is a full or incremental backup. Check the table of contents before searching the backups. Determine which backup has which files and the inode numbers of the files to be restored.

2. If you are performing an incremental backup, use the **restore** command on a file-by-file basis. If you are performing an image backup, use **restore** on the minidisk and copy the right files.

If a user deletes or damages an entire tree structure (login directory and all files and directories under this), special handling is required. All the tables of contents must be examined from the date of deletion and everything back to the last full backup. All changed files need to be found and read in separately.

Note: Restoring files on an MBCS system requires some special cautions. For information, read the sections entitled "Accessing Files under Different Locales" and "File Restoration between Singlebyte and Multibyte Character Clusters" at the end of this chapter.

3.4.4 How File System Replication Affects Backup Procedures

In general, file system replication makes it easier to backup file systems. Replication is especially useful in guarding against catastrophic failures (for example, when a primary copy disk crashes) because backbone copies can be substituted for full backups.

3.4.5 Creating a Primary Copy From an Existing Backbone Copy

Note: The following material assumes a worst-case scenario in which the damaged file system is the primary copy of the root filesystem, a file system that may not be unmounted. If the file system can be unmounted, this procedure is easier. Unmount all remaining copies of the file system and follow the instructions here without regard to rebooting or making the system quiescent.

It is presumed in the following procedure that the primary copy no longer exists (for example, a disk crash). If the damaged file system is the replicated root, perform the following steps:

1. Choose one of the backbone copies of the replicated file system to be turned into the primary copy. Reboot the site, stopping in maintenance (single-user) mode without enabling TCF traffic. This procedure provides a quiescent system for running the **sec2prim** utility.

2. Run the following command to verify that the conversion of the backbone copy is going to be successful.

   ```
   sec2prim /dev/root
   ```

   This command lists any files that would be lost as a part of the conversion if, for example, this backbone copy was on an AIX/370 site and the new primary site was a PS/2. Convert the backbone copy of the replicated file system into a primary copy by running the **sec2prim** command with the option set to update the file system:

   ```
   sec2prim -u /dev/root
   ```

   Repeat the **sync** command and reboot this site again, allowing the file system checks of the modified root file system to be performed, this time allowing the site to go to the multi-user mode with TCF traffic enabled. You now have a new primary copy of the file system.

3. Return to the site with the damaged primary copy of the replicated root. Bring up this site as an installation site. When you install this site, you will be creating it with a backbone copy of the root file system.

3.4.6 Volume Image Backups

A volume image backup is a bit-for-bit copy of the volume containing a file system. Image backups are generally simple to take and restore. They are the simplest and fastest means of protection against a major disk failure.

Because of the difficulty of recovering from the loss of a system residence disk, it is recommended that installations keep an emergency system disk (or at the very least, a copy of the distribution tapes). Generally, the only way to lose the system disk is through a major hardware failure, and occasionally major hardware failures do occur. For this reason, it is important to have a backup copy that can be restored in a standalone fashion. If at all possible, there should be two drives for the system residence volume. If there are two drives and one of them crashes, the system can be run with reduced capacity; this needs little effort: simply change the VM/SP directory, or change the virtual address of the disk by using CP DEFINE.

Image backups are usually quite simple, but they do have several disadvantages. It is difficult to restore an individual file or directory from an image backup. The entire backup must be restored to a scratch file system disk and the desired file(s) extracted from the restored file system.

Image backup can protect against catastrophic failure, but can be awkward as a means of protection against inadvertent deletion or limited file system damage. Because an image backup is a bit-for-bit copy of a file system disk, it can only be restored onto a file system disk that is at least as large as the one from which it was taken. Since an image backup includes all the contents of a disk (even the unallocated blocks and inodes), it wastes space.

3.5 File System Backup Utilities

With AIX/370, the following backup utilities can be used:

DDR (VM Utility
dd
backup
restore
tar
cpio

For information on the **dd**, **tar**, **cpio**, **backup**, and **restore** commands, refer to the *AIX Operating System Commands Reference* and *Installing and Customizing the AIX/370 Operating System*. For information on the DDR command, refer to CMS command descriptions in the *VM/SP System Product CMS User's Guide* or the *VM/XA System Product CMS User's Guide*.

Subtopics

3.5.1 The DDR Utility

The DDR utility can be employed to backup and restore entire disks. This is a fast and efficient means of performing large-scale backups. This can be used to move entire disks from one machine to another (cluster site to cluster site). The disadvantage of this approach is that backups must be restored to the same type of disk from which they were taken. The AIX **dd** command can be used for backup, but backups must also be restored to the same type of disk from which they were taken.

3.5.2 The Backup Utility

The AIX/370 **backup** by minidisk utility can be used to copy file systems from disk-to-disk, disk-to-tape, or tape-to-disk, blocking the transfers to maximize efficiency. If two (or more) disk drives are available, backup by minidisk can be used in a fast, simple, and efficient way to back up all the file systems. By copying each file system (logical disk) on one drive to the corresponding logical disk on the other drive, all the file systems can be backed up in a short period of time. Once the copy has been made, the copied volume can be stored in a safe place.

It is not necessary that a backed-up file system be restored to the same logical disk from which it came. The backed-up file system can be restored to any logical disk that is large enough to accommodate it. There must be the same amount of space available on the original file system disk/tape and the new file system disk/tape, in order for the **restore** command to work. If a drive is lost, the file systems on that drive can be restored onto logical disks of a different drive.

Note that backup by minidisk can be used to move file systems from one cluster site to another (provided the file systems are the same type.)

The **backup** command can be used to save file systems on disk or labeled tape. When saving to disk, ensure that the receiving file system is large enough to hold the copy. For example, to save from disk to disk (rchd000022 to rchd000023) the command is:

```
backup -mf /dev/rchd000023 /dev/rchd000022
```

The following example shows how to save from disk to tape:

```
backup -mf /dev/rchd000022 /dev/rmt0rh
```

The **backup** command prompts to mount additional tape reels if the file system does not fit on one reel. All tape reels must have identifying labels affixed.

Note: Backing up files on an MBCS system requires some special cautions. For information, read the sections entitled "Accessing Files under Different Locales" and "File Restoration between Singlebyte and Multibyte Character Clusters" at the end of this chapter.

3.6 Individual File Backups

An alternative to the volume image backups is backing up each individual file on the file system. Individual file backups permit files to be restored individually, and because only the files are backed up, no space is wasted for unused blocks. Also, a group of files can be restored into any file system that is large enough to accommodate that group of files. It is not necessary that the receiving file system be as large as the original file system. However, the replication and control information (if present) is not preserved for the restored files unless the files were restored onto a replicated file system.

AIX/370 has several utilities for doing and restoring individual file backups of file systems. These are the **cpio**, **tar**, **backup** and **restore** commands. Some utilities are usable only by the administrator while others are available to individual users. For more information on these commands, refer to the *AIX Operating System Commands Reference*.

Note: The **backup** and **restore** commands keep replication information on file systems, but the **cpio** and **tar** commands do not.

3.7 Incremental Backups

Taking complete backups of file systems every day can waste considerable time and space. Since only a small portion of the files on any given file system are likely to change during the course of a single day, it is unnecessary that all the data be backed up every day. The purpose of an incremental backup is to save only those files that have changed since the last backup (complete or incremental).

The availability of incremental backing-up mechanisms makes it possible to have an efficient backup procedure. This procedure requires the dedication of an amount of disk space large enough to hold copies of all the files that change during any one day, and if that space is available, it is a very convenient mechanism.

Ideally, backups should be performed late at night, when the system is not heavily loaded and few, if any, files are being actively used. At installations that do not have an operations staff working night shifts, it is not feasible to do a full set of backups at night, since there is no one available to mount the required tapes. However, it is not necessary to do backups to tape. An incremental backup for all the disks in a large system normally amounts to only a small fraction of blocks. If there is enough disk space available, the incremental backup can be done to a file on a reserved file system. (On a large AIX/370 system, a large spare file system should be kept available in any case for doing restores and for other bulk move operations, and that same file system can be used for the nightly incremental backups.) In the morning, the operator can easily copy the files containing the incremental backups to tape; this operation can be carried out while other users are on the system.

In addition to the convenience for users and operations staff, this scheme has another advantage in that recent incremental backups remain on-line. Incremental backups, however, should be on separate physical volumes from essential data. When a user accidentally deletes a file, it will not be necessary to restore a backup tape to retrieve it. The chances are good that the file can be recovered from an incremental backup that is already on disk. The number of incremental backups that can be maintained on-line depends on how much disk space is available.

There may be a need to back up an individual user's files. The administrator, or the user himself, might wish to back up infrequently-used files to free space on a file system. AIX/370 includes utilities (for example: **tar**, **cpio**, **backup/restore**) that can do this. These utilities can also be used to transfer files via tape between systems or sites.

Subtopics

3.7.1 tar

The **tar** utility can be used to backup individual files or entire directory structures to tape. The files can be read back in, individually or as directories, by specifying their names. The **tar** command is simple to use and because it does not require special privileges, any user can use it directly. This command does not put replication information (**fstore** values) on the tape.

For more information on the **tar** utility, refer to the *AIX Operating System Commands Reference*.

3.7.2 cpio

Another utility for backing up and restoring files is **cpio**. The **cpio** command allows files to be backed up and retrieved by name and is well-suited to the needs of the individual user as well as the administrator. However, it does have the same drawback as **tar** in that it also does not put replication information on the tape.

The **cpio** command takes a list of path names from standard input and copies the files to standard output. For example, the **find** command can be used to drive file backup; the following command will backup all the files under **/u2/steve** that have not been modified within the last 24 hours:

```
find /u2/steve -mtime -1 -print | cpio -oB > /dev/rmt0nh
```

Hidden directories can also be fully preserved, using the **-hidden** flag for the **find** command. For more information on the **find** command, refer to the *AIX Operating System Commands Reference*.

This same procedure can be used for system-wide or filesystem-wide incremental backups.

3.7.3 Backing Up Files by Inode

The **backup** command with **level** and **filesystem** specified will back up files or file systems by level number or inode. For more information, refer to the *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

3.7.4 Backing Up Files by Name

The **backup** command with the **i** flag will back up files or file systems by name. For more information, refer to the *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

3.7.5 restore

The **restore** command copies back files or file systems that were created by the **backup** command. For more information, refer to the *AIX Operating System Commands Reference* and *Managing the AIX Operating System*.

3.8 Backing Up a Remote File System

The **tar**, **backup/restore**, and **cpio** commands can all be used to backup file systems from one cluster site to another. An important distinction about devices needs to be made in order to understand how the commands are used.

Disks and tapes are examples of devices. A local device to a cluster site is a device that is connected directly to that cluster site. A remote device to a cluster site is a device that is not connected directly to that cluster site, but is connected to another site in the same cluster. When devices are local to a cluster site, then both raw (character) and block devices may be used for file system backing up. However, when devices are remote to a cluster site, then all of these devices must be accessed through the block device interface.

For example, if the local cluster site is **tigger** and the file system *user1* is to be copied to a disk connected to site **eyore**, do either of the following:

```
on tigger dd if=/tigger/dev/rfhd000021 of=/eyore/dev/fhd000022 bs=60k
on eyore dd if=/tigger/dev/fhd000021 of=/eyore/dev/rfhd000022 bs=60k
```

In this example, sites **tigger** and **eyore** are the volume IDs (volid). A copy could be done from the raw tape device to **eyore's** disk, but the reverse can not be done.

There is a special non-local device problem that can arise when using tape. Block devices are accessed in 4096 byte units. While this is a reasonable block size for normal disk access, it is too small to permit efficient tape usage. In order to access devices in units larger or smaller than 4096 bytes, the character device interface must be used. Raw devices cannot be accessed remotely, however.

For example, in order to backup a file system from **tigger's** device-attached disk onto **eyore's** device-attached tape with maximum efficiency, a raw character device read must be done on **tigger's** disk and a raw character write on **eyore's** tape. This is accomplished by piping the first process over to the second as follows:

```
on tigger dd if=/tigger/dev/rfhd000022 ibs=60k |
on eyore dd of=/eyore/dev/rmt0rh obs=60k
```

Note: The above example needs to be on one line, not two.

The input device is blocked 60k and the output device is blocked 60k. In this way, the block size can be defined without restricting the block-device size of 4096.

3.9 Quiescing the System

During normal system operation, it may be important for activity on the system to be quiesced so that particular functions can be performed in a reasonable manner. This section addresses these occasions and outlines the requirements and the flexibilities that exist to the administrator.

In a typical UNIX operating system, file system activity is not possible unless there is some local process entity making such changes. With the Transparent Computing Facility (TCF), Network File System (NFS), or AIX Access for DOS Users (AADU), this is not true in that remote processes can be executing and causing the changes to occur. Therefore, one significant component of quiescing the system is to make sure that all remote activity is terminated. For information on NFS, refer to *Managing the AIX Operating System*.

There are at least three situations when the system should be quiesced:

Normal system shutdown or reboo

File system consistency check

File system backups (or VM backups)

AIX PS/2 and AIX/370 are designed to minimize loss of data when a system failure occurs. A system administrator really needs to do little in order to make sure that no data is impacted. However, the subsequent system restart will be much slower if the system is not terminated properly. The system needs to clear all pending disk activity and flush all internally stored file system information to disk. If done properly, consistency checks need not be performed on those file systems. The **umount** system call and command perform the function of flushing pending I/O activities and making a file system unavailable. A **umount** operation of all user file systems should be part of normal system shutdown. This can either be done manually or as part of shutdown. It is also important that network activities and local processing activities be terminated during normal system shutdown.

By not doing a clean shutdown, it will be essential for all file systems to have consistency checks prior to complete system restart. Furthermore, recent file data which are still only present in the system buffer cache will be lost.

In order to perform a consistency check on a file system, there should be no ongoing activities in that file system. When a file system is active, it is not possible to get a single, consistent picture of the data structures over some period of time. The easiest way to guarantee that a file system is not changing during the execution of file system consistency checks is to make sure that there is only one **fsck** command running on a given file system and to make sure that file system is not mounted. If the file system must remain mounted (like the root and **<LOCAL>**) during the consistency checking operations, then the system activity is to be quiesced with respect to these file systems. This means there are no changes being made by the system during this time. If the **fsck** command changes any disk data structures of a mounted file system during the consistency checking, then to maintain data integrity, the system should be rebooted without a **sync** operation being performed. However, there are several types of changes that do not require this drastic action. All superblock counts and free block list correction activities can occur without a reboot.

In the case of file system backups, two situations can occur. If the backup activity is a regular backup and not a bit image copy of the entire file system, then it is possible to do the copy while the file system is active. For example, executing the **tar** utility on an active file system is possible and the risk involved is that some recently created files will not get on the tape. When moving file systems via a **backup** utility, it is essential that there be no activity to guarantee that all the data is correctly transferred. In the case of bit image copies of file systems, the risk is that the system is not able to snapshot the entire file system in a single, consistent state. Therefore it is possible that the binary copy of the file system will not be in a state where it is internally consistent. In most cases, it will be possible for the **fsck** command to correct the problems in the backup when restored. In some very rare cases, **fsck** will not be able to fix consistency problems that might exist in the backup when restored. The decision the administrator must make is a trade-off between guaranteed consistency and availability during backups. The administrator may chose to keep the system operational but pick a time of day to do the backups when there is limited activity on the system.

3.10 Backing Up Replicated File Systems

Earlier in this chapter, general issues concerning backups were discussed in detail. Primarily, consideration must be given to the same backup and restore issues for replicated file systems as for non-replicated file systems. For example:

Decide how often to do full backups versus incremental backup

Use many of the same tool

Use the same tape recycling procedure

However, when working with replicated file systems, there are some additional issues to consider. In most cases, these issues relate to restoring files from backups and following guidelines to prevent the violation of primary copy replication. These new issues are:

Common backup issues between replicated and non-replicated fil systems

Basic review of replication principle

Key considerations for replicated file system backups

Subtopics

3.10.1 Common Backup Issues

Several issues that must be considered when developing a backup procedure are exactly the same for replicated and non-replicated file systems. These issues include:

Time of day to do backup

Utilities used to do backup

Strategy to use to recycle backup tape

Full backups versus incremental backup

In addition, the following issues regarding backups on replicated file systems are similar to those of non-replicated file systems, but with additional precautions:

The reasons for doing backups: For both replicated and non-replicate file systems, perform backups so that files can be restored in case of system failure. With replicated file systems, however, the replication of files limits the impact of any single failure, except user error. If a user removes a file, all copies of the file are removed.

The only situation where a user can remove a file and not remove all copies is if one copy of the file is not present in the cluster at the time the file is removed. If the user detects the loss of the file before the offline copy is brought online with the cluster, the user should be able to retrieve the data from the offline copy.

The procedure to use for full backups: For both replicated an non-replicated file systems, decide whether to use image backups or complete file by file backups.

If the decision is to use image backups on a replicated file system, consider the issues regarding restoring the image backup. These issues are outlined in "Volume Image Backups" in topic 3.4.6.

3.10.2 Replicated File System Backups

Replicated file systems are easier to backup in some ways than non-replicated file systems. For example, it is not necessary to take the primary copy of a file system offline if a backbone copy of the file system exists. Make sure the backbone is sufficiently up-to-date and then do the backup from the backbone. If the backbone is to be unmounted, the data will still be available to users through the primary copy. In addition, it is not normally required to backup all copies of a replicated file system. Clearly, just backing up one of the backbone copies or the primary copy is sufficient. However, before using any backup procedure on a replicated file system, test the procedure carefully to verify that the replication control information is not lost. Not all backup utilities store this information on the backup media. In addition, some backup procedures discard the information (the replication control information is not copied by the **cpio** command) before the backup tool is used.

Most problems with replicated file systems and backups involve restoring data. For example, if the primary copy is destroyed due to media failure, a backbone copy must be converted to the primary copy or the file system must be restored from a backup and all the copies repropagated. Repropagate by using the **minidisks** command, which invokes the **mkfs** command and creates a new backbone or a secondary copy.

Any process that restores data into the file system via a file-by-file create, write, and close sequence causes minimum disruption. To make this function efficiently, a scratch disk or minidisk may be used to handle the complete binary backups efficiently.

Any operation which sets the low-water and high-water marks backwards in the primary copy relative to the backbone and secondary copies requires repropagation of all file system copies.

The following table illustrates some typical restore situations and indicates whether the copies must be repropagated.

Table 3-1. Typical Restore Situations

Operation	**Requires Repropagation of Backbone and Secondary Copies**
Restore files from **tar** or **cpio** media	No
Restore files from **backup** by name media	No
Restore backbone or secondary from **backup** by minidisk	No
Restore primary from **backup** by minidisk	Yes (1)
Restore primary from **dd** or DDR	Yes (1)
Restore backbone or secondary from **dd** or DDR	No
mkfs a primary copy file system restore **tar, cpio,** or	Yes (2)

backup by name tapes	

(1) If the primary copy is as recent or more recent than the secondary or backbone copy, the backbone or secondary sites do not need to be repropagated. For example, a primary copy of a file system can be moved using a binary image operation without repropagation.

(2) This is effectively a different file system, so the copies must be restarted with respect to the new primary copy.

3.10.3 Accessing Files under Different Locales

Considerable care must be taken when restoring files on a multibyte character system. This is especially true in a cluster where some of the users work in singlebyte characters and others work in Japanese.

Files which were created with multibyte character names will not be readable by any process running under a singlebyte character locale. File systems which are backed up on a Version 1.2.1 system operating under a Japanese locale are likely to contain files with multibyte Japanese filenames. Files with multibyte character names cannot be read by any process operating under a singlebyte character locale. If this file system were then restored to a directory used only under a singlebyte character locale, the files with Japanese names would be not only unreadable, the names would be unreadable in a file listing such as **ls**.

The same would be true even if the file system were restored to the very same machine and directory from which it was backed up; if that machine is operating under a singlebyte character locale, the files with multibyte filenames cannot be accessed.

On a system which is multibyte capable, the solution is simple. The user can simply change locale to match the locale under which the files were created and they can be accessed. There is, however, a situation in which the results of file restoration could be much more destructive, as noted in the following section.

3.10.4 File Restoration between Singlebyte and Multibyte Character Clusters

As a system administrator, you may face a situation in which a facility has both English and Japanese language users. This facility, therefore, will have two separate AIX clusters. The English language users work on the first cluster; it runs AIX/370 1.2, which recognizes only Roman characters, nearly all of which are singlebyte glyphs. The Japanese language users work on the second cluster; it runs AIX/370 1.2.1, which recognizes both ASCII and Japanese characters, many of which are multibyte glyphs. Both groups of users work for the same firm, use the same sort of applications and generate the same sort of data files. The only apparent difference is the language. It is easy to assume that data can be freely exchanged between the two with the only potential problem being that sometimes the users might not understand the language in which a file was written.

This assumption is erroneous. In fact, data can move relatively freely from the Roman character system to the Japanese-capable system, but NOT in the other direction. Most of the Roman characters created under AIX Version 1.2 are readable by the "pc850" file code of AIX Version 1.2.1. The few characters that are not readable are extremely rare. For information on file codes, refer to the *Guide to Multibyte Character Set (MBCS) Support*.

Problems can occur if the Japanese cluster is running out of disk space and someone decides to move an entire file system to the English cluster. That file system can be backed up (by **backup**, **tar**, **cpio** or **dd**) and restored by the appropriate method on the ASCII-only system; however, the problem is that the file system probably contains files that are stored under Japanese filenames. Any of the backup methods discussed restores them with the same filenames. Those files are then both unreadable and inaccessible by the European language system. Users on this system cannot change their locales to gain access to the files. A general rule to remember is to restore only those file systems in a like locale.

