IBM

ACADEMIC OPERATING SYSTEM 4.3

SYSTEM ADMINISTRATION GUIDE

ACADEMIC OPERATING SYSTEM 4.3

IBM Academic Operating System 4.3
System Administration Guide

First Edition (September 1988)

This edition applies to IBM Academic Operating System 4.3 Release 1 and to all subsequent releases until otherwise indicated in new editions or in Technical Newsletters.

Trademarks

Table of Contents

CHAPTER 1

Introduction to System Administration

1. About this Guide

The IBM Academic Operating System 4.3 (IBM/4.3) is a full implementation of the 4.3 Berkeley Software Distribution (4.3 BSD) of the UNIX[1] operating system, with enhancements such as support for the Token-Ring Network architecture and the Andrew File System, a distributed file system. The multi-user and multi-tasking capabilities of the IBM/4.3 system provide a powerful and flexible computing environment designed to support a variety of scientific and educational uses.

This guide is intended to assist you in operating and administering the IBM/4.3 system. It details the important functions involved in the everyday operation of the system, including starting up and shutting down the system, configuring the operating system, adding new users to and deleting old ones from the system, adding and configuring peripheral devices and network connections, backing up the files stored on disks, keeping track of system resources, and maintaining system security. This guide also provides help diagnosing and solving abnormal system occurrences related to these activities.

In general, this guide assumes that you have little or no system administration experience. It explains the fundamental parts of the operating system (such as the file system and access permissions) in somewhat greater detail than other parts. In system administration there is no substitute for on-the-job experience, but a good grasp of the underlying system will be invaluable in helping to track down and solve problems that arise.

2. What Is System Administration?

A system administrator (or manager) is responsible for managing all aspects of the installation and operation of a computer system. A computer system may be quite simple, consisting of a system unit and a console terminal, perhaps also with a modem and printer.

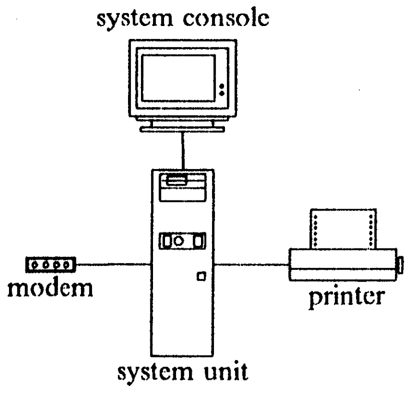


Figure 1-1: A Simple System

[1]UNIX is a registered trademark of AT&T Bell Laboratories.

On the other hand, the system you administer may be a very complex set of networked computers, each of which has one or more peripheral devices attached to it.

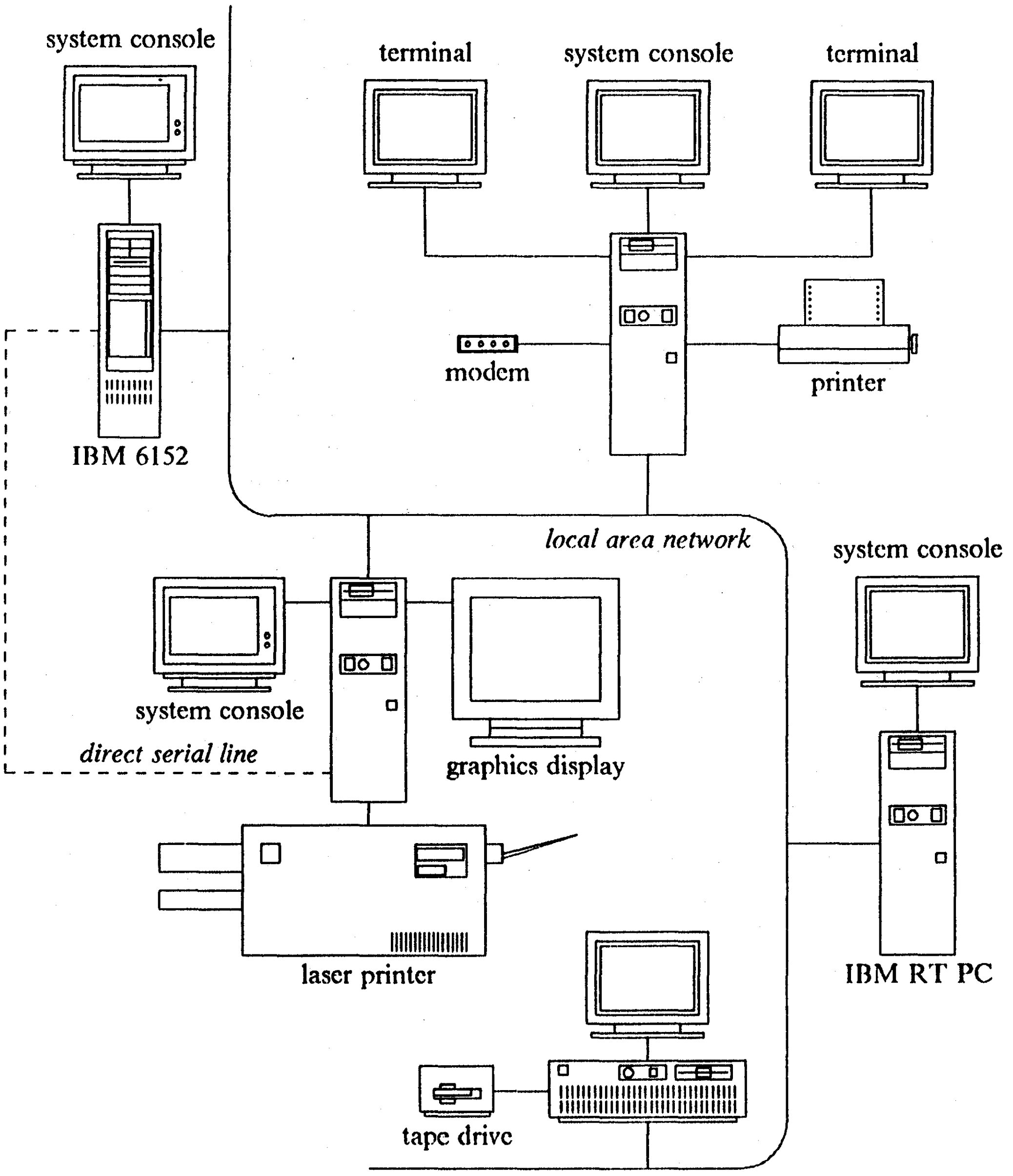


Figure 1-2: A More Complicated System

Whatever the complexity and size of the system, the primary goal of system administration is to keep the computer(s) and peripheral devices running smoothly and efficiently so that system facilities are available to the user community. To achieve this goal, the IBM/4.3 system requires careful control of its operations and a regular schedule of maintenance and backup. An administrator must oversee and maintain both hardware and software components of the system, ensure against loss or destruction of data stored within the system, and help track down problems when the system does not operate as expected.

Fortunately, the IBM/4.3 system can be instructed to perform many routine administrative tasks automatically. For example, by suitably configuring the `cron` daemon, the system administrator can have the system periodically clean out old and unused files from the file system, rotate log files, summarize disk usages, back up recently modified files and directories, and process accounting records. Automating routine system functions in this way allows the administrator to focus on more demanding (and less tedious) system maintenance operations. The IBM/4.3 system is rich with tools and tool-building utilities that you can employ to simplify the task of keeping the system running smoothly.

3. Cautions for System Administrators

When you are performing administrative tasks on the IBM/4.3 system, you will generally be logged in as the "superuser" (whose login name is usually '`root`') and you will have a tremendous amount of power within the system. In particular, you will be able to read, alter, or delete any file within the system, including the files and programs necessary to boot the system. The great power you have while logged in as the superuser must therefore be used with caution.

- Log in as the superuser *only* when necessary to perform administrative functions. Never log in as the superuser to do ordinary word processing or program compilation. It is extremely dangerous to overuse your superuser privileges as an administrator of the IBM/4.3 system.
- Be especially careful when executing commands like `rm`, `mv`, and `cp`, which may remove files that you want to keep. You should use the interactive forms of these commands (for example, `rm -i`) at all times. Ideally, you should alias these commands to their interactive forms by inserting the proper lines into your `.cshrc` file.
- Try to avoid using "wildcard" characters (such as '`*`') in file name specifications, since you may unwittingly select a file you don't really want to select. If you are using the C-shell, you should set the variable `noclobber` so that you do not overwrite existing files with output redirection constructs.
- Never leave your terminal unattended if you are logged in as the superuser. A naive or malicious user can easily remove or modify essential files and directories if presented with superuser privileges.
- Keep a system log book to record all administrative tasks you perform as the superuser. Then, other administrators can see what actions have been taken, and you can review your past actions. A well-tended log book can be your most valuable "peripheral device."
- Consult your documentation and confer with other local experts when dealing with system problems. Hasty and incorrect actions may only make a problem worse. Refrain from acting until you are reasonably sure that your plan will indeed solve the problem or allow you to pinpoint its cause.