For more information on MBCS capabilities, refer to material in this manual, the *Guide to Multibyte Character Set (MBCS) Support*, *Using the AIX Operating System*, and *Managing the AIX Operating System*.

3.10.5 Rules Summary for Mixing Systems

The following rules should be followed when mixing versions of AIX:

AIX Version 1.2.1 systems must never be mixed in the same cluster wit AIX Version 1.2 systems. All sites in a cluster must run the same version of the operating system. If an organization wants to use Japanese locales in a cluster, all sites in the cluster must run AIX/370 1.2.1 or AIX PS/2 1.2.1.

An organization that wants to run both AIX Version 1.2.1 and AI Version 1.2 should run them on different clusters and adopt careful policies and procedures for file transfer and backup/restore operations between the two clusters. File transfers between users may litter the AIX 1.2 system with files that can be read only on the AIX 1.2.1 system. If the files are user files, the results are at best inconvenient. If the files must be read by programs or scripts, the results can be very unpredictable. Backup and restore operations between the two clusters must also be done with care.

4.0 Chapter 4. Licensed Program Product (LPP) Service Process

Subtopics

4.1 Contents

4.2 About This Chapter

This chapter describes commands and processes that allow you to apply service to the AIX/370 system or cluster you administer. Be sure to read this entire chapter, and peripheral information (like the actual "Information File" that comes with the LPP) before you perform any process.

For additional information regarding LPPs, refer to the *AIX/370 Planning Guide, Installing and Customizing the AIX/370 Operating System,* and *AIX Programming Tools and Interfaces*.

4.3 Introduction: Theory of LPP Service Process

The service process exists to install corrected and updated software on your cluster. In order to give you some control over service, this process will allow you to also remove updates with which you are unsatisfied. These updates can exist on your cluster in any of four different states: applied, commit, uncommit and reject.

Note: For more information on the files and scripts involved in an update, refer to the chapter entitled "Installing and Updating an LPP" in *AIX Programming Tools and Interfaces*.

Subtopics

4.3.1 Applying Service

When you install service, you "apply" updates. This option allows you to update your cluster with corrected programs for the LPP you have installed on your system. The **updatep** program saves old versions of files and programs so that you may "reject" service with which you are not satisfied. When you are installing or servicing your system, you are, in effect, building up layers of new software, much like building an onion. This has important implications. Service is applied in this system with a Last-In-First-Out strategy. In order to remove any particular layer of installation or service, you will have to remove all other layers of service installed prior to that layer.

Note: There is no mechanism to remove an LPP after you have installed it, and you cannot remove service beyond the last LPP installed unless you take special precautions during the installation. For a description of these special precautions see "Rejecting Service Past an LPP Installation" in topic 4.3.10.

Also note that you can apply service for more than one LPP at a time during an **updatep** session. In order to maintain consistency across updates to multiple LPPs, the service process bundles together service applied to more than one LPP. When service is applied together for multiple LPPs, that service is manipulated as a single entity, and must then be committed, uncommitted, or rejected together. You may reuse your update media to apply other updates, or if you chose to do so, apply service to one LPP at a time. This uses a good deal of disk space to save the older versions of the system; therefore, you should apply updates together.

Note: You can have only one applied update on your system or cluster at a time.

During the update process, if problems are found, that level of service is automatically rejected and the system returns to its original state.

4.3.2 Committing Service

Once you are satisfied with an update you have applied, you can "commit" that update. Committing an update puts your service into a more permanent state. The previous levels of software are now moved to a save-stack frame and may be retrieved later, or archived to backup media. With all your service in this state, you may now apply other updates to the system.

4.3.3 Uncommitting Service

Should you decide that you want to restore previous levels of service that you have committed, you must "uncommit" the current service level. This option restores updates back to the applied state, where they may be rejected. If you have used the **cleanup** command to archive save-stack frames to tape, you must make sure you have restored the frames that you are about to uncommit. The uncommit option will not prompt you for them.

4.3.4 Rejecting Service

Once you have restored an update back to the applied state, you may reject that service. This removes the new update from your cluster and replaces it with the previous commands and files as they existed before you tried to install the update.

4.3.5 Updating Locals

The above commands apply service in general to an entire cluster. The new versions of commands are automatically propagated to the proper sites. For more information on propagation, refer to "fstore Values and Propagation" in topic 2.5.2. This automatic mechanism does not work for updates applied to the locals for any sites in your cluster. In order to update your local (with any of the above states of service) you need to run the **qproc** command. The **qproc** command runs scripts that apply or reject service to a single site. In general, you must run **qproc** on all sites in your cluster every time you do any of the service functions above. Running **qproc** on a system that needs no service applied does no harm. You may, at your discretion, wait to run **qproc** until you have an entire session of updates to be applied and committed. The **qproc** command then applies as much service as it can. Note that **qproc** is the mechanism that may install new kernels on your systems, and may request that you reboot the system after service is applied.

4.3.6 Managing Your Service Process

While this update process can be run from any site on your network, it is suggested that you service your cluster from its primary site. This will reduce network traffic and reduce the amount of time the update will take. While AIX supports automatic propagation of new and updated files within the cluster, it is recommended that you only update your cluster when it is idle. This is to make sure a minimal set of processes within the cluster are busy. See the explanation of the ETXTBSY error in the **/usr/include/sys/errno.h** file. More information on error conditions can be found in Appendix A of the *AIX Operating System Technical Reference*.

By updating your cluster when it is idle, you can also control the updates to locals and possible reboots. For each LPP update, there may be a file included on the update that contains important user information. You should make sure that you read these files as they may contain special instructions you may be required to perform before or during this update. They will also contain useful information about the update, its limitations, and other related material. Although most updates to a cluster are independent, you should strive to keep your system at the latest service levels. In particular, due to the close interaction of these LPPs, the updates to aix370, opsys, and TCP/IP should always be kept at the same levels of service.

The **restore** command handles the processes that are busy during update. But since these files cannot be removed while they are still active, the **restore** command renames them by appending the file called **original_filename.deletemeccccc** where **ccccc** are unique identifiers assigned by the system. These files may be removed subsequent to the update process.

4.3.7 Managing Your Disk Space

The update process is disk-space intensive. The documentation that you receive with each update will tell you how much space any particular update will require. This is the amount of space the update will require to keep the previous level of service available to you in case you wish to reject this particular update. In order to recover this disk space, the **cleanup** command is provided. This command archives updates to archive media. Be sure to record the frame number supplied to you during the commit step of the update process. For more information on the **cleanup** command, refer to the *AIX Operating System Commands Reference*.

If you wish to keep all the committed service available on your system, you should make **/usr/lpp.save** a mounted filesystem. The **/usr/lpp.save** is the directory where all committed service is stored. This filesystem should be mounted on the primary site of your cluster. Service that is only in the applied state is stored in **/usr/lpp/lpp-name** for each LPP. Although this directory could also be made a mounted filesystem, many LPPs store important files there, and so you may be affecting the performance of those LPPs by removing them from the replicated root filesystem.

4.3.8 User Configurable Files

If you have made any updates to your system that have not been controlled by the AIX service process, you should make sure that you have backed up those changes before any service session. These include any changes that you have made to system libraries, or configuration files, or software that was not installed as an LPP.

4.3.9 Prerequisites

When updating your system, certain updates may have dependencies on other LPPs being updated at the same or known previous levels of service, although for the most part, this will be transparent to the user of the **updatep** command. You may become aware of this dependency if you do not have the prerequisite LPP installed. If this is the case, you are allowed to install ignoring prerequisites. For more information, see the "User Information" files associated with those updates.

4.3.10 Rejecting Service Past an LPP Installation

You should install your system completely before ever doing service on it. You should also have all LPPs installed in your cluster before trying to apply service. If this is not possible, you should back up your root filesystem for your primary site, and the locals for all your other sites in a cluster before installing any new LPP. In addition, if you intend to remove old service beyond the layer that was a new LPP installation, you should back up the system at that point also. If you wish to reject service beyond an update, you can do so by restoring the primary site to its original level before the LPP was installed. This must be done with the primary site removed from the network by using the **clusterstop** command. At this point, you can then reject any other service to the level you desired. You may also apply and commit any later service you wish to reapply. Use the latest backup of the system to retain any customized files.

Note: In order to complete this re-installation of your system, you need to perform the processes to make all sites in your network repropagate their root filesystems.

4.4 Applying Updates

Updated files can be applied to a file system by using the **updatep** command. These updates are stored along with the original system files which can be recovered later, if necessary. The original system files are saved in **/usr/lpp/***lpp_name***/inst_updt.save**.

Notes:

1. You can only have one update in the applied state at any time. If you have already applied an update to the system, that update can be committed by using the **-c** flag.

2. The update process is disk-space intensive. Before applying an update, you should determine whether or not you have enough space on your system to continue. The documentation provided with the update describes what space you will need.

Normally your cluster or system should be in a quiescent state while installing updates. You should expect some degradation in performance while the update process is occurring. Note that each of the files installed by the update are being automatically propagated to each site in your cluster. Some updates will require the local of each site to be updated (special functions, new kernels, etc). Although the update mechanism will allow you to start these local updates, you will have to start this process for any site in your cluster that is not currently up.

Consult the Licensed Program Product (LPP) documentation to find out if an LPP requires a quiescent system. If it does, restart the system, keeping it in single-user mode, and make sure no other processes are running.

Note: Log in as root or be the superuser.

To apply an update to the system, follow these steps:

1. Mount the distribution tape.

2. At the AIX prompt (#), enter:

```
updatep -a -d /dev/xxx
```

   The **xxx** is the input device name. Messages similar to the following are displayed as the system begins restoring files:

```
+-------------------------------------------------------------------------
¦
¦ File System Restore Utility
¦ x ./usr/sys/inst_updt/special
¦ x ./usr/sys/inst_updt/control
¦ x ./lpp_name
¦     files restored: 3
¦ 048-116 The system has completed the update validation of the programs.
¦  The system is checking to see whether any programs require special
¦  functions or must be updated together. Please wait.
¦
¦
¦
¦
¦
¦
¦
```

The system then displays the "Apply Updates Menu" which is similar to the following screen. From this menu, you can choose the programs you wish to update:

```
048-082 Apply Updates Menu

        No Special Function Required

         ID UPDATE

         1 PROGRAM1 Program

        To cancel the updatep command enter "quit".

        You may select one or more numbers from the list.

        Type the ID numbers of the programs you wish to apply separated
        by spaces (example: 1 3) and press Enter.

        --->
```

3. From this menu, choose the number corresponding to the programs you wish to update. If more than one program is to be updated, enter the numbers for the programs separated by spaces.

   Notes:

   a. The options listed on this menu depend on the programs currently installed on the system.

   b. If you select more then one update at this point, those updates will be applied, committed, uncommitted, and rejected as a whole.

4. Update a program listed on this menu by entering the number corresponding to the program to be updated.

   After you press **Enter**, the system begins to apply the updates. Messages similar to the following are displayed:

   Note: The exact content of these messages depends on the programs chosen to be updated.

```
+-------------------------------------------------------------------------
¦
¦ 048-089 The system is now starting to apply the updates for program
¦         "PROGRAM1 Program".
¦ File System Restore Utility
¦ x ./usr/lpp/PROGRAM/lpp.loc/info
¦ x ./usr/lpp/PROGRAM/lpp.loc/lpp.hist
¦ x ./usr/lpp/PROGRAM/lpp.loc/inst_updt.loc
¦     files restored: 3
¦ File System Restore Utility
¦ x ./usr/lpp/PROGRAM/inst_updt/arp
¦     files restored: 1
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------
```

   After these messages, the system displays copyright information.

5. If the update being applied is an upgrade, you see messages indicating that the system is saving files written over by this update procedure. You can retrieve these files later, if necessary. When the last message is displayed, go to step 7.

```
+-------------------------------------------------------------------------
¦
¦ File System Restore Utility
¦ x ./usr/lpp/PROGRAM/lpp.loc/files/local/PROGRAM1
¦ x ./usr/lpp/PROGRAM/lpp.loc/files/local/PROGRAM3
¦ x ./usr/lpp/PROGRAM/lpp.loc/files/local/PROGRAM4
¦     files restored: 3
¦    -    File "/usr/bin/PROGRAM/PROGRAMx" is being saved.
¦    -    File "/usr/bin/PROGRAM1" is being saved.
¦    -    File "/usr/bin/PROGRAM3" is being saved.
¦    -    File "/usr/bin/PROGRAM4" is being saved.
¦    -    File "/usr/bin/PROGRAM5" is being saved.
¦ File System Restore Utility
```

```
¦ Please mount volume 1 on /dev/xxx
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------
```

6. After you press **Enter**, you see messages similar to the following:

   Note: The exact content of these messages depends on the fixes you have chosen.

```
+-------------------------------------------------------------------------
¦
¦ File System Restore Utility
¦ x ./usr/lpp/PROGRAM/lpp.loc/files/local/PROGRAM1
¦ x ./usr/lpp/PROGRAM/lpp.loc/files/local/PROGRAM2
¦     files restored: 2
¦    -    File "/usr/bin/PROGRAM1" is being saved.
¦    -    File "/usr/bin/PROGRAM3" is being saved.
¦ File System Restore Utility
¦ Please mount volume 1 on /dev/xxx
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------
```

7. Press **Enter**.

   If there were updates to local file systems, the following messages are displayed:

```
+-------------------------------------------------------------------------
¦
¦ x ./usr/bin/PROGRAM1
```

```
¦ x ./usr/bin/PROGRAM3
¦ x ./usr/bin/PROGRAM4
¦ x ./usr/bin/PROGRAM5
¦ x ./usr/bin/PROGRAM/PROGRAMx
¦     files restored: 5
¦ File System Backup Utility
¦ Done     at Fri Jun 17 14:48:05 1988
¦ 40 blocks on 1 volume(s)
¦ 048-088 All requested update processing is completed.
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter site name followed by the desired execution time.
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution.
¦ (STOP for break)
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+--------------------------------------------------------------------------
```

All the necessary files have been updated on the replicated root file system.

8. Specify the site to which you are updating files and the time you want the system to update the files on each of the local sites by entering:

   sitename time

   The **sitename** is the name of the site on which the local files that are to be updated reside. The **time** is the time when the update procedure is to begin on the specified site. For example, to update a single site called **SITE1** at 7:00 p.m., enter the following:

   SITE1 7:00 p.m.

   To update local files on all the listed sites, enter **all.** To begin the update immediately, enter **now**. If **now** is specified for the update time, the system performs the update immediately. If necessary, the system rebuilds the kernel and reboots the site.

   After the site name and time are entered, the system responds with possible sites.

9. Enter STOP in all uppercase letters (as shown) at the list of available sites. The **updatep** program exits and the AIX prompt (#) returns.

Notes:

1. For all other sites that were not available for update at this time, you must run **qproc** by hand. When you do this, it is suggested that you have the system nearly idle. If you are in a cluster environment, it is required that you have access to a primary or backbone site to

update your site's local.

2. If updating local files that require rebuilding the kernel and rebooting the site, update local files on the machine that is running **updatep** by specifying that machine last or specifying a future update time.

4.5 Committing Updates

Once an update has been applied, the update should be committed to make it a permanent part of the system. Committed updates are stored more permanently than applied updates.

When an update is committed, the files existing prior to the update are moved from the file **/usr/lpp/***lpp_name***/inst.updt.save** to another location. This makes the system ready for new updates to be applied.

Note: Log in as root or be the superuser.

To commit an update to the system, follow these steps:

1. At the AIX prompt (#), enter:

```
# updatep -c
```

   The system displays the "Commit Updates Menu" which is similar to the following:

   Note: The options listed on this menu depend on the programs for which updates are currently applied on the system.

```
+-------------------------------------------------------------------------
¦
¦ 048-082 Commit Updates Menu
¦
¦ No Special Function Required
¦
¦     ID UPDATE
¦
¦ PROGRAM1 Program
¦
¦    To cancel the updatep command enter "quit".
¦
¦    You may select one or more numbers from the list.
¦
¦    Type the ID numbers of the programs you wish to commit separated
¦    by spaces (example: 1 3) and press Enter.
¦
¦    --->
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------
```

   From this menu, choose the programs for which updates are to be committed.

2. Commit an update for a program listed on this menu by entering the number corresponding to the updated program to be committed. To

commit updates for more than one program, enter the numbers for the programs separated by spaces.

After you press **Enter**, messages similar to the following example are displayed indicating that the system is saving the files that are being overwritten. These files can be retrieved later if necessary.

Note: The exact content of these messages depends on the updates you have chosen to commit.

```
+-------------------------------------------------------------------------
¦
¦ Saving /usr/lpp.save/2/PROGRAM to /usr/lpp.save/2/PROGRAM
¦ File System Backup Utility
¦ Done     at Fri Jun 17 14:52:08 1988
¦ 40 blocks on 1 volume(s)
¦ Saving status /usr/sys/inst_updt to /usr/lpp.save/2/status
¦ File System Backup Utility
¦ Done     at Fri Jun 17 14:52:23 1988
¦ 40 blocks on 1 volume(s)
¦ 048-089 The system is now starting to commit the updates for program
¦         "PROGRAM1 Program".
¦ 048-088 All requested update processing is completed.
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------
```

During the above operation, you will see the message:

```
  Saving /usr/lpp/... to /usr/lpp.save/NN/...
```

You should record the number displayed in the message. This frame number is used when using the **cleanup** command described later. This command allows you to manage the amount of disk space used by the service system.

All the necessary files have been committed on the replicated root file system. If there were local updates, the system then displays the following message, which asks you to specify the sites to which you wish to commit updates and the time you wish the procedure to begin.

```
+-------------------------------------------------------------------------
¦
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter site name followed by the desired execution time
```

```
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution
¦ (STOP for break)
¦
+------------------------------------------------------------------------
```

3. Specify the site to which you are committing updates and the time you want the system to commit the updates by entering:

   sitename time

   The **sitename** is the name of the site to which you are committing updates. To commit updates to local files on all the listed sites, enter **all**. The **time** is the time when the update procedure is to begin on the specified site. Enter times using the syntax of the **at** command. To begin the commit immediately, enter **now**.

   After you enter all of the site names and times, the system responds with possible sites.

4. Enter STOP in all uppercase letters (as shown) at the list of available sites. The **updatep** program exits and the AIX prompt (#) returns.

Subtopics

4.5.1 Cleaning Up Updates

During the apply or commit phases of service, you are using up disk space on your system in order to preserve the older versions of commands and library members. If you wish, you may use the **cleanup** command to manage some of this space.

During the commit phase, you were asked to record certain information in a save-stack frame. The **cleanup** command can be used to archive one or more of these to tape, or to restore them later if you need them in order to restore previous service levels.

If at a later time you wish to uncommit some update on which you have used the **cleanup** command, you will have to restore it to the system. Use **cleanup -r** *frame number* to retrieve this information.

Subtopics
4.5.1.1 Other Notes

4.5.1.1 Other Notes

If problems are found during the update process, that level of service will automatically be rejected, and the system returns to its original state.

Service is applied in this system with a Last-In-First-Out strategy. That is, the last service you applied is the only one that you can manipulate until that step is rejected. Hence, in order to remove service levels that were committed earlier, you will have to remove any subsequently applied service in the same order that it was applied. The update commands will help you with this, notifying you of previous service that must be removed before some particular level of service is applied.

When service is applied together for multiple Licensed Products, that service is manipulated as a whole, and must then be committed, uncommitted, or rejected together.

4.6 Uncommitting Updates

Uncommitting an update reverses the process performed when the update was committed. When an update is uncommitted, the system returns to the same state as when the update was applied and the files existing prior to the update are moved back to **/usr/lpp/***lpp_name***/inst.updt.save**.

Note: Log in as root or be the superuser.

To uncommit an update on the system, follow these steps:

1. At the AIX prompt (#), enter:

   ```
   updatep -u lpp_name version_number
   ```

   The **lpp_name** is the name of the program to which the update or fix is to be uncommitted. The **version_number** is the number of the update you wish to uncommit.

   The system displays the following message:

   ```
   +-------------------------------------------------------------------------
   ¦
   ¦ 000-123 Before you continue, you must make sure there is no other activi
   ¦         on the system.  You should have just restarted the system, and r
   ¦         other terminals should be enabled.  Refer to your messages refer
   ¦         book for more information.
   ¦
   ¦         Do you want to continue with this command?  (y or n)
   ¦
   ¦         --->
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   +-------------------------------------------------------------------------
   ```

2. Enter **y**.

   The following messages are displayed as the system begins the uncommit procedure.

   Note: The exact content of these messages depends on the updates that you have chosen to uncommit.

   ```
   +-------------------------------------------------------------------------
   ¦
   ¦     The following LPPs need to be uncommitted and rejected for this upda
   ```

```
¦     PROGRAM  00.00.0020.0000 use: updatep
¦ LPPs committed together, and will be all uncommitted
¦ Uncommit proceeds ....
¦ File System Restore Utility
¦ x .
¦ x ./lpp.loc
¦ x ./lpp.loc/info
¦ x ./lpp.loc/lpp.hist
¦ x ./lpp.loc/inst_updt.loc
¦ x ./lpp.loc/files
¦ x ./lpp.loc/files/local
¦ x ./lpp.loc/files/local/PROGRAM1
¦ x ./lpp.loc/files/local/PROGRAM3
¦ x ./lpp.loc/files/local/PROGRAM4
¦ x ./lpp.loc/al.loc
¦ x ./lpp.hist.bak
¦ x ./lpp.hist
¦ x ./sl
¦ x ./appscr.0005
¦ x ./lpp.uninst
¦ x ./rejscr.0005
¦
¦


¦
¦ x ./inst_updt.save
¦ x ./inst_updt.save/update.list
¦ x ./inst_updt.save/update.1
¦ x ./inst_updt.save/update.2
¦ x ./inst_updt.save/update.3
¦ x ./inst_updt.save/update.4
¦ x ./inst_updt.save/update.5
¦ x ./al
¦     files restored: 25
¦ File System Restore Utility
¦ x .
¦ x ./inutemp.ndc
¦ x ./inutemp.dc
¦ x ./lpp_name
¦ x ./inutemp.vrmc
¦ x ./inutemp.mnuf
¦ x ./inutemp.max
¦ x ./updt_cntrl
¦ x ./inutemp.tmp3
¦ x ./inutemp.tmp
¦ x ./lppsize
¦ x ./inuapply.ok
¦ x ./inuname.tmp
¦     files restored: 13
¦
¦
+-------------------------------------------------------------------------
```

Updates to a program must be uncommitted in reverse order from the order in which they were committed. In other words, if updates were committed in the following order:

```
update1
update2
update3
```

then to uncommit update1, you must first uncommit update3 followed by update2.

If updates are uncommitted in the wrong order, the following error message is displayed:

```
+-----------------------------------------------------------------------
¦
¦     The following LPPs need to be uncommitted and rejected for this upda
¦     PROGRAM  00.00.0020.0000 use: updatep
¦ You must first uncommit/reject the above LPPs
¦
+-----------------------------------------------------------------------
```

If you receive this message, uncommit the updates listed in the error message before continuing the procedure.

When the uncommit procedure is completed on the replicated root file system, the following message is displayed:

```
+-----------------------------------------------------------------------
¦
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter site name followed by the desired execution time
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution
¦ (STOP for break)
¦
```

3. Specify the sites to which the update should be uncommitted and the times when the uncommit procedure should begin by entering:

   sitename time

   The **sitename** is the name of the sites on which the updates are to be uncommitted on the local files. To uncommit updates to local files on all the listed sites, enter **all**. The **time** is the time at which the uncommit procedure is to begin on the specified site. Enter times using the syntax of the **at** command. To begin the uncommit immediately, enter **now**.

   After entering the sitename and time, the system responds with available sites.

4. When finished specifying sites, enter STOP in all uppercase letters (as shown) at the list of available sites. The **updatep** program exits and the prompt returns.

Subtopics
4.6.1 Example

4.6.1 Example

Uncommitting updates on a single site called SITE1 at 7:00 p.m. may be similar to the following example:

```
+-------------------------------------------------------------------------------
¦
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter the site name followed by the desired execution time.
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution.
¦ (STOP for break)
¦ SITE1 7 p.m.
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter the site name followed by the desired execution time.
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution.
¦ (STOP for break)
¦ STOP
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------------
```

4.7 Rejecting Updates

Rejecting an update reverses the process performed when the update was applied. When an update is rejected, the system returns to the same state as it was before the update was applied.

When an update is uncommitted, the files existing prior to the update are reconstructed using the information stored in the **/usr/lpp/***lpp_name***/inst_updt.save** file. This directory is then deleted.

When you reject an update that has already been committed, you must first uncommit that update.

Note: Log in as root or be the superuser.

To reject an update to the system, follow these steps:

1. At the AIX prompt (#), enter:

   ```
   updatep -r
   ```

   The system displays the following message:

   ```
   +-------------------------------------------------------------------------
   ¦
   ¦ 000-123 Before you continue, you must make sure there is no other activi
   ¦         on the system.  You should have just restarted the system, and r
   ¦         other terminals should be enabled.  Refer to your messages refer
   ¦         book for more information.
   ¦
   ¦         Do you want to continue with this command?  (y or n)
   ¦
   ¦         --->
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
   +-------------------------------------------------------------------------
   ```

2. Enter **y**.

   The system then displays the "Reject Updates Menu", which is similar to the following. The options listed on this example menu depend on the updates currently applied on the system.

   ```
   +-------------------------------------------------------------------------
   ¦
   ¦ 048-082 Reject Updates Menu
   ```

```
No Special Function Required

 ID UPDATE

  1 PROGRAM1 Program

To cancel the updatep command enter "quit".

You may select one or more numbers from the list.

Type the ID numbers of the programs you wish to reject separated
by spaces (example: 1 3) and press Enter.

--->
```

Enter the number corresponding to the update you want to reject. If more than one update is to be rejected, enter the numbers for the updates separated by spaces.

After you press **Enter**, messages similar to the following are displayed:

Note: The exact content of the messages depends on the updates that are to be rejected.

```
048-089 The system is now starting to reject the updates for program
        "PROGRAM1 Program".
   -    File "/usr/bin/PROGRAM/PROGRAMx" is being recovered.
   -    File "/usr/bin/PROGRAM1" is being recovered.
   -    File "/usr/bin/PROGRAM3" is being recovered.
   -    File "/usr/bin/PROGRAM4" is being recovered.
   -    File "/usr/bin/PROGRAM5" is being recovered.
048-088 All requested update processing is completed.
```

When you see the message indicating that the processing is completed, the indicated updates have been rejected on the replicated root file system.

3. Specify the site on which you want updates rejected and the time you want to reject the updates by entering:

   ```
   sitename time
   ```

   The **sitename** is the name of the sites on which updates are to be rejected to local files. To reject updates to local files on all the listed sites, type **all**. The **time** is the time at which the reject procedure is to begin on the specified site. Enter time using the syntax of the **at** command. To begin the reject immediately, enter **now**.

4. After the sitename and time are entered, the system responds with available sites. When finished specifying sites, enter STOP in all uppercase letters (as shown) at the list of available sites. The **updatep** program exits and the prompt returns.

Subtopics
4.7.1 Example

4.7.1 Example

Rejecting updates on all sites on a system immediately may be similar to the following example:

```
+-------------------------------------------------------------------------------
¦
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter the site name followed by the desired execution time.
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution.
¦ (STOP for break)
¦ all now
¦ The following sites are available:
¦ SITE1 SITE2 SITE3 SITE4 SITE5
¦ Please enter the site name followed by the desired execution time.
¦ You can enter "all" to choose all sites and "now"
¦ for immediate execution.
¦ (STOP for break)
¦ STOP
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
¦
+-------------------------------------------------------------------------------
```

If now is specified for the reject time, the system performs the reject immediately. If necessary, the system rebuilds the kernel and reboots the site when ENTER is pressed.

Note: If rejecting updates to local files that require rebuilding the kernel and rebooting the site, first reject updates to local files on the machine that is running the **updatep** command, then specify that machine last or specify a future reject time. If rejecting updates to local files that require rebuilding the kernel and rebooting the site immediately (by typing **now** as the reject time) on the machine on that is running **updatep**, the **updatep** procedure will be stopped when the kernel is rebuilt and the machine rebooted; therefore, STOP does not have to be entered.

5.0 Chapter 5. Solving System Problems

Subtopics

5.1 Contents

5.2 About This Chapter

The operator at the console only handles a limited number of system problems (for example, freeing a jammed line printer). The administrator is called on to solve more serious problems. The following chapter provides information on:

- Killing a runaway process with the **kill** command
- Replacing a forgotten passwor
- Removing hidden file
- Restoring free spac
- Restoring lost system file
- Restoring an inoperable system

For more information on system problems, refer to the *AIX/370 Diagnosis Guide*.

5.3 Killing a Runaway Process

A runaway process is a program that cannot be terminated from the terminal at which it was started. This occurs whenever an error in the program locks up the terminal, thereby preventing anything entered on it from reaching the system.

To stop a runaway process at any site, follow these steps:

1. Go to a terminal that is not locked up.

2. Log in as the superuser.

3. If logged in on the site with the runaway process, enter:

   ```
   ps -a
   ```

   This command lists the processes on the site where you are logged in. If logged in on a site other than the site with the runaway process, enter:

   ```
   onsite <site> ps -a
   ```

   where **<site>** is the sitename where the runaway process is operating.

   This command lists processes on the specified site and displays all current processes with their process identification numbers (PID).

4. Find the PID of the runaway program.

5. Enter:

   ```
   kill <PID>
   ```

   The runaway program should stop in a few seconds but may leave temporary files or a non-echoing terminal. If the process does not terminate, enter:

   ```
   kill -9 <PID>
   ```

5.4 Replacing a Forgotten Password

The AIX/370 Operating System does not provide a way to decipher an existing password. If a user forgets a password, the system administrator must change the password to a new one. To change an ordinary user password, refer to *Managing the AIX Operating System*.

5.5 Removing Hidden Files

A hidden file is any file whose name begins with a dot (.). Hidden files in a directory can be listed by entering the following command:

```
ls -a
```

Most hidden files can be removed from a directory by entering:

```
rm .[a-z]*
```

Warning: Do not use **rm .*** as this will also attempt to remove the special directories . (dot) and .. (dot dot).

Remove remaining files and directories individually.

Note: There are many useful hidden files in users' home directories which should not be removed. Be certain that these files are not accidentally removed.

5.6 Restoring Free Space

The system displays a message whenever a file system has little or no space. To restore system operation, one or more files must be deleted from the identified file system. The system message identifies the file system by the pathname of its mount point.

The most common file system where space problems occur is the **/tmp** file system. To remove the old **/tmp** files, enter:

```
find /tmp -mtime +7 -atime +7 -exec rm -f {} \;
```

This example removes files that have not been used in seven or more days. The **skulker** command can also be used for this purpose.

The users should be warned that some of their temporary files will be removed. The **wall** command can be used to inform the users.

5.7 Restoring Lost System Files

If a system program or data file is accidentally modified or removed from the root file system, it must be restored. If the file is only missing in a non-primary copy of the file system, force it to re-propagate by using the command:

```
touch <filename>
```

If the file is lost in the primary copy of the file system, recover the file from the periodic backup. It will propagate to other packs automatically.

For more information on restoring files, refer to *Managing the AIX Operating System*.

5.8 Restoring an Inoperable System

On rare occasions, one or more of the critical AIX system files may be accidentally modified or removed, thus preventing the system from operating. In this case, the AIX system and user program and data files must be restored from backup tapes or disks.

For more information on restoring the system, or on restoring files from backup tapes or disks, refer to the **backup** and **restore** commands in the *AIX Operating System Commands Reference*.

5.9 Other Problems

For problem determination and problem source identification, see the *AIX/370 Diagnosis Guide*. This book describes startup, shutdown and general system operation problems. It includes a description of useful tools for debugging and provides examples.

For information on routine system management and maintenance, see *Managing the AIX Operating System*.

6.0 Chapter 6. Performance Tuning

Subtopics

6.1 Contents

6.2 About This Chapter

The major focus of this chapter is performance considerations within the AIX system using TCF. Guidelines are presented that the system administrator can use in cluster configuration. In addition, system utilities that can be used to monitor system activity are described.

6.3 Introduction

The AIX system offers the system administrator considerable flexibility regarding configuration. Because all cluster sites are fully functional, it is possible to assign any or all functions to any site in the cluster. For example, one configuration option is to partition the user population and user data among all cluster sites in the cluster. This type of division results in each cluster site supporting both batch and interactive cycles to the users. A second configuration may involve shifting the user population so that there are certain cluster sites that support mostly interactive tasks and other cluster sites which are largely batch servers. In the latter configuration, the interactive cluster sites could be PS/2 machines running AIX since they are better able to support full screen applications like the **vi** editor. In addition, as the user population grows, additional batch and interactive machines can be easily added to the cluster. New users are provided access to the cluster via a new PS/2 machine. As the batch load grows, an additional 370 machine can be brought online, and in many cases a portion of the offered batch load can automatically be redirected to this new AIX/370 cluster site. These two general configurations are some of the options available.

Each installation has specific needs which may generate different performance characteristics. No single configuration will be the optimal configuration for all installations. The goal of this chapter is to present the system administrator with enough understanding of some of the choices so that decisions can be made. In general, some experimentation with different configurations may be necessary and may help the system administrator understand the needs and usage characteristics of the user population.

The remaining portion of this section defines a few critical terms related to performance.

In order to compare the performance of a task in two different environments or configurations, there must be some measure of the performance of the task in each environment. They are:

Elapsed time
: The "wall clock" time required for a task to complete. It makes no statement about the amount of CPU cycles expended to complete the task.

CPU time
: The actual number of instructions or processor cycles used to complete the task. One way to compute CPU time is to use the **time** command and add the system and user times together. CPU time attempts to measure the actual amount of time the processor was executing on behalf of the particular task.

Response time
: The reply to a user's request. An example would be the "wall clock" delay a user experiences in **vi** when a character is typed.

Cost
: CPU time expended normalized into some monetary factor (for example, CPU time * Process cost / Processor instructions-per-second).

Note that a task with a long elapsed time may consume only a small amount of CPU time and provide the user with long response times. An example would be an editor session.

Improving performance and making a task faster means that by some measure the performance behavior of the task has improved. It could mean the response time in the editor is faster, the elapsed time of a compiler is shorter or the CPU time consumed by a user application is reduced.

It is also useful to have a measure of a processor's current free processing capability. The measure used in this chapter is relative load factor. The load on a processor is the number of processes capable of running that are waiting to execute. This number is measured at periodic intervals. In order to have a measure that is independent of processor speed, the load for a given processor is normalized to produce the relative load factor. The normalization constant is obtained from the **/etc/site** file. The system administrator is responsible for placing a value in the **/etc/site** file for each cluster site that is a reasonable measure of the performance of a given site compared to the other cluster sites.

6.4 Performance Considerations

Principle targets for performance tuning are:

- Main storag
- Central processor unit (CPU
- Expanded storage (if available
- Type and number of paging devices availabl
- AIX/370 file systems

For an overview of these resources within a total system, refer to Figure 6-1.

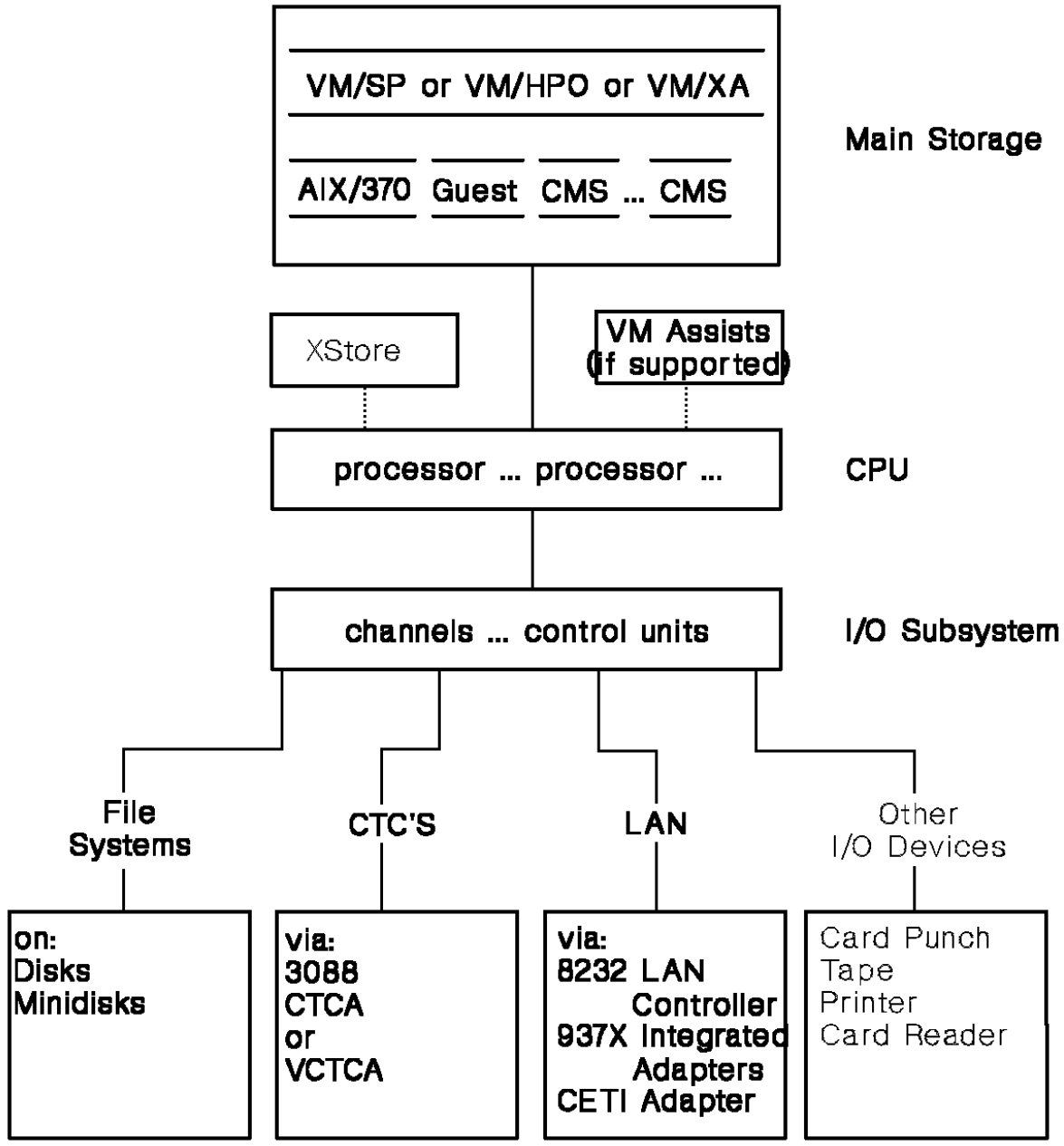

Figure 6-1. System Overview

The performance of AIX/370 is determined by factors on different levels of the system. Some factors are:

- Hardware configuratio
- VM performanc
- AIX/370 performanc
- Work loa
- Application performanc
- Cluster balancin
  - Where users login
  - Where user files are stored.

When planning for tuning and capacities, consider each level of the system (CPU, VM, AIX/370, and file systems). Consider the performance factors for the following system levels:

Table 6-1. Performance Factors at Different System Levels

Level	**Performance Factor**
CPU	AIX/370 requires a minimum of 4M bytes (6M bytes is recommended) of real storage for the AIX/370 virtual machine. Performance can be improved by providing additional real storage.
VM	V=R environment V=V environment V=F environment (VM/XA only).
AIX/370	Size of AIX/370 buffered I/O cache Number of hash buffers. **Note:** These quantities are dependent on the size of the virtual machine allocated to AIX/370.

Subtopics
6.4.1 Tuning AIX/370 for Large Program Usage
6.4.2 Tuning Main Storage at VM Level
6.4.3 Tuning Main Storage at AIX/370 Level
6.4.4 Tuning Central Processor

6.4.1 Tuning AIX/370 for Large Program Usage

While system tuning is always an issue with typical UNIX systems, it is especially critical when a system is to be used in a specialized capacity. AIX/370 presents special tuning considerations since it runs as a guest operating system under VM.

The type of system to be considered is one that typically runs multiple compute-intensive processes with very large working sets. Of course, the term "large" is ambiguous since it is dependent on the hardware architecture, that is, dependent on the available real memory which is constrained by the virtual address space and the I/O subsystem for the processor. A process with a 6M byte working set would not be considered large on a S/370 XA system configured as V=V with 256M bytes but would be on a non-XA system with only 16M bytes of storage defined. The I/O subsystem is an issue because of throughput performance. Any process whose working set exceeds the throughput that the processor can process in one second should be considered large. This convention is somewhat arbitrary and based on the idea that if and when a process needs to be swapped, the time taken to swap it exceeds an acceptable interactive response time limit. For instance, the PS/2 ESDI controller can move about 750K bytes/second, while the 370 channel performs somewhere around 5M bytes/second. This difference can have an obvious effect in determining the meaning of "large working set".

With jobs that have very large working sets, for instance 20M bytes or more, the system administrator's goal should be to tune the system so that such processes are not swapped, because whenever they are swapped, system response suffers. There are two important items to consider in configuring an AIX/370 system for such usage. One is the amount of storage is defined on VM. The second is the tuning of the pager for optimal system performance.

Because of the design of UNIX, if the combined working-set size of some number of large processes that are runnable exceeds the amount of real memory, the system thrashes, that is, it spends all or nearly all its cycles swapping these jobs in and out and nearly no time actually running them. To avoid this problem, it is critical that the administrator analyze memory requirements for the combined users of the system and configure the storage accordingly. You can get an idea of the working set for a process by using the **ps** command.

The second critical system feature to tune is the pager. The pager is a kernel process which periodically checks free memory. If it is less than a certain water mark, the pager cycles through process vsegs and free unreferenced pages. It continues this operation until it reaches another water mark. This procedure trims the working set of user processes and assures that an adequate pool of free pages is available for paging demands. Keeping this pool large enough for system demand is the key to avoiding excessive swapping.