4. Notes for System V Administrators

If you have experience in administering other UNIX-based systems, particularly the releases from the University of California at Berkeley (BSD releases), you will find administering an IBM/4.3 system relatively easy. While there are many important differences between the 4.3BSD release and earlier BSD releases, most administrative procedures have not changed very much. (Where such procedures have changed, it is largely in the direction of greater simplicity and ease of use.) Even if most of your system management experience is with System V, which is in many ways incompatible with BSD systems, you will find that you already know quite a bit about administering the IBM/4.3 system. The following sections describe the major similarities and differences between the two systems; a familiarity with them will help you to know when your System V experience applies to administering a machine running the IBM/4.3 system, and when it does not.

4.1. Similarities between IBM/4.3 and System V

Many features of System V administration and IBM/4.3 system administration are identical or nearly identical. Some of the major similarities are the following:

- You will do practically the same things to set up user accounts on the two systems. Under both BSD and System V, the file `/etc/passwd` maintains user account information; more importantly, the format of the file is the same on both systems. The IBM/4.3 system, however, maintains two hashed versions of the password data base, `/etc/passwd.pag` and `/etc/passwd.dir`, to speed up password verification. The data base files are generated by the utility `mkpasswd`, which is called automatically by `vipw`.
- The same procedures are used to mount and unmount file systems to increase or decrease the available storage space. Also, the same commands are used to check and repair file systems (except for the Andrew File System, if present).
- Nearly the same administrative accounts (such as `root`, `lp`, and `uucp`) are provided on both System V systems and IBM/4.3. BSD systems and the IBM/4.3 system provide several additional accounts intended for administrative use, most notably the `operator` account (for use by the person performing file system backups) and the `nobody` account (for use in `crontab` entries where minimal access privileges are required).
- File access permissions operate similarly on the two systems. The major difference between IBM/4.3 and previous BSD releases (and all System V releases) is that the "sticky bit " may now be set on directories as well as on executable programs. If a directory's sticky bit is set, only the owner of a file (or, of course, the superuser) can remove it from that directory, regardless of the permissions set on the file itself.
- Process accounting, or keeping track of who is doing what on your machine, works in the same way on both systems.
- Terminals and modems are connected to the system in nearly the same ways. However, the format of the terminal line configuration file, `/etc/ttys`, has changed significantly from the 4.2 to the 4.3BSD releases.
- The uucp system functions nearly identically on both types of system. If you successfully administered uucp on a System V machine, you should have no problems administering it under IBM/4.3. One main difference between the two systems is that uucp on the IBM/4.3 system can operate over TCP/IP links, whereas this capability is absent on many System V machines.

For further details, consult the relevant chapters in this guide.

4.2. Differences Between IBM/4.3 and System V

There are many important differences between System V-based systems and IBM/4.3. For this reason, you should exercise caution in applying what you learn in this guide to non-IBM/4.3 systems, or in applying to the IBM/4.3 system what you have learned administering a System V-based system. The most important differences between the two systems, from an administrator's point of view, are the following:

- The IBM/4.3 system incorporates the new "fast file system" introduced in 4.3BSD. Under this system, file systems have been subdivided into *cylinder groups*, so that a directory and the i-nodes for the files in that directory are stored as close together as possible and file fragmentation is reduced. In addition, the size of data blocks may be different on different file systems, if necessary to maximize throughput. The minimum block size is now 4096 bytes, as opposed to previous values of 512 or 1024.
- The new file system is able to manage the subdivision of blocks into *fragments*, thereby reducing the amount of space wasted when using a larger minimum block size. In addition, the traditional free list has been replaced by a free bitmap, which allows addressing of

individual block fragments.

- The IBM/4.3 system provides an extension of normal file and directory links, known as "symbolic links," that allows one file to be linked to another, even across file system boundaries. This capability is absent from most current System V systems.
- The IBM/4.3 system allows very long file names, up to 255 characters in length. In addition, a path name may be as long as 1024 characters. In System V, file names are usually limited to 14 characters in length (although implementations allowing as many as 22 characters are not uncommon). You may therefore have trouble moving files or directories from a machine running the IBM/4.3 system to one running a System V derivative.
- File systems are now created using the program `newfs`, which is a more friendly version of `mkfs`. The `newfs` command allows the administrator to tailor file system parameters toward achieving either maximum throughput or maximum space efficiency, depending on the needs of the local user community.
- In the IBM/4.3 system, a user may belong to several groups simultaneously, while in System V a user may belong to only one group at a time and must manually switch (using the `newgrp` command) from group to group if different file access permissions are desired.
- Under most System V systems, the primary networking software is the uucp system, which operates largely as a batch type process (where files are spooled for forwarding at specified times). Under IBM/4.3 both Ethernet and Token-Ring networks are available. These allow Internet communications services such as interactive remote login and file transfer. In addition, the Andrew File System provides the ability to maintain a file system distributed across a local area network.
- Backups and retrievals under System V systems are usually accomplished with either the `tar` or `cpio` utilities. Under IBM/4.3, the `dump` and `restore` commands are used to provide both full and partial backups and to retrieve files interactively from backup tapes.
- IBM/4.3 includes improved error message reporting and logging facilities using the daemon `syslogd`. Messages are categorized by origin and severity, and the system's responses to such messages can be specified with a high degree of selectivity. Error messages can also be logged from the local machine to a remote machine across a network. Note in particular that messages, even ones indicating very severe problems with the system, are *not* automatically written on the system console. Under IBM/4.3, you must specifically configure the error logging daemon to write messages there, if so desired.
- Under System V, there are no limits set on a user's disk usage, other than the limit on maximum file size provided by the `ulimit` command. Under the IBM/4.3 system, the system administrator may impose disk usage quotas to restrict the total number of disk blocks used, the number of files and directories, or both.
- Under System V (Release 2.0 or greater), normal unprivileged users may automate the periodic execution of commands by creating their own personal `cron` configuration files in the directory `/usr/spool/cron/crontabs`. (Such users cannot edit their `crontab` files directly, however. They must use the `crontab` command to retrieve and replace those files.) Under IBM/4.3, only the system administrator may alter `crontab` files; there is no provision for user-specific `crontab` files.