The water marks mentioned above are configurable and are called **gpgslo** and **gpgshi**. These parameters are given defaults in the **/etc/master** file. The **gpgslo** parameter is the number of free pages where the pager should begin its page-stealing activity. The **gpgshi** parameter is where the pager should be satisfied and stop. The default values for these parameters are oriented toward small systems; the **gpgslo** parameter is set to 10, and the **gpgshi** parameter is set to 20. For larger 370 systems these values are far too small. There are several ways to approach changing these values. If either of the two parameters is set to 0, an autoconfiguration takes

place. A formula is applied to the amount of real memory such that some fraction of it is used to get **gpgslo** and **gpgshi**. The problem with this approach is that there is a hard limit set in the autoconfig code so that, regardless of how large memory may be, **gpgshi** is limited to 128 and **gpgslo** is limited to 32. Of course this is better than 10 and 20, but still not realistic on a large system. The only real option for such systems is to set these values manually. The recommendation is that **gpgslo** be set to 10% of defined storage and **gpgshi** be set to 15%. Note that these are the number of 4K pages in memory, so this value needs to be calculated from the total bytes.

With the pager properly tuned, large jobs should have their working sets trimmed and thus be kept from being swapped. Naturally, since very few systems are static, the system administrator must monitor performance and dynamically alter these parameters when needed.

6.4.2 Tuning Main Storage at VM Level

There are a number of performance considerations when the AIX/370 Operating System is running as a guest under VM. Some general considerations that apply to all memory modes are:

There are several VM flags that control when warnings are displayed o the console. These flags and their correct settings are:

MSG,OFF EMSG,ON IMSG,OFF SMSG,OFF WNG,OFF

Selected pages of a virtual machine can be locked in real storage This is done with the LOCK command. Use this feature carefully since resource deprivation can occur if used inappropriately. Reasonable pages to lock are:

- Page 0 of the virtual machine
- Kernel file system buffer cache
- Kernel text modules
- All of the kernel.

SET PAGEX ON to activate the pseudo page fault facility

Altering priorities for AIX is not recommended

Subtopics
6.4.2.1 The V=R Environment
6.4.2.2 The V=V Environment
6.4.2.3 The V=F Environment

6.4.2.1 The V=R Environment

Tuning Parameters for VM/SP

After the IPL of AIX/370 on the virtual machine, use VM SET command to **SET STBYPASS VR** and **SET NOTRANS ON**.
On VM/SP HPO, **SET CCWTRANS OFF**.

After the AIX/370 guest has logged on, use VM SET commands to **SET FAVOR name** (without percent). Using this VM SET command eliminates any wait for storage. In addition, use **SET FAVOR** with a percent to request a specific percentage of the CPU for the virtual machine.

QDROP OFF

SET ASSIST ON SVC for various assists.

SET TIMER OFF since VM/SP supports a virtual timer.

Tuning Parameters for VM/XA
Put the following statements in the directory of the guest:

OPTION QUICKDSP
Enable quick dispatch for this guest and keep it off of the VM eligible list.

SHARE ABSOLUTE nn
Set the share of the guest. A minimum of 4% absolute share is suggested. Use SHARE sparingly to prevent CPU resource deprivation.

IUCV ANY PRIORITY MSGLIMIT 255
This statement sets a message queue limit of 255. The **ANY** option allows any other VM guest to connect to this guest.

IUCV *CSS PRIORITY MSGLIMIT 255

OPTION MAXCONN 16
This statement specifies a maximum of 16 IUCV connections. These last two set up the proper environment for use of an IUCV communication channel between two AIX/370 guests or AIX/370 and VM TCP/IP.

MACHINE XA
Enable XA mode for this guest.

CPU 01 DEDICATE
Dedicate an available processor for exclusive use by this guest in certain environments. This can be dynamically changed while the guest is running as the load requires. Use DEDICATE sparingly to prevent CPU resource deprivation.

SET TIMER OFF since VM/XA SP supports a virtual timer.

After the guest is already running, the following parameters can be run using the SET command if they are not included in the directory:

QUICKDSP ON

SHARE

NOTRANS ON.

The VM Dispatch slice may also be decreased by issuing the command:

```
SET SRM DSPSLICE
```

This command permits better timeslicing between multiple guests under heavy loads and affects the dispatch slice for base VM operating system.

Notes:

1. Dedicated DASD devices will get significant performance improvements for preferred guests over VM minidisks.

2. The frequency of the VM dispatcher queue rescan should not be decreased if more than one AIX guest is being supported on the same real CPU.

To check the share given a guest under XA, issue the command:

```
QUERY SHARE userid
```

This applies to all storage modes.

Monitoring
To verify that all settings are correct, issue the following commands in the AIX/370 virtual machine:

QUERY VIRTUAL STORAGE (to check the amount of virtual storage allocated to the AIX/370 virtual machine)

QUERY SET (to check the setting of other **values** as determined by the installation)

INDICATE FAVOR (1) (to display the setting of the VM ***Favored Execution Option*** - VM/SP only)

INDICATE USER name (to display statistics about the user)

Note: For a V=R user, this shows a count of zero for ***working set*** and ***resident pages***.

The **INDICATE** command (with no parameters), returns information on the loads and state of the processor(s).

(1) This is a privileged command. If the AIX system has been given extensive privileges under VM, it may be issued by the AIX user. If no such extensive permissions have been given (which may be the more typical situation), the command must be issued by a privileged user (for example, OPERATOR).

6.4.2.2 The V=V Environment

Tuning Parameters for VM/SP

Carefully tune the size of the virtual machine. Be sure to keep th size consistent with the amount of real memory available to support the virtual machine.

Note: AIX/370 cannot be expected to execute effectively in a virtual machine of less than 4Mb.

QDROP OFF

After the AIX/370 guest has logged on, use VM SET commands to **SET FAVOR name** (without percent). Using this VM SET command eliminates any wait for storage. In addition, use **SET FAVOR** with a percent to request a specific percentage of the CPU for the virtual machine.

Allow for enough VM free space to build shadow tables by using the V SET commands (for VM/SP only):

- For maintaining the last **n** shadow tables copies, issue:

  SET STMULTI n USEG xx

  where:

  n is less or equal 6 (VM)
  n is less or equal 16 (VM HPO)
  USEG defines the size of the contiguous pre-allocated pool of shadow page tables
  xx is less or equal 99

- **CSEG** option cannot be used for AIX/370

- If the AIX/370 virtual machine is to be preferred for storage allocation, issue: **SET QDROP name OFF** and **SET FAVOR name** (1)

In a heavy-paging environment, issue

LOCK name 0 0 (page zero) (1)
LOCK name first page last page (1) (for AIX/370 shared buffers)
where **name** is the userid of the virtual machine
SET RESERVED nn

Note: The **nn** (number of pages) to be reserved could be the average working set size. These pages will be available to other guests but are only to be used as a last resort. Do not reserve more than available real memory.

SET ASSIST ON SVC for various assists.

SET TIMER OFF since VM/SP supports a virtual timer.

The following information is the result when the Q SET command is issued on VM/SP for a V=V virtual machine:

```
MSG OFF, WNG OFF, EMSG ON, ACNT ON, RUN OFF
LINEDIT ON, TIMER OFF, ISAM OFF, ECMODE ON
```

```
ASSIST ON SVC NOTMR, PAGEX ON, AUTOPOLL OFF
IMSG OFF, SMSG OFF, AFFINITY NONE, NOTRANS OFF
VMSAVE OFF, 370E OFF
STBYPASS OFF, STMULTI 3/3 00/000
MIH OFF, VMCONIO OFF, CPCONIO OFF, SVCACCL OFF, CONCEAL OFF
```

Tuning Parameters for VM/XA

Utilize Expanded Storage to improve the VM paging rate. (Expande Storage is available on the 3090 only.)

SET QUICKDSP ON

SET TIMER OFF since VM/XA SP supports a virtual timer.

The tuning parameters described for the V=R environment also apply to V=V tuning.

Monitoring
Use data from VM Monitor to study the effects of parameter variations on the total system behavior.

To verify that all settings are correct, issue the following:

QUERY VIRTUAL STORAGE (to check the amount of virtual storage allocated to the AIX/370 virtual machine)

QUERY SET (to check the setting of other **values** as determined by the installation)

INDICATE FAVOR (1) to display the setting of the VM ***Favored Execution Option*** (for VM/SP only)

INDICATE USER name to display statistics about the user

The **INDICATE** command (with no parameters), returns information on the loads and state of the processor(s).

The following information is the result when the Q SET command is issued on VM/XA SP for a V=V virtual machine:

```
MSG OFF, WNG OFF, EMSG ON, ACNT OFF, RUN ON
LINEDIT ON, TIMER OFF, ISAM OFF, ECMODE ON
ASSIST OFF, PAGEX ON, AUTOPOLL OFF
IMSG OFF, SMSG OFF, AFFINITY NONE, NOTRANS OFF
VMSAVE OFF, 370E OFF
STBYPASS OFF, STMULTI 00/000
MIH OFF, VMCONIO OFF, CPCONIO OFF, SVCACCL OFF, CONCEAL OFF
MACHINE XA, SVC76 CP, NOPDATA OFF
CCWTRAN ON
```

(1) This is a privileged command. If the AIX system has been given extensive privileges under VM, it may be issued by the AIX user. If no such extensive permissions have been given (which may be the more typical situation), the command must be issued by a privileged user (for example, OPERATOR).

6.4.2.3 The V=F Environment

Note: The V=F environment requires a 3090 with either the MPG or PR/SM hardware features.

Tuning Parameters
The parameters for the **V=F** environment are identical to the parameters for the **V=R** environment. See page 6.4.2.1.

Monitoring
Use data from VM Monitor to study the effects of parameter variations on the total system behavior.

To verify that all settings are correct, issue the following:

QUERY VIRTUAL STORAGE (to check the amount of virtual storage allocated to the AIX/370 virtual machine)

QUERY SET (to check the setting of other **values** as determined by the installation)

INDICATE USER name (to display statistics about the user)

The **INDICATE** command (with no parameters), returns information on the loads and state of the processor(s).

6.4.3 Tuning Main Storage at AIX/370 Level

General Considerations
AIX/370 allows a portion of storage to be reserved as a cache for disk buffered I/O.

Tuning Parameters

Define appropriate buffer size and number of hash buffers. In typical environment, reserve 33% of virtual machine storage.

Set the sticky bit for programs that are re-used

Operation
The AIX/370 administrator must configure the system file with the appropriate values for:

buffer
bhas
paging parameters

As a general rule, the sum of the space allocated to buffers and bhash should be approximately 33% of the virtual machine size. A more precise formula is **(buffers*3)+(bhash*128)=VM size/3**. The value of bhash is always less than or equal to the number of buffers.

Monitoring
Output from **sar** can be used to monitor the cache hit rate and exec rate. System accounting indicates which commands are commonly executed by users and on which sites they are executed. Ask the administrator to monitor these rates. The **lnetstat** and **pstat** commands can also be used to help assess the effects of tuning on cluster and file systems operations and the cluster communications load. Refer to the *AIX Operating System Commands Reference* for more information on the **lnetstat** and **pstat** commands.

6.4.4 Tuning Central Processor

Overview

Tuning at VM level

- To reduce CPU usage, have VM Assists available and activated

- To control CPU allocation between AIX/370 and other users, issue SET commands.

Tuning at AIX/370 level

- To reduce CPU usage:

  Reduce system calls
  Perform program load leveling
  Perform data access load leveling
  Move interactive loads to AIX PS/2 cluster sites and AIX Access for DOS Users
  Perform data access load leveling.

- To control process dispatching priority, use the **nice** command.

General Considerations

AIX/370 requires real time response to the LAN and therefore require frequent dispatching by VM
When running in mixed environments (for example, together with CMS an RSCS), the scheduling parameters of VM need adjusting
Performance tuning may be a trial-and-error process

Tuning Parameters

VM Assists are generally installed and enable

From the VM operator console, issue at any time

SET QDROP name OFF (VM/SP only)

SET FAVOR name nn% (VM/SP only) (1)
where **nn** is the percentage of CPU time that VM allocates to AIX/370; typically, **nn** is a value between 50 and 100.

SET PRIORITY name p (VM/SP only) (1)

SET SRM
This calls the System Resource Manager which allows the setting of a number of performance parameters.

Other SET commands as required.

Monitoring

To check VM Assists, issue: **QUERY SAS** and **QUERY CPA**

Use VM Monitor program to check the effects of SET commands on th total system behavior. This allows the VM system to be monitored and gives performance statistics on individual virtual machines and the

entire system. IBM offers SMART for VM/SP and RTM (Real Time Monitor) for VM/XA.

To check effects on AIX/370 response time and throughput, use **sar**.

Use the VM INDICATE command to determine the load on the physica processor(s).

(1) This is a privileged command. If the AIX system has been given extensive privileges under VM, it may be issued by the AIX user. If no such extensive permissions have been given (which may be the more typical situation), the command must be issued by a privileged user (for example, OPERATOR).

6.5 Performance Considerations for an AIX Cluster

Principal targets for performance tuning are:

AIX cluster file system
AIX cluster configuratio
Cluster considerations
- Tuning replicated file systems
- Where to store replicated file systems to enhance performance
- When to choose:
  replicated file systems over non-replicated file systems
  non-replicated file systems over replicated file systems.
- Balancing loads across the cluster for both interactive and non-interactive tasks
- File access balancing across the cluster
- User balancing across the cluster

Cluster balancin
- Where workstation users login
- Where AIX Access for DOS Users login
- Where user files are stored

Note: Be sure to read "Considerations for the System Administrator" in topic 1.4 and "MBCS Considerations" in topic 1.5 for necessary background information.

6.6 Data Location and Transfer Issues

AIX provides transparent access to data and processes, thereby permitting actions to be performed by cluster sites without concern for the location of the resources used by the actions. The transparency provided by AIX does not hide the performance differences that may result when the different selections are used. The result is that all options will result in the correct performance of the specified action but not all will be completed as quickly or with equal investments of CPU cycles.

Performance transparency is the concept of imperceptible differences in performance based on the location of tasks. AIX does not, in general, provide performance transparency. This does not imply that remote performance is slow; it does suggest that under some conditions, a user will be able to detect whether a task is running locally or remotely. In fact, performance transparency makes no statement about whether local or remote performance is superior. A system with faster remote than local data access would not be performance transparent. In the case of AIX, remote performance is very good but still there are cases where local performance is superior.

Subtopics

6.6.1 Selecting Job Execution Sites

One significant decision which must be made and which impacts system performance or user response time is the execution site for a given task. AIX permits a process on any site in the cluster to access data on any other machine in the cluster. Hence, there is a great deal of freedom available in the selection of the execution site for a process. (Note that this is a correctness or operational statement and not a performance statement.) It is unlikely that all cluster sites will complete the process in the same period of time, either because of differing machine speeds, differing loads on the cluster sites, or the comparative expense of data access.

Before a decision can be made about the execution site for a process, a basic understanding of the operational behavior, data location and program availability is required.

The operational behavior factor most useful to understand is whether a particular task is I/O intensive or CPU intensive. A task is I/O intensive if the bulk of the elapsed time of a process is spent waiting for data to be made available or for data to be written. As an example, one task that is I/O intensive is the copy command **cp**. It does very little processing of the input and output data but spends the bulk of its execution time reading or writing data (exactly how I/O intensive **cp** is will depend on the size of the files being copied). Not all I/O intensive tasks are doing only reads and writes. A command, like **find**, which will scan entire directory subtrees, is I/O intensive even though it does not do a large number of read or write operations. A CPU intensive (compute-bound) task is one which does a significant amount of processing with very little I/O interspersed. One example of a CPU intensive task is the UNIX command **troff**. The **troff** command does I/O but the bulk of the **troff** process execution time is spent formatting the words on the pages.

Some commands are not obviously I/O or CPU intensive or will change from I/O to CPU intensive depending on the input, so the general rules may have exceptions. Some experimentation with frequently used commands will help in understanding the characteristics of these commands.

A second criterion used in deciding where a task should be executed is the location of the input and output of that task. If a task consumes very little input and produces very little output, then clearly the location of the input and output data should not be a critical consideration in determining where the task is executed. If there is significant input or output data, then the **where** utility should be used to determine the location of the input and output data.

Another consideration in determining the execution site of a task is the machine types that can execute the command. If the command to be used only operates on a subset of machine types, then only machines of that type can be considered as possible execution sites. The **which** and **file** commands can be used to determine where an application can be run.

Note: Executing **which** can indicate the command was found in a hidden directory; the system places an @ at the end of the command and before the hidden directory component. Other files in that hidden directory should be the executable binaries for other machine types. For more information on the **which** command, refer to the *AIX Operating System Commands Reference*. For more information on hidden directories, refer to the *AIX Operating System Technical Reference*.

6.6.2 Selecting Execution Sites

When trying to determine the most suitable site for a task, a number of approaches can be taken. The simplest approach is to ask whether elapsed execution time of the task is significant. If the answer is no, the task can be run anywhere. If the answer is yes, a more careful decision may be appropriate.

Clearly this approach is a good one to use for most commands as most commands complete in a short period of time. If a command completes in a short period of time, then the execution site is not a significant factor.

A second approach which handles all tasks that are not I/O intensive reasonably well is implemented by the **fast** command.

Determine which cluster site has the least relative load. By default **fast** will only consider sites of the same type as those on which it is currently executing, but an option permits it to examine all sites.

Execute the task on that cluster site

For example, **fast** can be used as follows:

```
fast cc -O -o demo demo.c
```

In this case **fast** would select the least-loaded machine which is the same machine type as the local machine and execute the **cc** command on that cluster site.

If the command you wish to execute only operates on particular machine types, then **fast** and **fastsite** can still be used, but it should be run on a machine which can run the command. This can be done using the **onsite** command with the **fast** command.

For I/O intensive tasks, it is not particularly important to determine the relatively least-loaded site. The major consideration in the timely completion of an I/O intensive task is the location of the data.

Use the **where** command to determine the location of the input and output data.

If all the data is on a single cluster site, then execute the task o that cluster site.

If the input and output data are on two different cluster sites, the execute the task on the cluster site that will store the output data assuming that there is at least as much output as input data. If most of the data is input rather than output, then the task should be executed on the cluster site maintaining the input data.

If the data is replicated, then determine what cluster site has th primary copy of the file system with the output data. If the input data is available on the cluster site with the primary copy of the output data, then execute the task on that cluster site. Also use that site if at least half of the data transferred is output data. If more of the transferred data is input data rather than output data, the task should be executed on a cluster site with a copy of the input data. The only exception is when the site with the data is heavily loaded. In this case, it is sometimes faster to use a site with a lower load.

A few approaches have been presented which should help in the selection of execution sites for tasks. In some cases these approaches result in a relatively even load among all the sites in the cluster. Attempting to make all the sites in the cluster equally loaded (load leveling) is a desirable goal. The approaches outlined above will encourage this leveling and thoughtful decisions in configuration will also stimulate load leveling among all cluster sites.

Applications can be constructed to perform better in a distributed environment. One advantage of the distributed environment is the ability to do several functions in parallel on different machines. By doing functions in parallel, it is possible to decrease the elapsed time to complete the task. Tasks that can be broken down into several smaller tasks (sub-tasks) each executed in parallel can be optimized to use separate machines and improve performance. In selections where the sub-tasks should execute, the same algorithms as given above for the larger task can be used; however, an overall goal of maintaining parallelism (multiple execution cluster sites) among the sub-tasks should be considered.

6.6.3 Cluster Communication Traffic Considerations

AIX provides a transparent distributed system model of the system environment with excellent performance characteristics. As in all centralized and distributed systems, bottlenecks can be created by mistakes in configuration of the overall system. This section presents information to consider when the overall resource configuration is being prepared.

Cluster communication to or from a cluster site (Site A) is the result of one of the following:

Accessing data stored on Site A

Processes on other cluster sites communicating with processes on Sit A.

Propagation of changes to or from replicated files stored on Site A

File access synchronization of processes on Site A and other sites This occurs under the following conditions:

- Two or more processes on different sites try to read and write the same file.
- Two or more related processes access a file using the same file block.

Synchronization of open files stored on Site A (this is the loa generated from Site A being a Current Synchronization Site (CSS) for some of its file systems.).

Processes moving to or from Site A

Processes on Site A interacting with other sites

The cluster communication traffic patterns can be classified in two categories: dynamic and static. Dynamic traffic varies with system load and general usage patterns of the user population. For example, the selection of Site A as the execution site in the example above would generate dynamic traffic when it executed the **cc** command on Site A. Static traffic is the direct result of the configuration choices made in the setup of the cluster. For example, if Site A stores the primary copy of a replicated file system, then it must support the static traffic load of being the CSS for that file system. The point of defining static and dynamic traffic patterns is to separate the performance considerations into two classes. Addressing issues about static traffic loads will generally result in the reconfiguration of data (file systems) or users within the cluster. Dynamic traffic patterns can be changed through modification of the behavior of the user population. However, the dynamic traffic patterns will also change as a result of reconfiguration of data or users within the cluster. The **lnetstat** utility can be used to examine network load to a given site. For more information on the **lnetstat** command, refer to the *AIX Operating System Commands Reference*.

Avoid the static and dynamic traffic patterns from overloading any single cluster site. If a cluster site becomes a bottleneck with respect to cluster communication, then overall system performance will be degraded. By carefully locating file systems and users within the cluster, the administrator can prevent any one cluster site from becoming a bottleneck

under most conditions.

6.6.4 Raw-mode Terminal Access

A typical performance issue within a UNIX operating system is the support of applications which operate terminal lines in raw mode. Normally, when characters are typed on a terminal line, a complete line of input is collected before any application reading the terminal line is awakened and any echoing of characters is done by the system. In raw-mode, the application process is awakened on each character and it decides how a character should be echoed. The performance implications of applications which operate the input terminal line in raw mode is real and significant. Typical raw-mode applications are **vi**, **uucp**, **connect** and **ftp**.

In the AIX environment, the problem is addressed via the configuration flexibility of off-loading heavy raw-mode applications to AIX PS/2 machines. The result is that a relatively lightweight resource is consumed handling the raw-mode applications and the heavy computing can be handled by the AIX cluster sites that are better able to accommodate the volume.

By providing most AIX terminal access through AIX PS/2 machines, the raw-mode load will naturally fall on the AIX PS/2 machines. For example, a user who logs into a AIX PS/2 machine will get the AIX PS/2 version of **vi** when the editor is executed. Furthermore, as the number of cluster users increases or the editing load increases, the system can be easily expanded to handle the additional load by adding additional AIX PS/2 machines.

6.6.5 File System Backups

One significant I/O intensive task is file system backups. Unless care is taken, these backups can significantly consume the impacted cluster sites and/or the LAN network.
The following information offers several suggestions to consider when preparing backup strategies.

Whenever feasible, backups should be done with both the input an output device local to the process doing the backup. This will not generate cluster communication traffic and will permit the backup process to use character special devices which requires the least overhead for the system.

Whenever possible, backups, especially full backups, should be done a non-prime hours.

Full backups, though important, can be more expensive (in terms o media and time) than incremental backups.

If a backup must involve more than one cluster site, then care shoul be taken regarding the manner in which the backup utility is executed. Backups should only be run in such a way as to create the smallest amount of network traffic.

AIX does require that the process accessing an unbuffered characte special file (for example, the character device of the disk and tape) be on the cluster site with the real device. For instance, if it is desired to copy a disk on one cluster site (site **eyore**) to a tape drive on another cluster site (site **pooh**), then it is not possible for a single process to access both character special files. A simple solution is given below (the command should be entered on site **pooh**):

```
onsite <site pooh> dd if=disk | dd of=tape
```

In this example, a pipe is used to connect the reading and writing process. In this way the character special files for both the tape and the disk can be used. This is especially important for the tape as the character special file is the only way large records can be written on a tape. Some care has also been used with respect to the location where the pipe was created. It is more efficient for the pipe to be created at the site of the writer rather than the site of the reader, and by issuing the command line from Site B, the pipe will be created on Site B.

6.7 Data Organization

The overall performance of the system is impacted by the performance provided by the file system. When the file system throughput is unable to respond to the needs of the users, then it becomes a bottleneck. In this section key issues that impact file system performance are considered.

Subtopics
6.7.1 Replicated and Non-replicated File Systems
6.7.2 Large and Small File Systems
6.7.3 Large Directories
6.7.4 Long Pathnames and Symbolic Links

6.7.1 Replicated and Non-replicated File Systems

There are two types of file systems in AIX: replicated and non-replicated file systems. A replicated file system has at least one copy and may have more than one. Only one copy, the primary copy, will support user level modification of the data. The other copies are read-only and can only support user read requests. A copy of a replicated file system can be a primary, backbone or secondary copy and each copy is maintained by a different cluster site. A non-replicated file system has exactly one copy which exists on one cluster site.

Replicated and non-replicated file systems are equally expensive to access for read. However, opening a file for read in a replicated file system may be slightly more expensive than opening a file for read in a non-replicated file system. The reason for this is because all open operations must inform the current synchronized site that a specific file is being opened. In the case of a file in a non-replicated file system, the CSS is guaranteed to be the cluster site with the file system. This may not be true for a replicated file system, as the CSS is usually the cluster site with the primary copy of the replicated file system. Though the open operation might be slightly more expensive, reads are probably cheaper for replicated file systems, as the presence of multiple copies of the data increases the probability that a copy of the data will be locally available to the reading process.

A close operation (after the file was only read) is more expensive for a replicated file system than a non-replicated file system. When a file is closed, the system must update the access time of the file. This involves accessing the inode for the file and updating the access time and then writing the inode back to disk. For a non-replicated file system, these updates are batched and not done at the time the file is closed. For replicated file systems, no batching is done; the update of the access time is done as part of the close operation (however, the write I/O operation which updates the disk inode is a batched write) and in addition, the CSS is informed that the file is being closed.

Like a read operation, a write operation is equally expensive for both non-replicated and replicated file systems. There is one exception which is the result of synchronization overhead. When multiple processes on different machines are reading and writing the same file, additional overhead may be incurred to provide the single machine UNIX system synchronization model. A sample situation is:

> Both Site A and Site B have copies of a replicated file and Site maintains the primary copy.
>
> A process on Site B is reading the replicated file
>
> A process on Site A is writing the replicated file

UNIX operating system semantics require that the reader see the latest changes the writer has made. However, the system attempts to use local copies of data whenever possible so the process on Site B will be reading the local copy of the file. When the process on Site A opens the file for writing, the reader on Site B is temporarily blocked. This additional synchronization overhead may be incurred on replicated file systems.

A close or commit operation (after the data was updated) on a replicated file system is more expensive than a close or commit operation on a non-replicated file system. A close operation implies a commit operation

if the file has been updated. In the case of a non-replicated file system, internal to the system, the commit/close operation is usually batched and delayed so that a process may access the same file without incurring additional overhead. No batching or delays are done for a file system that is very low in free space or is a replicated file system. In addition, the system must notify all currently available copies of the updated file that a new version of the file is available. The kernel at those cluster sites will propagate the changed version to their local copy.

Replicated file systems can offer a performance advantage when the files in the file system are accessed for read more than for write operations. This advantage is the direct result of the increased probability that a local copy of the data will be available. In addition, the data is available from more than one cluster site, thereby decreasing the traffic to any one site because the traffic is split between more than one cluster site. All aspects must be considered before a decision is made but, in the very least, performance is improved with the use of replication.

6.7.2 Large and Small File Systems

AIX does not put significant limits on the size of any particular file system. As in most UNIX operating systems, the file system performance is impacted by the size of the file systems. When large file systems are allocated, the disk arm must move a relatively large distance from the disk inode of a file to the data blocks of a file. This increases the time required to open and read a file. Smaller file systems do not have this problem; however, they do have the problem that they may fill up unevenly since free blocks from one file system cannot be used by another file system. It is recommended that the administrator attempt to keep file systems relatively small and have several rather than a single large file system. Smaller file systems also make it easier to do physical reorganization.

When file systems become very full or are very active (updated extensively), the list of free file system blocks can become very fragmented. A fragmented free list means that successive blocks of the free list (which will become successive blocks of a file) are not physically close to each other on the disk. Again, this results in a relatively long time to access the data. The administrator should attempt to keep at least 15% free space in all file systems. In addition, after a significant amount of modifications, all file systems should be physically reorganized. This can be accomplished by using the **backup/restore**, **tar**, or **cpio** commands. For more information on these commands, refer to the *AIX Operating System Commands Reference*.

6.7.3 Large Directories

A problem similar to the "full file" system issue involves large directories. Large directories are expensive to search and expensive to update. In addition, directories are never shrunk, except when they are deleted. Care should be used to make sure that frequently searched directories are kept relatively small. One example would be **/tmp**. The simplest way to make a big directory with a significant number of deleted entries smaller is to create a new directory. The next step would be to move all the existing files from the big directory to the new directory. When moving files, remember those files whose names begin with . (dot). Then rename the two directories. If the large directory is the root directory of a file system, the only option is to back up the file system and use the **mkfs** command to create a new file system.

6.7.4 Long Pathnames and Symbolic Links

AIX, like other UNIX systems, provides a hierarchical file name structure. More specifically, a directory can contain files or other directories. Typically directories are used to structure data in some logical organization. In AIX, as in other UNIX systems, long pathnames can increase the overhead in translating the character string name into a physical object. This problem can be further aggravated by symbolic links. Symbolic links permit a user or programmer to put a pointer into the file system that references another directory or file somewhere else in the hierarchical file name tree. The pointer is actually a file name itself. For more information on symbolic links, refer to the introductory material in the *AIX Operating System Technical Reference*.

If a pathname has a symbolic link as one of its components, then, in terms of performance, the pathname has implicitly been lengthened by the length of the filename in the symbolic link. In fact, there may be several symbolic links referenced by a given pathname. When the total number of directories and symbolic links to be examined to translate a character string pathname becomes very large, the system performance can be degraded. Symbolic links or long pathnames should be avoided.
Long pathnames can be avoided by using the current directory mechanism, the **cd** command. For information on the **cd** command, refer to the *AIX Commands Reference*.

6.8 System Configuration

The AIX/370 and AIX PS/2 kernels require some parameter tuning like most UNIX operating system kernels. Typical constants to consider for tuning are:

AIX kernel: Generally the goal is to have the kernel be as small as possible but still be able to handle the offered load even during heavy load periods. At most, the kernel should not consume more that 70% of the available real memory (in the case of AIX/370, 70% of the virtual machine size).

Buffer cache: Maintaining a reasonable size buffer cache is also important and must be balanced with the size of the kernel.

Incore inode table size: This affects the delayed commit results in recently used incore inodes being cached until the next **sync** system call. As tables increase in size, the size of any corresponding hash structures should increase.

Number of network message buffers: These are relatively small headers that are used by the TCF protocol.

All of these constants have reasonable default values configured in the distributed system. An exception could be the buffer cache size, whose value is directly related to amount of physical memory available to the (virtual) machine. The default values are driven by the number of expected users for the system, so by setting this single constant, most of the other constants should have appropriate values.

With AIX/370, there are some additional considerations. In the VM environment, it is possible for the virtual machine to be paged out or written to disk at various times. This should be avoided since the performance implications are significantly detrimental in that the entire virtual machine is suspended while waiting for that page to become available. The physical machine should be configured to have enough real memory to support the AIX/370 virtual machines executing on the physical machine. Assigning the AIX/370 virtual machine to the **V=R** or **V=F** option improves performance. With this option set, NOTRANS should be enabled to allow the AIX/370 system to do its own address translation. STBYPASS (VM/SP only) should be set to VR so that standard page and segment tables are bypassed and VM is prevented from paging the virtual memory of the AIX/370 virtual machine.

6.9 Monitoring System Performance

There are several utilities that are available to a system administrator to monitor the operation of the system. The **lnetstat** utility was already mentioned on page 6.6.3. Others involve system accounting and system activity reporting. For more information on the utilities, refer to *Managing the AIX Operating System*.

For more information on monitoring the operation of the system, refer to the **ps**, **crash**, **df**, **rdf**, and **loads** commands in the *AIX Operating System Commands Reference*.

For those frequently executed user applications available in source form, the system administrator is encouraged to use user level profiling to determine if there are significant performance bottlenecks in those applications. Improving performance of frequently executed user applications can result in significant performance improvements.

The operations staff should also regularly examine system console output for any problem messages. An example would be messages indicating the system is continuously respawning the **getty** command due to improperly terminated or connected terminal lines. Another example would be messages indicating problems in the physical LAN media. Correct operation can often continue with both conditions but performance will be degraded.

6.10 Additional Considerations

Other performance issues involving availability, job control, and daemons are described in the following sections.

Subtopics
6.10.1 Availability Issues
6.10.2 Job Control
6.10.3 Daemons

6.10.1 Availability Issues

AIX will execute transparently any programs even if the program is not available on the disk of the execution site. This is just one important example of the transparency provided by AIX. However, it can lead to some unintended performance degradation. A program executed in this fashion may not be invoked as rapidly as it might if it were stored on the local disk. If the program is lightly used, then this is not an important concern and can be ignored. For heavily used system utilities and user applications, care should be used to guarantee that the application is locally available to the execution sites.

Several possible reasons that a program binary might not be available locally are:

A user application which is in a file system that is not replicated incorrectly replicated, or replicated on a subset of the appropriate sites. The program can either be properly replicated or correctly installed in the root file system with the correct **fstore** value.

The program is part of the root file system but has a 0 or incorrec **fstore** value. This can be corrected with **chfstore**. In addition, the system utility **chkfstore** can be used on the primary copy of the root file system to determine what files have an inappropriate **fstore** value.

The replicated file system with the program binary may not b completely propagated and current with respect to the primary copy of the file system. The **rdf** command can be used to verify that a non-primary copy of a replicated file system is up-to-date.

For more information on the **rdf**, **chfstore**, and **chkfstore** commands, refer to the *AIX Operating System Commands Reference*.

6.10.2 Job Control

The AIX C-shell provides job control support. If users learn to use it correctly, they can reduce system overhead in moving between a C-shell session and an editor session. Job control permits a user to request that a task be suspended and its parent be given control of the terminal. This mechanism is a significantly less expensive method to return to a shell than using the typical shell escape mechanism (":!" in **vi**). For more information on job control, refer to *Using the AIX Operating System*.

6.10.3 Daemons

One of the advantages of transparency is that programmers need not think about machine boundaries in writing programs. However, some algorithms have performance effects and should be recognized for them. This is especially important in daemons and heavily used applications. Daemons can easily start to consume more of the available cycles than desired if care is not used to monitor the cycles expended by the various system daemons. Often, any problem will become visible with careful inspection. Further examination can be performed with user level profiling.

7.0 Chapter 7. VM File Transfer

Subtopics

7.1 Contents

7.2 About This Chapter

VM File Transfer is a facility for sending and receiving files. This chapter describes the facility that permits an AIX/370 user to communicate with other AIX users as well with CMS and MVS/TSO users.

7.3 Introduction

VM File Transfer allows an AIX/370 user to send files to and receive files from other users with the **uvcp** and **vucp** commands. VM File Transfer uses the NETDATA protocol, which is supported by both CMS and MVS/TSO systems; therefore, users may be local or remote AIX/370 users, CMS users, or MVS/TSO users and can communicate through the RSCS program as shown in Figure 7-1. The files being received are normally converted into ASCII as they are read from your AIX/370 **netfile** directory and the files being sent are normally converted into EBCDIC. Whether a file is converted or not can be controlled with the same commands.

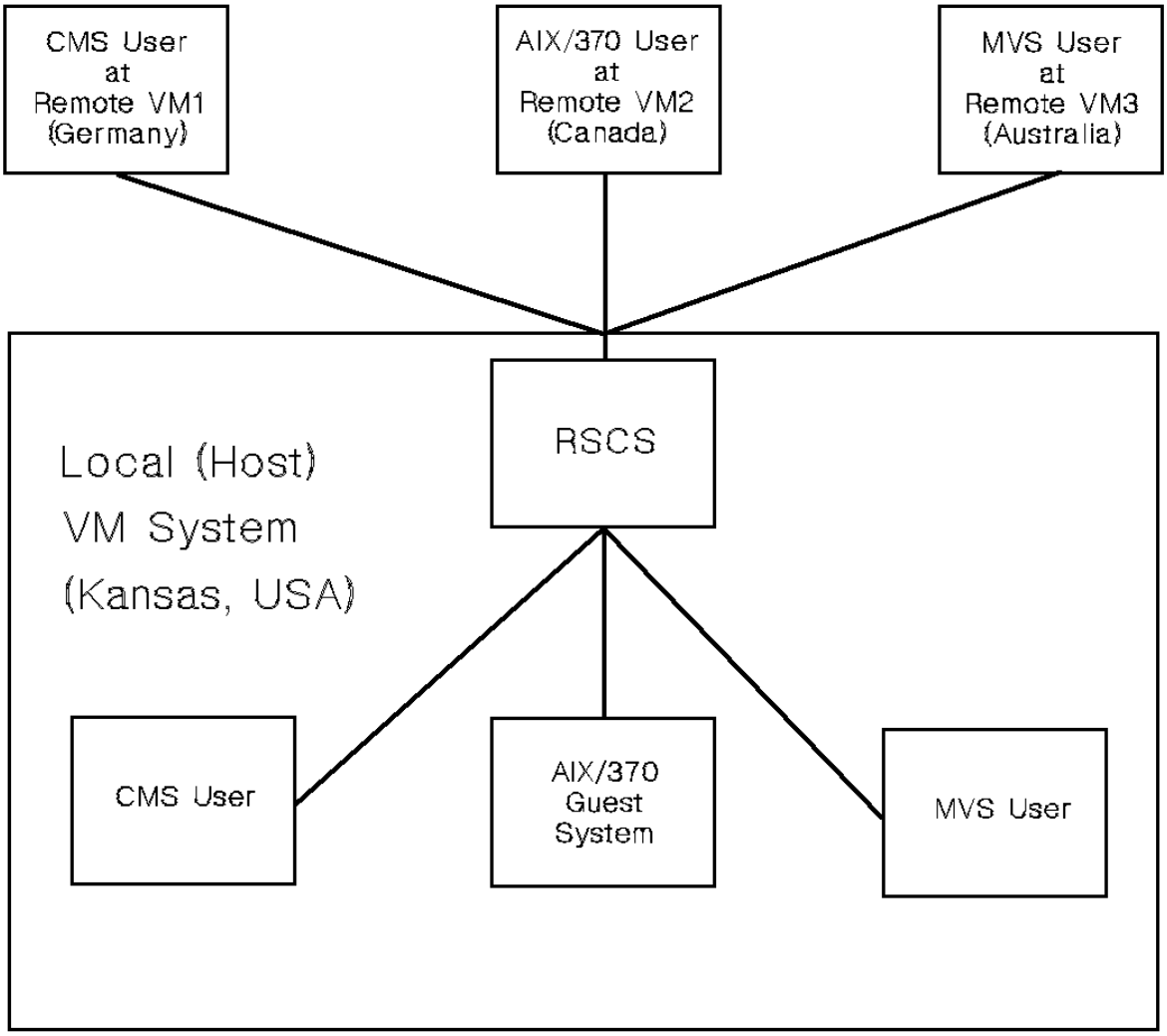

Figure 7-1. VM File Transfer

Figure 7-2 shows in detail the different ways of communication using **uvcp** and **vucp**. VM File Transfer allows file transfer between all subsystems on an AIX/370 guest, between any AIX/370 subsystem and any other AIX/370 guest on the same VM Control Program, or to remote systems.

VM File Transfer also allows communication between two users on a given subsystem. This transfer is similar to local mail transfer.

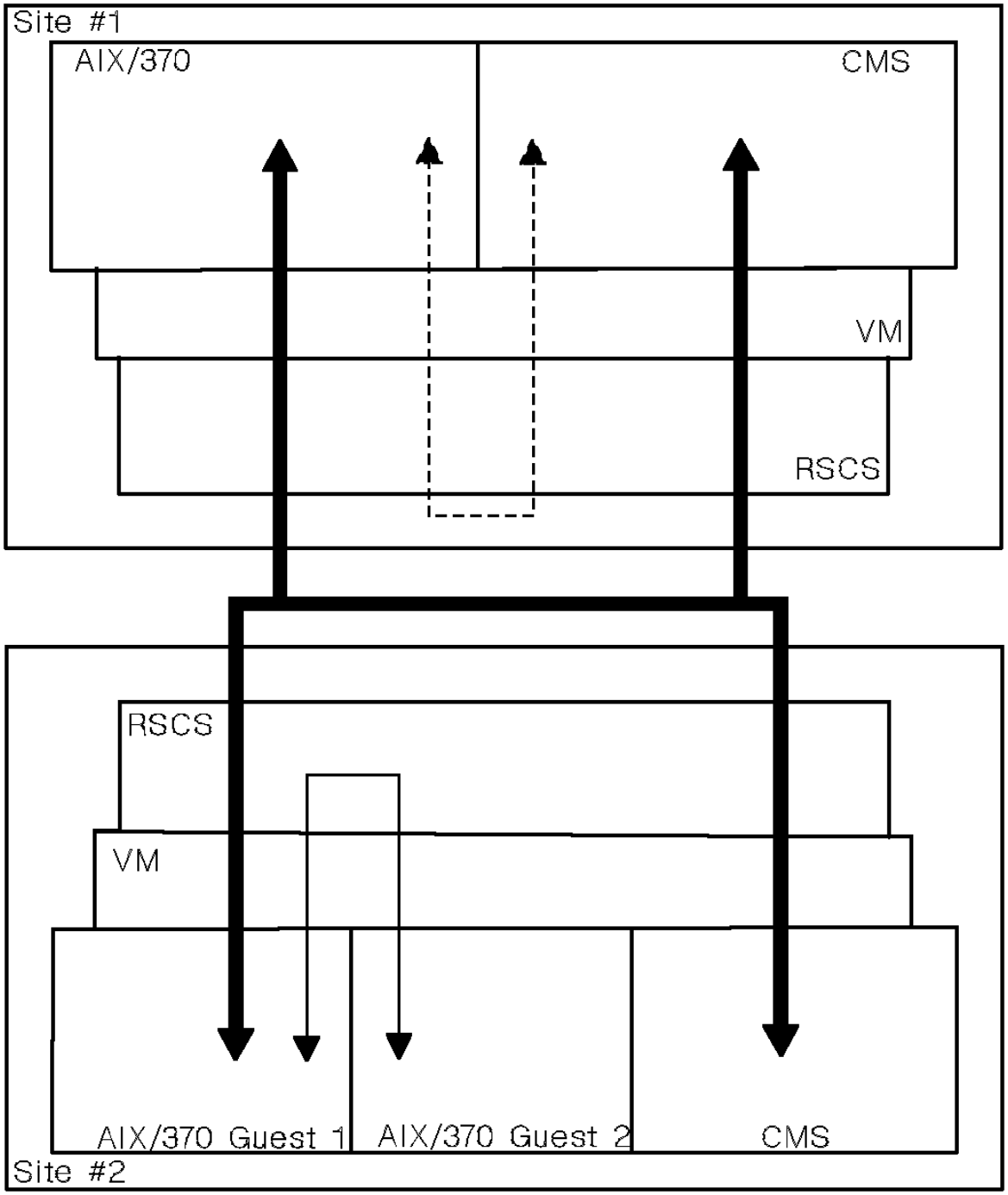

Figure 7-2. Communication Using VM File Transfer

The VM File Transfer software uses the VM CP spool file mechanism to transfer files, either between AIX/370 and CMS or MVS/TSO or between one AIX/370 user and another.

The files can be binary, ASCII, EBCDIC, or in a form that you have created. There is no restriction on the size of the file that may be sent or received by an AIX/370 user other than the size of the file system; however, the receiving AIX/370 file system must have sufficient space to read in the file.

The VM File Transfer facility handles the translation of AIX/370 files from ASCII to EBCDIC code and from EBCDIC to ASCII code. The operating system includes translate tables and also allows you to create tables

tailored to the requirements of the installation.

7.4 Program Requirements

If VM File Transfer is to be implemented at your installation, RSCS (Program Number 5748-XP1, Version 1 Release 3 or Program Number 5664-188, Version 2) is required. Also, to support the exchange of files between an IBM AIX/370 system and an IBM MVS/TSO system, the IBM MVS feature Bulk Data Transfer (BDT), feature number 5665-302, is required.

7.5 VM/SP Administrator Tasks

The system administrator is responsible for defining punch devices and reader devices in AIX/370.

Subtopics
7.5.1 Define VM Punches for AIX/370 Guest
7.5.2 Define VM Readers for AIX/370 Guest

7.5.1 Define VM Punches for AIX/370 Guest

One or more punch devices (PUNCH) must be defined for the AIX/370 guest machine. Define the punches in the VM/SP directory of the AIX/370 guest machine with the statement:

```
SPOOL cuu 2540 PUNCH
```

7.5.2 Define VM Readers for AIX/370 Guest

One or more reader devices must be defined for the AIX/370 guest machine. Define the readers in the VM/SP directory of the AIX/370 guest machine with the statement:

```
SPOOL cuu 2540 READER
```

7.6 AIX/370 Administrator Tasks

The AIX system administrator should verify that certain administrative tasks have been performed. For example, the Remote Spooling Communications Software (RSCS) line driver must be installed. The driver has to be installed after AIX/370 has been generated and configured for VM File Transfer. In addition, the AIX/370 name needs to be defined as a node identifier in the RSCS routing tables. Personnel responsible for RSCS have to make updates separately. Therefore if VM File Transfer is to be used, inform the RSCS administrator. Node identifiers in any RSCS network must be unique. Node identifiers need not be the name of the AIX/370 virtual machine system but generally will be.

The AIX/370 Administrator is responsible for

Configuring Virtual Punches and Reader

Insure that the following entry exists as a stanza in the **/etc/qconfig** file. In most cases, the administrator can uncomment this stanza since it may already exist.

```
pun:
  argname  = -pun
  node     = vm_node1, vm_node2 ...
  device   = pundev
  acctfile = /usr/adm/qacct
```

The names **vm_node1** and **vm_node2** represent those AIX systems running under VM.

Verify that virtual readers are configured into your AIX/370 kernel and that a special file for rdr exists.

Defining Software Device for CP Command

Verify that **/dev/CPcmd** is generated.

Customizing the **/etc/System.Netid** file

The VM File Transfer routines need specific information about AIX/370, VM/SP, and RSCS. The information must be defined in **/etc/System.Netid**. AIX/370 (as shipped) includes a sample file that you use for entering your definitions.

The AIX/370 operating system must have a **/etc/System.Netid** file. The system administrator has to define the file during the system generation process. In the shipped sample file, you define the:

- node ID of the local VM/SP system
  (to identify local CMS users).

- user ID of the AIX/370 guest machine

- node ID of the AIX/370 guest machine. Generally the user ID and node ID of the AIX/370 guest machine are the same.

- user ID of the RSCS virtual machine
  (for file transfer to/from remote users using the VM File Transfer function).

- reader spool class
  (for receiving files for the master/subsystem).

- reader HOLD class
  for files to be put on temporary HOLD because of conditions like the **netfile** directory not being available.

For more information on the **/etc/System.Netid** file, refer to the AIX Operating System Technical Reference.

Defining Translate Table

Define the NLS ASCII/EBCDIC translation tables in the environment variables NLIN (for EBCDIC to ASCII) or NLOUT (for ASCII to EBCDIC) pointing to the table in the file **/usr/lib/nls/nlin** or **/usr/lib/nls/nlout**. For more information, see "File Translation" in topic 7.10.

Note: Check periodically for CP spool files put in HOLD because they could not be delivered to an AIX/370 user. If the AIX user ID doesn't have Class D privileges, issue the command:

```
cp q rdr all hold
```

If the AIX user ID does have Class D privileges, issue the command:

```
CP Q /* RDR ALL HOLD
```

Then change the TAG and DISTCODE information or PURGE the spool file.

7.7 VM/SP RSCS Administrator Tasks

The system administrator responsible for VM/SP RSCS should implement the following:

1. Specify RSCS routing tables.
   Defining the identifiers and building the appropriate routing tables is the responsibility of the RSCS administrator. For information, refer to "Building Routing Tables" in topic 7.7.2.

2. Install the special RSCS line driver needed for VM File Transfer. The driver provides the interface to the RSCS network and is shipped as part of AIX/370. The code is stored in the AIX/370 root-file system under the directory **/usr/lib/RSCS**. After installing the line driver, the RSCS administrator has to update RSCS configuration files. The driver has to be installed after AIX/370 has been generated and configured for the VM File Transfer function. For information on installing the RSCS driver, refer to *Installing and Customizing the AIX/370 Operating System*. Some of the procedure steps must be done by the AIX/370 administrator, while others must be done by the VM/SP RSCS administrator.

Subtopics
7.7.1 Identifiers
7.7.2 Building Routing Tables
7.7.3 RSCS Routing Table Example

7.7.1 Identifiers

A unique identifier of a particular AIX/370 system is the individual system's node ID; the node ID is defined in the **/etc/System.Netid** file. Because RSCS controls the routing of files, RSCS node identifiers corresponding to the AIX/370 system must be defined in the RSCS virtual machine.

7.7.2 Building Routing Tables

The following should be considered when building the routing tables required by RSCS to handle AIX/370 file transfers (using the VM File Transfer function):

Node identifier
LINK statement
CP SPOOL file classe
ROUTE statement
START statements

Node Identifiers:
Unique node IDs to represent the AIX/370 system must be defined in the RSCS routing tables contained in the RSCS DIRECT file.

LINK Statements:
A separate LINK statement is used to define the path to the AIX/370 guest machine running on the local VM system. If more than one AIX/370 guest machine is running on the VM system, a separate LINK statement is required for each of these AIX/370 guest machines. The RSCS line driver DMTVIX is used to transfer files destined for the AIX/370 users and must be specified on this LINK statement.

CP SPOOL File Classes:
Different CP SPOOL file classes are used to separate the files targeted for the AIX/370 guest machine.

ROUTE Statements:
The paths to a remote AIX/370 guest machine are defined by using ROUTE statements.

START Statements:
A START statement may be included in the PROFILE RSCS file corresponding to every LINK statement in the RSCS DIRECT file to automatically activate communication on the LINK at RSCS startup. The RSCS START command parameters are used to specify:

CP SPOOL CLASS to be used for sending files to the AIX/370 syste represented by the node ID.

user ID of the AIX/370 guest machine

This provides the binding between the node ID and the AIX/370 system that it represents.

The details of constructing the PROFILE RSCS and RSCS DIRECT files depend on the RSCS version and are described in the following publications:

For RSCS Version 1 Release 3
IBM Virtual Machine/System Product: Remote Spooling Communications Subsystem Networking Program Reference and Operations

For RSCS Version 2 Release 1
IBM Virtual Machine/System Product: Remote Spooling Communications Subsystem Networking Program Reference and Operations.

These manuals also describe online commands to manipulate the RSCS routing tables.

The line driver DTMVIX uses CP TRANSFER commands to transmit files to AIX/370. The RSCS force and drain commands are supported for this line driver.

7.7.3 RSCS Routing Table Example

Note: This example applies to RSCS Version 1, Release 3.

Assume a VM system named SFCVM1 which runs an AIX/370 guest machine named AIX1. In RSCS, let the node AIX1 represent the AIX/370 system on SFCVM1. The RSCS DIRECT file for the RSCS virtual machine must have a LINK card to define a line driver and a path to AIX1. This card has the form:

```
LINK AIX1 DMTVIX nnn
```

where **nnn** specifies a dummy line address to AIX1. A dummy device should be defined at address **nnn**. This can be done with the following definition in the RSCS profile:

```
CP DEF CTCA nnn
```

This parameter is required and all line addresses within a RSCS system must be unique. A dummy device can also be defined in the RSCS entry in the CP directory. For more details, refer to the RSCS manuals noted earlier in this chapter.

A START statement must be included in the PROFILE RSCS file. This statement tells the RSCS supervisor to start communications for the node AIX1 with the AIX/370 guest machine AIX1, using the DMTVIX line driver at RSCS startup. The START statement has the form:

```
START AIX1 PARM A AIX1
```

The first parameter following **PARM** on the START command specifies the CP SPOOL CLASS to be used in sending the files for this node. In this example, **A** is the CP SPOOL CLASS. The second parameter following **PARM** specifies the user ID of this AIX/370 guest machine. In this example, **AIX1** is the user ID of the AIX/370 guest machine.

Now consider a remote VM system called SFCVM2. If SFCVM2 is directly connected to SFCVM1 by a binary synchronous communication line, then the RSCS DIRECT file for the RSCS virtual machine at SFCVM2 will have a LINK card defining the path to SFCVM1. This could be:

```
LINK SFCVM1 DMTVMB 057
```

A ROUTE card is required to define the path to AIX1 through SFCVM1:

```
ROUTE AIX1 SFCVM1
```

On the other hand, if the SFCVM2 is connected to SFCVM1 by means of an intermediate node called NYCVM1, then the path from SFCVM2 to SFCVM1 is defined by a ROUTE card:

```
ROUTE SFCVM1 NYCVM1
```

A ROUTE card is still required to define the path to AIX1:

```
ROUTE AIX1 NYCVM1
```

Furthermore, the RSCS system at NYCVM1 must have a LINK card to define the path to SFCVM1. This might have the form:

```
LINK SFCVM1 DMTVMB 043
```

It must also have the ROUTE card to define the path to AIX1 as follows:

```
  ROUTE AIX1 SFCVM1
```

7.8 VM File Transfer Operation

VM File Transfer uses the VM CP spool file mechanism to transfer the files between users. Figure 7-3 illustrates the various routes described in the following section.

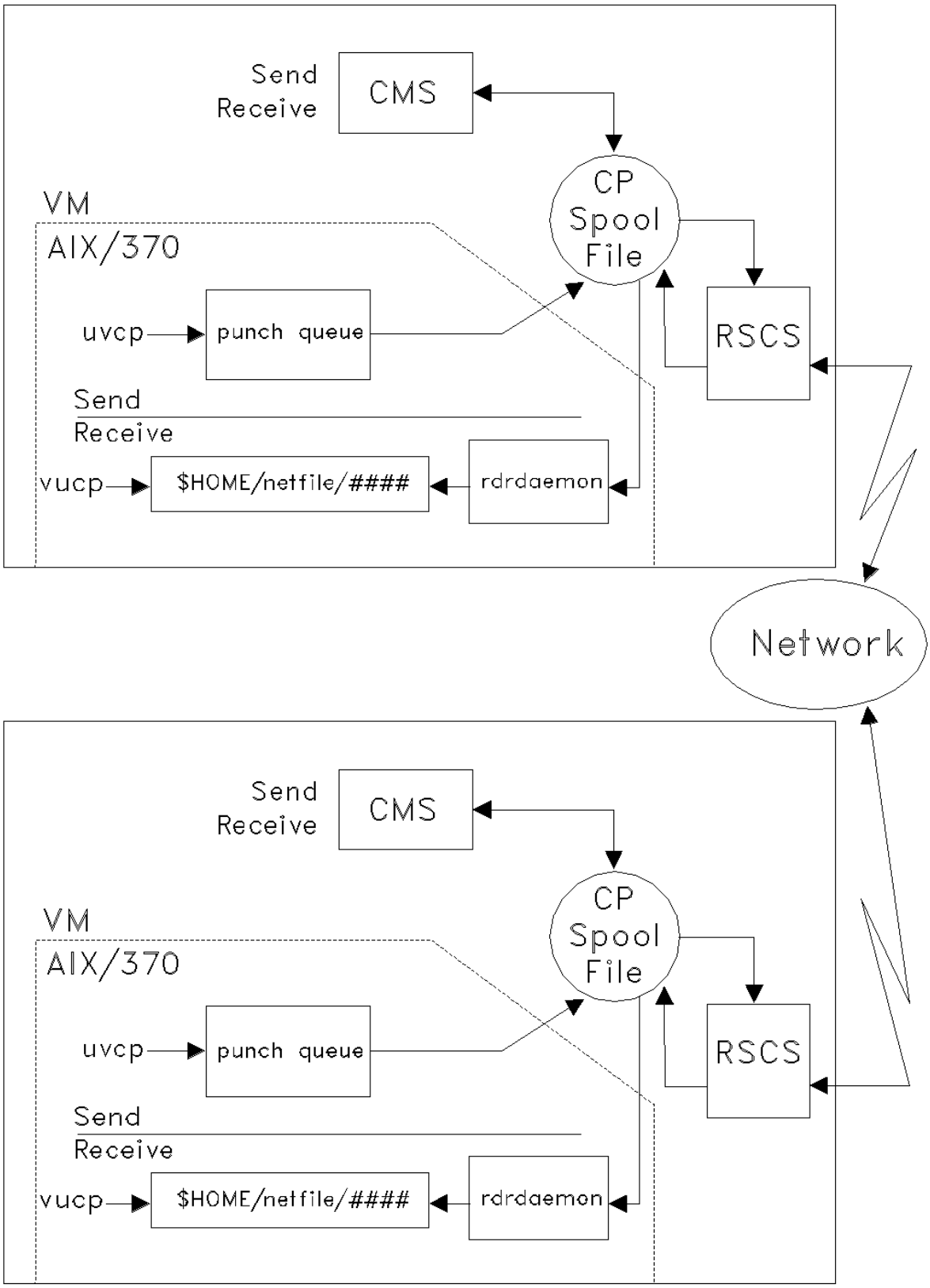

Figure 7-3. VM File Transfer Operation

To transfer a file:

to another user on the same AIX/370 system

Using the **uvcp** command, you can request a file to be sent to another

user. The file is translated (if necessary), changed into NETDATA format, sent to the **pun** queue, and then to the CP spool system. From there it is routed to the reader of that same AIX/370 system. The **rdrdaemon** process, when activated, reads the file into the correct user's **$HOME/netfile** directory using the spoolid as the file name.

to another AIX/370 user on another AIX/370 guest system on the same V system.

Using the **uvcp** command, you can request a file to be sent to another user. The file is translated (if necessary), changed into NETDATA format, sent to the **pun** queue, and then to the CP spool system. From there it is routed to the reader of the RSCS virtual machine, where the node ID is checked, and RSCS transfers it to the correct node. The **rdrdaemon** process, when activated, reads the file into the correct user's **$HOME/netfile** directory using the spoolid as the file name.

from a CMS user to an AIX/370 user

Using the SENDFILE command, you can request a file to be sent to an AIX/370 user. The file is sent to the CP spool file, then to RSCS where the node ID is checked to determined whether the node is on the same or on a different VM system. If the AIX/370 user is on a different VM system, the file is first sent to the standard RSCS line drivers. If the user is on the same VM system, the file is routed to the CP spool file, then to the correct node reader. The **rdrdaemon** process, when activated, reads the file into the correct user's **$HOME/netfile** directory using the spoolid as the file name.

To receive a file:

1. The **rdrdaemon** process notifies the user through the **mail** facility when a file is received out of the CP SPOOL file into the user's **netfile** directory.

2. Use the **ls -t netfile** command to determine the spoolid for the files in your **netfile** directory. Then use the **vucp** command to place the file from that directory to the directory where you want it stored. The file is changed from the NETDATA format and translated (if necessary) while being transferred from the **netfile** to the other directory.

3. If the file was sent as a note with the CMS **note** command, the **rdrdaemon** process transfers the file into the user's incoming mailbox. The note can then be read with the **mail** facility.

7.9 File Names for NETDATA

You should consider how file names are interpreted by CMS or TSO. AIX/370 allows both upper and lowercase characters in file names, while CMS and MVS/TSO only allow uppercase alphameric characters. Hierarchical file names are supported by MVS/TSO but not by (VM/SP) CMS.

File names may be changed when received by CMS or TSO. The **uvcp** command builds the NETDATA control records with the file name described as a two level hierarchy. The right most **.** (dot) is used as the hierarchy delimiter and the sender's login name is used as the highest hierarchical level. Because CMS and MVS/TSO require uppercase characters, the file name is converted into uppercase. When you use the **-n** (note) option of **uvcp**, the file is received by another AIX/370 user, a CMS user, or a MVS/TSO user as

username.note

where **username** is the login name of the sender.

CMS allows only the file name and file type as name descriptors for a file. Therefore, the CMS RECEIVE command constructs

the filetype from the lowest (right most) hierarchical component o the name in the NETDATA control file and

the file name from the next hierarchical component of the origina file name.

For example, if the file

/usr/lib/test

was transferred from an AIX/370 to

a VM system, the file is RECEIVEd wit

file name the sender's login name
file type TEST

a MVS/TSO system, the file is received with the data set name o TEST .

The CMS RECEIVE command also allows the CMS user to receive the file with a different file name and file type than those specified in the NETDATA control record.

The following table shows the file name conversions between AIX/370 and CMS.

Table 7-1. File Name Conversions between AIX/370 and CMS

VM File Transfer	**CMS**			**AIX/370**
	fn	**ft**	**fm**	**file name**
From -> To	**(8)**	**(8)**	**(2) char**	
CMS -> AIX/370	A	B	C	A.B
CMS -> AIX/370	TEST	SCRIPT	A	TEST.SCRIPT

CMS -> AIX/370	TESTFILE LISTING A	TESTFILE.LISTING
AIX/370 -> CMS	A B	a.b
AIX/370 -> CMS	B C	a.b.c
AIX/370 -> CMS	B C	A.B.C
AIX/370 -> CMS	TESTFILE H	testfile0010.h
AIX/370 -> CMS	loginname TESTFILE	testfile

7.10 File Translation

The **uvcp** and **vucp** commands translate files into ASCII or EBCDIC character data by using the **axeb** or **ebxa** command. The tables for EBCDIC to ASCII translation are stored in the **/usr/lib/nls/nlin** directory while the tables for ASCII to EBCDIC translation are stored in the **/usr/lib/nls/nlout** directory. Each directory contains tables for particular countries. To select one for your system, define the environment variables NLIN (for EBCDIC to ASCII) or NLOUT (for ASCII to EBCDIC) to point to the table for your national language. If NLIN or NLOUT is not specified, a default universal table is used. For information on creating translation tables, refer to the **genxlt** command in the *AIX Operating System Commands Reference*.

7.11 Overview of AIX/370 and CMS Commands

The following tables show the commands used for checking whether you have any files to read, sending a file, and receiving (placing it into your directory) with VM File Transfer. In the tables, the following abbreviations are used:

fn	file name
ft	file type
fm	file mode
dest	destination (user at node)

An empty box indicates that this function is not available.

Subtopics

7.11.1 Check Files to Be Read

Table 7-2. Commands for Reading Files

Function	**CMS Commands**	**AIX/370 Commands**
List reader files	CP Q RDR ALL	ls -t $HOME/netfile
Query files	CP Q FILES	

7.11.2 Send Functions

Table 7-3. Commands for Sending Files

Function	**CMS Commands**	**AIX/370 Commands**
Send a file	SF fn ft fm to dest	uvcp file -duser at node
Send a file with acknowledgement	SF fn ft fm to dest (Ack	uvcp -a file -duser at node
Send a file with no log entry	SF fn ft fm to dest (NOLog	
Send files specified in a file	SF fn ft fm to dest (File	
Send files and do not display nodes	SF fn ft fm to dest (NOType	
Send a file with nickname support	SF fn ft fm to nickname	
Send a note	SF fn ft fm to dest (NOTE	uvcp -n file -duser at node
Send a note to user at destination through the RSCS network		mail user at node.RSCS
Translate and send file		uvcp -txlt file -duser at node

7.11.3 Receive Functions

Table 7-4. Commands for Receiving or Viewing Files

Function	**CMS Commands**	**AIX/370 Commands**
Receive a file into your directory	RECEIVE spoolid	vucp spid .
Receive a file into your directory and replace	RECEIVE spoolid (rep	vucp -r spid .
Receive a file into your directory with a new name	RECEIVE spoolid fn ft fm	vucp spid new-filename
Receive a note into a notebook	RECEIVE spoolid (not fn	mail (1)
Peek (Read) a file	PEEK spoolid	vucp spid

(1) A file of type **note** is delivered by the **rdrdaemon** process to the AIX user's incoming mail and can be processed by the AIX **mail** command.

7.12 Sending Non-NETDATA Files from AIX/370

You can send non-NETDATA files through the VM spooling system. You might, for example, want to submit a job stream to a job entry subsystem such as JES3. It is necessary to supply the **tag** text associated with each file using the parameters of the **punbkend** command. For more information on the **punbkend** command, refer to the *AIX Operating System Commands Reference*. To send non-NETDATA files, complete the following steps: They can be concatenated with pipes.

1. Translate the file to EBCDIC using the **axeb** command which preserves new-line characters.

2. Remove the new-line characters and convert every line into a record of 80 characters padded with blanks by use of a filter which you must implement yourself.

3. Issue the command:

   ```
   print -pun -u user -n node file
   ```

   to route the job stream to its destination.

You will not be able to receive messages or print output; therefore, it is necessary that you ensure through appropriate JCL options that the print output is routed to a printer or saved in a data set at the executing site. You should avoid using default routing back to the AIX/370 user.

You will be able to receive punch output. This output is not in NETDATA format, so you have to apply the necessary conversion to the file stored in your **$HOME/netfile** directory.

7.13 Sending Files (uvcp)

The following examples show how you might want to use the **uvcp** command. Each example is for a given condition; if you send the same file to a different system (AIX/370 or CMS), the acknowledgement message (if requested) would look slightly different.

Send a binary file from an AIX user to a VM user

```
uvcp -b a.out -d lmixu
```

In this example, translation option **-b** has been specified for a binary file; the file **a.out** will not be translated. Therefore, user **lmixu** is assumed to be a local VM user on the same AIX/370 system. No acknowledgement is requested (default).

You receive a message that the file is being sent; the format is

```
Sending file a.out as a.out to lmixu at nycix1
on Fri Apr 18 12:08:22 1989
```

Note: When you are sending files to another VM user, the **-b** option eliminates the default translations.

Send a file to a CMS user at another node and request acknowledgement

```
uvcp -a test.tstfile -d smith at nycvm1
```

In this example, no translation option has been specified. Therefore, the file **test.tstfile** will be translated to EBCDIC using the translate table as specified in the NLOUT environment variable. The translated file will be sent to user **smith** at node **nycvm1**. Acknowledgement has been requested as specified by the **-a** option.

You will receive a message that the file is being sent; the format is

```
Sending file test.tstfile as test.tstfile to smith
at nycvm1 on Fri Apr 18 12:08:22 1989
```

User **smith** will receive this file with a

file name TEST
file type TSTFILE
file mode A1

When **smith** receives the file on his A disk, you will receive a message in the form of a file (which is placed into your **netfile** directory). The format of the message is

```
Ackn: file test.tstfile recv
by SMITH at NYCVM1 on Fri Apr 18 12:27:07 1989.
```

Send a file to more than one CMS user using your own translate table

```
uvcp -n test -d smith at nycvm1
             -d brown at lonvm1
```

Note: The two lines shown in the previous examples would be on a single line in the actual program.

The file **test** is taken from the current directory. This file is translated to EBCDIC and is then sent to both users (**smith** at node **nycvm1**, and **brown** at node **lonvm1**). No acknowledgement is requested, so none is received. The file is treated as a NOTE file (as specified by the **-n** option) at the target systems.

The two users **smith** and **brown** will receive this file with a file name of **USER** where **USER** is the appropriate login name, a file type of **NOTE**, and a file mode of **A1**.

If the users each have a NOTEBOOK, the file is entered there.

You will receive messages that the file is being sent to each user; the format is

```
Sending file test as user.note to smith at nycvm1 on
Fri Apr 18 12:08:22 1989
Sending file test as user.note to brown at lonvm1 on
Fri Apr 18 12:08:25 1989
```

Send a file to a VM or MVS/TSO system user requesting a fixed recor length of 80.

```
uvcp -a -f80 test.asm -d smith at nycmvs
```

Acknowledgement has been requested as specified by the **-a** option.

When **smith** places the file into his directory, you will receive a message in the form of a file (which is placed into your **netfile** directory). The format of the message is

```
Ackn: file test.asm recv
by SMITH at NYCVM1 on Fri Apr 18 12:27:07 1989.
```

If the receiving system is a VM or MVS/TSO system, it formats the file as a fixed format file with a record length of 80.

The user **SMITH** receives this file with a file name of **TEST**, a file type of **ASM**, a file mode of **A1**, format of fixed (f), and record length (recl) of 80.

Send a note to a CMS user on your local VM system

```
uvcp -n -d jones at localvm
```

In this example, an input file has not been specified, so standard input will be used. The ID of your VM system is **localvm**. When you have finished the note, press **<Ctrl-d>** to signal the end of the file. The file will be translated to EBCDIC and sent to user **jones** on **localvm** as a Note file. When you press **<Ctrl-d>**, you receive a message stating that the file is being sent; the message form is

```
Sending file /usr/tmp/aaaa02116 as user.note to jones
at localvm on Fri Apr 18 14:00:11 1989
```

The user **jones** will receive this file with a file name of **USER** where **USER** is your login name, a file type of **NOTE**, and a file mode of **A1**. The file is added to a NOTEBOOK owned by **jones**.

7.14 Receiving Files (vucp)

The **rdrdaemon** process notifies you about any new files received into your **$HOME/netfile** directory. Note the spoolid which is the name of the file in your **netfile** directory, or issue the command:

```
ls -t netfile
```

This will provide you with a list of files (spoolids) such as:

```
1379
3656
```

The following examples show how you might want to use the **vucp** command to read in these files.

1. To find out who sent a file (with a spoolid of 1379), from where, when, and what type of file, issue the command:

   ```
   vucp -q 1379
   ```

   The following type of message will be displayed at the standard output. The file will not be placed into a different directory, so an acknowledgement if requested will not be sent at this time.

   ```
   File recv from USERA at NYCIX1 sent as auxrout
   at Thu Apr 17 13:43:18 1989, text file.
   ```

   This message identifies the sender, the time sent, and the type of file sent. The type of file may be text or binary. A text file means that the file was sent in EBCDIC. A binary file means that the file was sent without translation to EBCDIC.

   Note: A file or note sent from a CMS or MVS/TSO user does not have a file type indicated.

2. To look at a file (with a spoolid of 1379) without placing it in your current directory, issue the command:

   ```
   vucp 1379
   ```

   The contents of the file is displayed followed by an informative message such as:

   ```
   File received from USERA at NYCIX1 sent as auxrout at
   Thu Apr 17 13:43:18 1989.
   ```

3. To put a file into your current directory, issue the command:

   ```
   vucp 1379 .
   ```

   Acknowledgement (if requested) will be sent at this time and the following type of message will be displayed at the standard output:

   ```
   File ./auxrout.c received from USERA at NYCIX1 sent
   as auxrout.c at Thu Apr 17 13:43:18 1989.
   ```

   Note: If this file was not sent using **uvcp**, or the option **-b** was not specified, the received file was translated using the default translate table.

If the file was sent using **uvcp**, the received file may or may not be translated. This depends on the translation option identified in the file information message. If the **-b** option of **uvcp** was used, the file is not translated when put into the current directory.

4. To place a file in the current directory without translation, issue the command:

```
vucp -b 1379 .
```

The file **1379** is processed as a binary file (no translation). No new-line characters are generated when end of record conditions are found during the processing of the input file.
The output file is placed in your current directory. The name is taken from the input file name, unless that file name already exists. In that case a message would appear telling you that the file name already exists, and suggesting to you to use the **-r** option. Acknowledgement (if requested) will be sent at this time and the following type of message will be displayed.

```
File ./auxrout received from USERA at NYCIX1 sent
as auxrout at Thu  Apr 17 13:43:18 1989.
```

Note: If you use the **-b** option with a file that should be translated, the file is placed in the current directory. However, when you try to use the file, you will receive an error message. You should place the file into your current directory again without using the **-b** option (if you have not deleted it from your **netfile**).

5. To place a file not sent by the **uvcp** command (with a spoolid of 1379) into the current directory and assign a new file name to the file, issue the command:

```
vucp 1379 test.tstfile
```

The file is translated into ASCII by using the translate table. The translated file is placed in your current directory with file name of **test.tstfile** if that file does not already exist. If the file already exists, you will receive a message stating so and to use the **-r** option.

6. To place a file into the current directory using the NLIN translate table, and replace an existing file with the same name if it exists, issue the command:

```
vucp -r netfile/1379 .
```

In this example no explicit translation option is specified; therefore, the file is translated into ASCII by using the translate table in the NLIN environment variable. The processed file is placed in your current directory with the file name specified in the NETDATA record even if a file with that name already exists. The old file is purged and replaced because the **-r** option is specified.
Acknowledgement (if requested) will be sent at this time and the following type of message will be displayed.

```
File ./auxrout received from USERA at NYCIX1 sent as
auxrout at Thu Apr 17 13:43:18 1989.
```

7. To translate a file into ASCII, store it in the current directory, and have the information message added to the netlog file, issue the command:

```
vucp 1379 . >>netlog
```

In this example, the file **1379** is

translated (if necessary),
placed in your current directory with the file name generated from the information in the NETDATA record, and
the **vucp** information message is appended to the end of the netlog file.

The format of the information message is the same as the one you would normally see displayed; an example is

```
File ./auxrout received from USERA at NYCIX1 sent as
auxrout at Thu Apr 17 13:43:18 1989.
```

No message is displayed.

Acknowledgement (if requested) will be sent at this time.

7.15 Sending CMS Files to AIX/370

PROFS users cannot send notes to an AIX/370 user because their note are not in NETDATA format.

The CMS user can define nicknames for AIX/370 users

TSO user can use XMIT to send files

Subtopics
7.15.1 Using the NOTE command
7.15.2 Using the SENDFILE command

7.15.1 Using the NOTE command

A CMS user creates and sends a note using the NOTE command. The AIX/370 **rdrdaemon** process delivers the note to the user's incoming mailbox. The AIX user can then receive the note with the **mail** command.

7.15.2 Using the SENDFILE command

A CMS user can send a file to an AIX/370 user with the SENDFILE command. The AIX/370 **rdrdaemon** process notifies the AIX user through the **mail** facility that a new file is received and stored in the user's **netfile** directory. To place the file (with a spoolid of 3344) into the AIX user's current directory, issue the command:

```
vucp 3344 .
```

A message similar to the following is displayed at the AIX user's terminal:

```
File ./filename.filetype received from USER at NODE
sent as filename.filetype Fri Apr 18 13:15:16 1989.
```

7.16 Receiving Files from AIX/370

The CMS user may use the RECEIVE and PEEK commands on the files sen by the AIX/370 user. Acknowledgement is sent to the AIX/370 user if requested.

PROFS users can receive files from an AIX/370 user

MVS/TSO users can use the RECEIVE command to receive files

7.17 Sending Mail to CMS Users

The AIX/370 user can send mail (or in CMS terms can send a NOTE) to a CMS user in the following ways:

1. Issue the command:

   ```
   uvcp -n file.note -d user at nodeid
   ```

   This command sends the file **file.note** as a NOTE file to the CMS user at the VM node **nodeid**. The file can be received with the CMS command RECEIVE and will be added to one of the CMS user's NOTEBOOKS. PROFS users can also receive the file by opening the mail.

2. Issue the command:

   ```
   mail user @ nodeid.RSCS
   ```

   This command invokes the AIX mail facility and calls **rscsmail** to deliver the mail with the RSCS spooling system.

A.0 Appendix A. VM File Transfer Messages

The following messages may be seen when using the functions and commands of VM File Transfer. In this appendix, the word **cmdname** is used to represent the specific command that may appear when you receive a message. Otherwise, the actual command name is used just as you will see it. There are other messages which are associated with VM File Transfer but are only seen at the AIX/370 system console. They are explained in the *AIX Operating System Messages Reference*.

UVC001 cmdname: INVALID OPTION option

Explanation: You have entered a command which does not support the indicated option. This message is followed by a usage message indicating the correct form of the command.

System Action: Command processing is terminated.

User Response: Re-enter the command in a correct form.

UVC002 uvcp: INVALID DESTINATION

Explanation: You have entered a **uvcp** command with an invalid destination specification. This message is followed by a usage message showing the correct form of the destination parameter.

System Action: Command processing is terminated.

User Response: Re-enter the command with a correct destination parameter.

UVC004 uvcp: INVALID RECORD LENGTH SPECIFICATION

Explanation: You have entered a **uvcp** command with an invalid record length specification. This message is followed by a usage message showing the correct form of the record length parameter.

System Action: Command processing is terminated.

User Response: Re-enter the command with a correct record length specification.

UVC005 cmdname: EXTRA ARGUMENT argument

Explanation: You have entered a command with too many arguments. This message is normally followed by a usage message indicating the correct syntax of the command.

System Action: Command processing is terminated.

User Response: Re-enter the command in a correct form.

UVC104 cmdname: UNKNOWN VM USERID userid

Explanation: The **punbkend** could not spool a punch file to the indicated login name, because the login name was rejected by CP. If the login name was used for the RSCS virtual machine, this message is followed by message UVC105.

System Action: Processing for the current punch file is terminated.

User Response: If message UVC105 follows, see description for that message. Otherwise, re-enter the **uvcp** or **print** command with a correct local login name.

UVC105 cmdname: ERROR IN CONFIGURATION FILE /etc/System.Netid

Explanation: The file **/etc/System.Netid** does not exist, or has less than five fields, or the spool class field is not alphanumeric, or the RSCS login name is invalid (see also UVC104).

System Action: Processing for current command or spool file is terminated.

User Response: Have the System Administrator correct the **/etc/System.Netid** file.

UVC106 cmdname: ERROR IN ASCII/EBCDIC TRANSLATION

Explanation: A character in the input file could not be translated using the NLIN translate table.

System Action: Processing for the current command is terminated.

User Response: Verify or correct the format of the translate table.

UVC107 vucp: INVALID INPUT FORMAT

Explanation: The file specified as input to a **vucp** command does not have the expected NETDATA format. This error may also occur for a file stored into the **$HOME/netfile** directory by the rdrdaemon.

System Action: Processing for the current command is terminated.

User Response: For a file in the **$HOME/netfile** directory, determine the origin of the file from the header line, and use another method for processing/interpreting the file. Although AIX/370 does not provide commands to process non-NETDATA control records, the **xlate** command may be helpful as a processing step for your own defined interpreter.

UVC201 vucp: FILE filename ALREADY EXISTS, SPECIFY -r OPTION

Explanation: The file specified as output for a **vucp** command already exists, and is by default not overwritten.

System Action: Processing for the current command is terminated.

User Response: If the file should be overwritten, use the **-r** option, or specify another output file name.

UVC202 uvcp: FILE filename IS A DIRECTORY OR A SPECIAL FILE

Explanation: The file was specified as input to a **uvcp** command, and is not a normal file suitable for transmission.

System Action: Processing for the specified file is terminated. Processing for any other files specified on the same command is not affected.

User Response: Resubmit the command with a valid input file.

UVC204 cmdname: CANNOT ACCESS file

Explanation: An attempt to open the file (by a background task or while processing a command) failed for one of the following reasons:

The file does not exist
The requested access was denied
The access is physically inhibited

System Action: Any processing, which depends on the file access, is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC205 cmdname: CANNOT CREATE file

Explanation: An attempt to create the file (by a background task or while processing a command) failed for one of the following reasons:

The file already exists and write permission was denied
Write access is inhibited
A system limit is being exceeded

System Action: Any processing, which depends on the file access, is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC207 cmdname: READ ERROR errno ON file

Explanation: An error corresponding to **errno** occurred while a background task or a command process was reading the file.

System Action: Any processing, which depends on the file access, is terminated.

User Response: None specifically related to VM File Transfer.

UVC208 cmdname: WRITE ERROR errno ON file

Explanation: An error corresponding to **errno** occurred while a background task or a command process was writing to the file.

System Action: Any processing, which depends on the file access, is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC209 cmdname: IOCTL ERROR ON device

Explanation: An ioctl request for the device by the **rdrdaemon** or the **punbkend** failed. This is a system error.

System Action: Processing of the current spool file is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC210 cmdname: ERROR ON CP command

Explanation: The CP command issued by the **rdrdaemon** or the **punbkend** failed. This is a system error.

System Action: Processing of the current spool file is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC301 cmdname: OUT OF STORAGE

Explanation: A malloc call by a background task or while processing a user command failed.

System Action: Any processing which depends on the requested work storage is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC302 uvcp: CANNOT OPEN TEMPORARY FILE

Explanation: A temporary file required for processing a **uvcp** command could not be allocated.

System Action: Processing for the current file is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

UVC303 cmdname: CANNOT FORK

Explanation: An fork request during **rdrdaemon** initialization failed. The system limit for the total number of processes is being exceeded.

System Action: The **rdrdaemon** is terminated.

User Response: None specifically related to VM File Transfer. See your System Administrator for assistance if necessary.

GLOSSARY Glossary

AADU. See **AIX Access for DOS Users**.

access. To obtain computing services.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The login directory and other information that give a user access to the system.

activity manager. A collection of system programs allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

administrative event. A specified group of base events for which a user can be audited.

AIX Access for DOS Users (AADU). A program that allows you to bridge the gap between PC and UNIX hosts.

AIX cluster. A group of computers operating under the AIX Operating System with the Transparent Computing Facility (TCF).

All Points Addressable (APA) display. A display that allows each pixel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with **character display**.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters A through Z.

alphanumeric character. Consisting of letters, numbers, and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

APAR. See **Authorized Program Analysis Report**.

application. A program or group of programs that directly apply to a particular user problem, such as inventory control, word processing, or accounts receivable.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that affect the way a command works.

ASCII. See **American National Standard Code for Information Interchange**.

assembler. A computer program that converts assembly language instructions into object code.

asynchronous transmission. In data communication, a method of transmission in which the bits included in a character or block of characters occur during a specific time interval. However, the start of each character or block of characters can occur at any time during this interval. Contrast with **synchronous transmission**.

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

authorize. To grant to a user the right to communicate with, or make use of, a computer system or display station.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current unaltered release of a program.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

AUTOLOG. Otherwise called "automatic logon", this feature is a process by which a user's virtual machine (VM) is initiated by someone other than the user of that VM. For example, the primary system user's VM is activated automatically during initialization. A user can issue the AUTOLOG command to activate some other (disconnected) virtual machine.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator

intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

BAL. Basic Assembler Language.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A filename specified without its parent directories.

Basic Networking Utility. A program that enables users to communicate with computers other than their local computers.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background process. Contrast with **interactive processing**.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. Eight bits make a byte. See also **byte**.

block. (1) A group of records that is recorded or processed as a unit. Same as **physical record**. (2) In data communication, a group of records that is recorded, processed, or sent as a unit. (3) A physical block in AIX is 4096 bytes long. (4) A logical block in AIX/370 and AIX PS/2 is 1024 bytes. (5) A logical block in AIX/RT is 512 bytes.

block file. A file listing the usage of blocks on a disk.

BNU. See **Basic Networking Utility**.

block special file. A special file that provides access to an input or output device capable of supporting a file system. See also **character special file**.

boot. To prepare a computer system for operation by loading an operating system.

bootstrap. A small program that loads larger programs during system initialization. Sometimes referred to as IPL (Initial Program Load).

Bourne shell. A flexible command language that can be customized to specific applications or user needs.

bpi. bits per inch.

bps. bits per second.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

BSD. Berkeley Software Distribution.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. To activate a program or procedure at its entry point. Compare with **load**.

callouts. An AIX kernel parameter establishing the maximum number of

scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

CCW. Channel Command word.

central processing unit. The part of a computer that includes the circuits that control the interpretation and execution of instructions.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with **All Points Addressable display**.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with **function keys**.

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also **block special file**.

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System child is a **process** spawned by a parent process that shares the attributes of the parent process. Contrast with **parent**.

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

client. A computer or process that accesses the data, services, or resources of another computer or process on the network.

cluster. (1) Any configuration of workstations for the purpose of sharing resources (for example, Local Area Networks (LANs) and host attached workstations). (2) A group of storage locations allocated at one time.

CMS. Conversational Monitoring System.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to **program**. (3) A representation of a condition, such as an error code.

code segment. See **segment**.

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program (such as the Bourne or C shell) that sends instructions from the command line to the kernel.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command name. (1) The first or principal term in a command. A command

name does not include parameters, arguments, flags, or other operands. (2) The full name of a command when an abbreviated form is recognized by the computer (for example, print working directory for **pwd**).

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language. See **shell procedure**.

communication adapter. A hardware feature enabling a computer or device to become part of a data communication network.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communication, to delete a series of duplicate characters in a character string.

concatenate. (1) To link together. (2) To join two character strings.

concurrent group set. A list of the IDs of the various groups to which a user belongs.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also **system customization**.

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station for that site. (2) A device name associated with the main AIX display station at that site.

constant. A data item with a value that does not change. Contrast with **variable**.

context search. A search through a file for a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in shell procedures.

control program. Part of the AIX Operating System that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run, and saves any new data already created. The job that is running can continue with the next job step.

coredump. A kernel memory image dump that is given a unique name so that it will not be over-written in case of another failure. This enables the system administrator to analyze the dump and determine the cause of failure at some later time.

coupler. A device connecting a modem to a telephone network.

CPU. Central Processing Unit.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

CSS. See **current synchronized site**.

CTC. Channel-to-Channel.

current directory. The directory that is the starting point for relative pathnames.

current line. The line on which the cursor is located.

current synchronization site (CSS). The site in the cluster containing the primary copy of the replicated file system.

current working directory. See **current directory**.

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See **daemon process**.

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times to more than one task or user, such as sending data to a printer.

DASD. Direct Access Storage Device.

data block. See **block**.

data communication. The transmission of data between computers, and/or remote devices (usually over long distance).

data link. The equipment and rules (protocols) used for sending and receiving data.

data stream. All information (data and control information) transmitted over a data link.

dbm. Format for the Network File System network information service (NIS) data base files.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

dependent work station. A work station having little or no standalone

capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to your computer, for example, a printer, plotter, or disk drive.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device manager. Collection of routines that act as an intermediary between device drivers and virtual machines for complex interfaces. For example, supervisor calls from a virtual machine are examined by a device manager and are routed to the appropriate subordinate device drivers.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copied from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

distributed file system. A file system whose files, directories, and other components are stored on different sites in a particular cluster.

distributed operating system. An operating system where multiple machines cooperate to seem like one machine.

distributed processing. Results when a user involves multiple cluster sites in a single operation--for example, by editing a remote file and starting a task on another cluster site using the **on**, **fast**, **fastsite**, and **migrate** commands.

Distributed Services (DS). A licensed program that allows you to share files with other AIX systems in a network. You can mount the file systems located on other AIX systems to created file trees that are independent of the file systems.

DOS. Disk Operating System.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See **extended binary-coded decimal interchange code**.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

effective ID. The ID, either group or user, that is used to run a process. It can be set by a program either to the real or saved ID.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

END OF FILE. The user-definable character used to indicate end-of-file. By default, END OF FILE is set to be Cntr-D. For more information about user-definable characters, see the **termio** file format entry in the *AIX Operating System Technical Reference*.

enter. To send information to the computer by pressing the **Enter** key.

entry. A single input operation on a work station.

enumerate. Returns values from a network information service data base in a specified order.

environment. The settings for shell variables and paths associated with each process. These variables can be modified later by the user.

EREP. Environmental Recording Edit and Print program.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

ESSL. Engineering and Scientific Subroutine Library.

Ethernet. A physical medium through which computers in the same or different clusters can communicate and share files.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping. Exit values are also called Return Codes.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See **first-in-first-out**.

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also **label**.

filename. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. In DOS a file specification consists of a drive specifier, a path name, and a file name.

file system. A collection of files and directories stored on logical and physical devices (such as disks) and logically organized in a hierarchical fashion.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

filter programs. Programs designed to accept information from input, process the data, and write the results to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

first level interrupt handler (FLIH). A routine that receives control of the system as a result of a hardware interrupt. One FLIH is assigned to

each of the six interrupt levels.

fixed disk. A flat, circular, non-removeable plate with a magnetized surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System are almost always preceded by a dash.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

FORTRAN. A programming language primarily used to express computer programs by arithmetic formulas and numeric computations.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See **free list**.

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling routine.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead.

generation. For some remote systems, the translation of configuration

information into machine language.

GFS. See **Global File System**.

Gid. See **group number**.

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. See **pattern-matching character**.

Global File System (GFS). The entire composite file system of the AIX cluster. It consists of the root file system of the primary site plus all the mounted file systems from secondary sites.

global file system (gfs) number. In AIX, every mounted file system is identified by the global file system number (gfs). In a normal file system, this number is hardcoded in the superblock of the physical device where that file system actually resides. In a remote NFS file system, however, there is no such hardcoding; an arbitrary assignment is made by the system administrator. The operating system distinguishes every file system from every other by its gfs number. Each mounted file system is assigned a particular machine to serve as its current synchronization site (CSS). If any file system should somehow be assigned two gfs numbers, it would be considered two different file systems by the operating system and might well be entrusted to two separate CSSs.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

hexadecimal. Pertaining to a system of numbers using base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

hierarchical tree structure. The organization of files on AIX, similar to tree-structured directories, with each file like a small branch of a larger branch that represents the file's parent directory. A directory can also be contained in another higher level directory, with the parent of all directories represented by the tree's root (***root*** or ***root directory***).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history. A C-shell mechanism that lists previously executed commands. These commands can be re-executed with the **!** command.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. The directory a user accesses when logged in. Synonym for **login directory**.

home site. The computer that stores the modifiable copy of a user's home directory. This is the cluster site with the primary copy of his home directory if it is replicated. A user typically logs in to the computer that is his home site.

I/O. See **input/output**.

IDAW. Indirect Addressing Word.

IEEE. Institute of Electrical and Electronics Engineers.

IF expressions. Expressions within a procedure, used to test for a

condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

INed. A full screen editor that also features windows.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See **initialize, bootstrap**.

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

inode. The internal structure for managing files in the system. Inodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of inodes is stored near the beginning of a file system.

i-number. A number specifying a particular inode on a file system.

inodetab. An AIX kernel parameter that establishes a table in memory for storing copies of inodes for all active files.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output file. A file opened for input and output use.

input-output device number. A value assigned to a device driver by the guest operating system or to the virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with **batch processing**.

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communication, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

INTERRUPT. The user-definable character used to kill a process running in the foreground. By default, INTERRUPT is the Del key. For more information about user-definable keys, see the **termio** file format entry in the *AIX Operating System Technical Reference*.

interrupt character. A key sequence (**Alt-Pause** on some systems) typed in to cancel a foreground process.

IPL. Initial Program Load.

JES. Job Entry Subsystem

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job control. A feature that lets the system accept your commands to stop and start processes (jobs) and move them between background and foreground. The commands **ps** and **jobs** report the status of jobs, (each of which is assigned a Process Identification Number or PID to show its process status), and the **kill** command can be used to stop them.

job number. A number assigned to a background process when it is started. The job number is displayed when the process is started and when the **jobs** command is invoked. It can also be used to kill the process.

job queue. A list, on disk, of jobs waiting to be processed by the

system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte (Kb). See **kilobyte**.

kernel. The memory-resident nucleus of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also **file name**. (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

LAN. Local Area Network.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or file is in the leftmost position. Contrast with **right-adjust**.

library. A collection of functions, subroutines, or other data.

licensed program product (LPP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an inode and one or more file names associated with it. Synonym for **UNIX link** or **hard link**.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communication line. Contrast with **remote**.

<LOCAL> alias. The <LOCAL> alias can translate into different strings on different cluster sites for different processes. When <LOCAL> is the first component of the destination name for a symbolic link, it is replaced with its alias string, normally /**machinename**.

local area network (LAN). A physical medium that allows computers in the same or different clusters to communicate and share files. Ethernet and Token-Ring are two examples of a LAN.

local cluster site. The site on a cluster that the user is logged in to.

The term **local** normally refers to a TCF cluster site.

locale. Each process operates in its own locale. A set of environment variables that determines the language for input and output as well as the character sets for data in files and on networks. These variables also determine the message catalog and collating sequence as well as time, monetary, and numeric conventions to be used.

<LOCAL> file system. The part of the root file system hierarchy comprising system directories and files (such as the /etc/motd "message of the day" file) defined uniquely on a particular computer in the cluster. These files are not replicated. The name of the <LOCAL> file system appears in response to the **site-l** command.

location transparency. Allows an object to change location without the user's or program's knowledge if that location is not part of the object's name. For example, **/u/joe/glossary** may have been a file on **eyore** last week, but it is a file on **pooh** this week. Joe does not need to know that the file was on either **eyore** or **pooh**. If, however, Joe wants to find out where the site is located, he may invoke the **where** command.

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log off. See **log out**.

log on. See **log in**.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

login directory. See **home directory**.

login ID. The ID set by the system for a user after login but before running any programs.

login shell. The program, or command interpreter, started for a user at log in.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

LPP. See **licensed program product**.

macro. A set of statements defining the name of, format of, and conditions for generating a sequence of assembler statements from a single source statement.

mailbox. An area designated for storage of mail messages directed to a specific system user.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. An AIX kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

MBCS. See **Multibyte Character Set**.

M-byte (Mb). Megabyte (1,048,576 bytes).

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See **storage**.

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An inode field that describes the type and state of the inode.

modem. See **modulator-demodulator**.

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communication line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See **load module**.

mount. To make a file system accessible.

mount point. Any directory which has a file system mounted to it.

mountab. An AIX kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

Multibyte Character Set (MBCS). A method of encoding a set of glyphs (written symbols) so that the coded set is large enough to encompass all written language. The MBCS system handles character sets which range in encoding size from one to four bytes.

multiprogramming. The processing of two or more programs at the same time on the same logical system.

multivolume file. A diskette file occupying more than one diskette.

multi-user environment. A computer system that provides terminals and keyboards for more than one user at the same time.

MVS. Multiple Virtual Storage.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the

nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of computers that can communicate with each other. A network can consist of several interconnected computers or one computer with a number of remote terminals connected to it. Any of a variety of communication media can be used, such as RS-232, Ethernet, Token-Ring, or PC Net.

Network File System (NFS). A licensed program that allows you to share files with other computers in one or more networks that have a variety of machine types and operating systems. You can mount file systems located on network servers and use remote files as if they were on your workstations by creating file trees that are independent of the file systems.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

node. An individual element of a full pathname. Nodes are separated by slashes (/).

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instructions, usually generated by a compiler from source code written in a higher level language. It consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

OCO. Object Code Only.

online. Being controlled directly by, or communicating directly with, the computer, or both.

open. To make a file available to a program for processing.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

overflow condition. A condition that occurs when part of the output of an operation exceeds the capacity of the intended storage unit.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space. The area on a fixed disk that temporarily stores instructions or data currently being run. See also **minidisk**.

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

PANIC. An error message generated by the kernel indicating that an error has occurred which is sufficiently severe to prohibit kernel recovery.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a command or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with **child**.

parent directory. The directory one level above the current directory.

partition. See **minidisk**.

Pascal. A high-level, general purpose programming language, related to ALGOL. Programs written in Pascal are block structured and consist of independent routines. They can run on different computers with little or no modification.

Pass-Through Virtual Machine (PVM). A VM program which allows the user to open a computing session on a different machine. It allows the user to "pass through" from one virtual machine to another.

password. A string of characters that, when entered along with a user identification, allows an operator to login to the system.

password security. A program product option that helps prevent the unauthorized use of a display station by checking the password entered by each operator at log in.

path name. The sequential list of directory name(s) that identify the location of a particular directory, and directory name(s) and file name that identify the location of a particular file in the file hierarchy. The path name is displayed in response to the **pwd** (print working directory) command. Each file has a full path name, beginning with / (the root directory) and ending with the file's name. The file's relative path name does not begin with /.

pattern-matching character. Special characters such as * or ? that can be used in search patterns. They are sometimes used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. Pattern-matching characters are also called wildcards.

PC. Personal Computer.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read,

write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages of file system checking and repair performed by the **fsck** command.

physical device. See **device**.

physical file. An indexed file containing data for which one or more alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as **block**. (2) A unit of data moved into or out of the computer.

PID. See **process ID**.

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. (1) A logical connection between physical blocks. (2) A link to something else. (3) An address.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (2) A part of the system unit or remote controller to which cables for display stations and printers are attached.

portmap service (portmapper). A daemaon process that matches RPC port numbers to RPC services provided by NFS servers to conduct remote services in the NFS.

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A shell facility for assigning values from the command line to variables in a program.

POSIX. Portable Operating System for Computer Environments.

preprocessor. (1) A functional unit that effects preparatory computation or organization. (2) A program that examines the source program for preprocessor statements which are then executed, resulting in the alteration of the source program.

primary copy. Each replicated file system has a copy designated as the **primary copy**, which is the copy that may be modified. It resides on the **primary site** and its purpose is to guarantee that file updates are kept consistent.

primary site. The cluster site that maintains the primary copy of a replicated file system.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communication facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See **shell procedure**.

process. A program now running. A **foreground** process executes as soon as you type in the command line and completes before returning the system prompt to accept your next command. You can start one or more **background processes** to run independently while you type in a separate command for

another process to run in the foreground.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

process transparency. The ability to execute and control tasks on any site in the cluster, regardless of where the user is logged in (to find out where that is, type the **site** command). The same system calls and commands are used, no matter where the process is located. For example, a remote job is stopped the same way that a local job is stopped.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A set of instructions for the computer to interpret and execute.

program temporary fix (PTF). A temporary solution or by-pass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communication line to another.

protocol. In data communication, the rules for transferring data.

protocol procedure. A process that implements a function for a device manager. For example, a virtual terminal manager may use a protocol procedure to interpret the meaning of keystrokes.

PR/SM. Process Resource/System Manager.

PVM. See **Pass-Through Virtual Machine**.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This

is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

radix. The positive integer by which the weight of the digit place is multiplied to obtain the weight of the digit place with the next higher weight; for example, in the decimal numeration table, the radix of each digit place is 10, in a binquinary code the radix of each fives position is 2.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

ratfor. Rational FORTRAN.

read-only. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

real storage. The main storage in a virtual storage machine.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an inode, a record of the total number of directory entries that refer to the inode.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communication line. Contrast with **local**.

remote cluster site. A site on the cluster that the user is not logged in to. The term **remote** normally refers to a TCF cluster site.

Remote Procedure Call (RPC). The interface used by NFS. A request for a service located on another computer in the network.

Remote Spooling Communications Subsystem (RSCS). The licensed program that transfers spool files, commands, and messages between VM users, remote stations, and remote and local batch systems through HASP-compatible telecommunications facilities.

replicated root file system. The part of the root file system hierarchy comprising system directories and files found under the root (/) and replicated on all sites in the cluster. Replicated root files are not specific to individual cluster sites. The replicated root file system is a file system with key common files and directories for basic system operation. Almost all system binaries, programs and libraries are in the replicated root file system. Other user and system file systems (like the local file system) are mounted on top of directories in the replicated root file system.

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right adjust. The process of aligning lines of text at the right margin or tab setting such that the rightmost character in the line or file is in the rightmost position.

right justify. See right align.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with **left-adjust**.

root. (1) Another name sometimes used for superuser. (2) The main file system to which others are appended.

root directory. The top level of a tree-structured directory system.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

RPC. See **Remote Procedure Call**.

RSCS. See **Remote Spooling Communications Subsystem**.

RTM. Real Time Monitor.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

SCCS. Source Code Control System.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See **display screen**.

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

second level interrupt handler (SLIH). A routine that handles the processing of an interrupt from a specific adapter. A SLIH is called by the first level interrupt handler associated with that interrupt level.

secondary copy. A read-only copy of the primary copy of a replicated file system. Files in the secondary copy are automatically modified or deleted when the corresponding file in the primary copy is modified or deleted. New files added to the primary copy will be automatically added to the secondary copy only if the appropriate **fstore** value has been set.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

semantic transparency. Allow the same command to function identically from all cluster sites. It provides, for example, for the **grep** command to have the same options and give the same results no matter where it is invoked.

semaphore. An indicator used to control access to a file: for example, in a multi-user application, a flag that prevents simultaneous access to a file.

separator. A character used to separate parts of a command or file.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

server. A program that handles protocol, queuing, routing, and other tasks necessary for data transfer between devices in a computer system.

session. The period of time during which programs or devices can communicate with each other.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the shell set command.

shadow page table. A table that maps real storage allocations (first-level storage) to a virtual machine's virtual storage (third-level storage) for use by the real machine in its paging options.

shared printer. A printer that is used by more than one work station.

shell. See **shell program**.

shell options. The shell provides two different types of options. **Set options** are put into effect with the **set** command and alter the way the shell runs. **Command line options** are entered on the command line (but not with the **set** command) and alter the way the shell starts.

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the **sh** command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system.

shell prompt. The character string on the command line indicating the system can accept a command (typically the **$** character).

shell script. See **shell procedure**.

shell variables. Facilities of the shell program for assigning variable values to names.

size field. In an inode, a field that indicates the size, in bytes, of the file associated with the inode.

SNOBOL. A programming language designed for string processing and pattern matching.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language and compiled before the program can be run.

special character. A character that has special meaning to the shell, such as ? and *.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with **directory** and **file**. See also **block special file** and **character special file**.

spool. A VM program that allocates disk areas to hold files in queues, usually while they await some sort of service. Print jobs, for example, are held in a print queue awaiting service from the printer.

spool file. (1) A disk file containing output that has been saved for later printing. (2) A file used in transmitting data among devices.

SRM. System Resource Manager.

standalone shell. A limited version of the shell program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See **memory**.

storage device. A device for storing and/or retrieving data.

string. A series of characters to be taken literally by the system. A string may be specified for a context search, for instance, or for global substitutions.

su. (1) An AIX command that runs a shell and allows you to operate there with the privileges of the specified **user** (by default **root**). (2) See **superuser**.

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the shell program started from an existing shell program.

substitution. A procedure used by a text editor like **ed** or **vi** to replace one specified string of characters with another. If a global substitution is made, all occurrences of the specified text pattern are replaced with the new one.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or de-allocation of a block in the file system.

superuser (su). (1) The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0). (2) Root permissions.

supervisor. The part of the AIX/370 Operating System control program that coordinates the use of resources and maintains the flow of processing unit operations.

synchronous. Occurring in a regular or predictable sequence.

synchronous transmission. In data communication, a method of transmission in which the sending and receiving of characters is controlled by timing signals. Contrast with **asynchronous transmission**.

system. The computer and its associated devices and programs.

symbolic link. A mechanism that lets you assign a second name to a file or a directory. It functions as a pointer to the other file or directory. There are no restrictions pertaining to source or destination locations. A file may be deleted if the only pointers to the file are symbolic links. See also **link**.

system administrator. The person at a computer installation who designs, controls, and manages the use of the computer system.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with **task dump**.

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See **kernel parameters**.

system primary site. The machine (cluster site) designated to hold the primary copy of the replicated root file system. When files are changed in the replicated root file system, the primary site for the cluster must be available.

system profile. A file containing the default values used in system operations.

system-replicated file system. One that contains files and directories accessed by many users regardless of the users' specific applications. These system files, programs and directories are replicated on different sites in a cluster.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

systems network architecture (SNA). A set of rules for controlling the transfer of information in a data communication network.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with **system dump**.

TCF. See **Transparent Computing Facility**

TCF cluster. A group of computers operating under the AIX Operating System and using the Transparent Computing Facility (TCF).

TCP/IP. See **Transmission Control Protocol/Internet Protocol**.

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text data (for example, memos, reports, and letters).

text editing program. See **editor** and **text application**.

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry each active, shared program text segment.

Token-Ring network. A network that uses a ring topology, in which tokens are passed in the circuit from node to node. A node ready to send can capture the token and insert data for transmission.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk, diskette, magnetic tape, or CD ROM on which information is magnetically recorded and from which recorded information is read.

transfer. To move data from one location to another in a computer system or between two or more systems.

transmission control characters. In data communication, special characters that are included in a message to control communication over a data link. For example, the sending station and the receiving station use transmission control characters to exchange information; the receiving station uses transmission control characters to indicate errors in data it receives.

Transmission Control Protocol/Internet Protocol (TCP/IP). A network protocol that connects to other systems that support TCP/IP. TCP/IP allows a user to send and receive mail, transfer files across a network, print files, run commands on remote systems, and log in to remote systems.

transparency. The obscuring of machine boundaries in a distributed system. The AIX/370 system supports several kinds of transparency, including name, location, semantic, data, and process transparency.

Transparent Computing Facility (TCF). A facility that automatically allows for data, process, name, location and semantic transparency. Process transparency is the ability to execute and control tasks on any cluster site, no matter where the user program is currently executing. A TCF LPP is required to obtain support.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

trojan horse. A program that can vandalize files although it completes its defined task.

truncate. To shorten a field or statement to a specified length.

trusted computing base. The total of all system components, both hardware and software, that protect data in the system.

trusted program. A program with an ID of root (superuser) known to be

free of trojan horses and computer viruses.

trusted shell. A modified command interpreter that provides a restricted environment to perform administrative tasks in a secure manner.

TTY. Designates a terminal. On a system with more than one terminal, the TTY field of the process status displayed by the **ps** command indicates which terminal started the process.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See **user number**.

UNIX link. A mechanism that lets you use the **ln** command to assign more than one name to a file. Both the new name and the file being linked to must be in the same file system. A file is deleted when all the UNIX links (including the first link--the original name) have been removed. Synonym for **hard link**

UNIX-to-UNIX Copy Program (uucp). A set of programs that allows you to copy files from a local UNIX system to a remote UNIX system. The uucp command is part of the Basic Networking Utility (BNU) program.

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See **account**.

user ID. See **user number**.

user list. A list, containing the user identification and access levels, of all operators who are allowed to use a specified file or library.

user name. A name that uniquely identifies a user to the system.

user number (Uid). A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and

defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

user-replicated file system. A file system containing files and directories accessed only by specific users or for particular applications. These user files and directories are replicated on different sites in a cluster.

utility. A service; in programming, a program that performs a common service function.

uucp. See **UNIX-to-UNIX Copy Program**.

V=F. Virtual=Fixed mode.

V=R. Virtual=Real mode.

V=V. Virtual=Virtual mode.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with **constant**.

VCTC. Virtual Channel-to-Channel.

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

vi. A full screen editor.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine (VM). A functional simulation of a computer and its related devices. A hypervisor (a program that runs operating systems)

that runs on System/370 hardware. VM can divide the processor into any number of virtual machines, each of which appears to be a complete System/370 machine.

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

VM HPO. Virtual Machine High Performance Option.

VM HPO PMA. Virtual Machine High Performance Option Preferred Machine Assist.

VM/SP. Virtual Machine/System Product.

VM/XA. Virtual Machine/Extended Architecture.

VM/XA SP. Virtual Machine/Extended Architecture System Product.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system.

VTAM. Virtual Telecommunications Access Method.

wildcard. See **pattern-matching characters**.

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

workstation. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See **programmable work station** and **dependent work station**.

working directory. See **current directory**.

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

XA. Extended Architecture.

XDR. A data definition language used as a standard to the RPC routines for remote communications. The internal data representations of various machine types are represented in a uniform format so that networked machines can communicate regardless of their manufacturer or structure algorithm.

S

V

W