Again, you can find greater detail on any of these differences in a later chapter of this book. See the following Summary of Chapters for a description of each chapter's contents, or refer to the Index at the end of this guide.

5. Summary of Chapters

Each chapter in this guide is devoted to a topic of concern to IBM/4.3 system administrators. The guide contains the following chapters:

Chapter 1, Introduction to System Administration provides a general introduction to administering an IBM/4.3 system and to this guide, including some important cautions for new system administrators. This chapter also discusses some major differences between IBM/4.3 and System V machines.

Chapter 2, Starting Up and Shutting Down the System describes the steps required to bring the system into operation and to take it out of operation. This chapter also discusses the various run levels of the IBM/4.3 system and how to recover from a system crash.

Chapter 3, Reconfiguring the Operating System describes the steps involved in creating, installing, and booting a new operating system (or kernel). It also discusses the format and contents of kernel configuration files, and illustrates a sample kernel reconfiguration.

Chapter 4, Managing User Accounts describes the process of maintaining user accounts, including adding accounts, removing accounts, and establishing passwords. The system accounts provided with the IBM/4.3 system are discussed, along with some recommendations for adding further system accounts.

Chapter 5, Using File and Directory Permissions discusses how IBM/4.3 manages file and directory access by maintaining a set of permissions. Related topics include `setuid` and `setgid` files and the mechanism of "sticky bits."

Chapter 6, Managing Terminals and Modems illustrates how to add terminals and modems to a machine running IBM/4.3, including hardware hook-up and software configuration. The `termcap` data base is discussed briefly.

Chapter 7, Creating and Maintaining File Systems provides essential information on the structure of IBM/4.3 file systems, how to create new file systems, add them to (and remove them from) the existing file system hierarchy, and maintain the integrity of the entire file system.

Chapter 8, Backing Up and Recovering Files discusses backing up and restoring the IBM/4.3 system file structure using the utilities `dump` and `restore`. Also discussed are the different types of file system backups that you will perform, as well as a recommended backup schedule.

Chapter 9, Managing the Line Printer System explains the configuration and operation of the line printer system. This chapter illustrates the normal use of the system and tries to provide enough troubleshooting advice to help you debug a malfunctioning system. It also provides an introduction to constructing printer capability descriptions and printer interface programs.

Chapter 10, Understanding the Uucp Network covers setting up and maintaining network operations using the uucp networking programs, a batch-type system that provides for remote file copying and remote command execution.

Chapter 11, Implementing Local Area Networks gives an overview of of local area networks and lists available sources of information.

Chapter 12, Managing USENET presents an overview of the USENET network used to exchange information, programs, and miscellaneous messages across the world.

Chapter 13, Accounting discusses user connect time accounting and system resource accounting as they are implemented on the IBM/4.3 system.

Chapter 14, Administering Quotas shows how to install, configure, and start up the optional disk quota subsystem that allows system administrators to place limits on the amount of disk space used by an account, on the number of files owned by an account, or on both. Data compaction to reduce space consumption is also illustrated.

Chapter 15, Handling System Messages describes the configuration and operation of the IBM/4.3 system message logging facilities, provided mainly by the daemon process `syslogd`.

Chapter 16, Executing Commands Periodically explains the configuration of the clock daemon, `cron`, which is used to schedule tasks for periodic execution.

Chapter 17, Controlling Log Files identifies various log files in the IBM/4.3 system that can, if left unmanaged, grow without limit. This chapter also presents several techniques useful for finding and truncating such files.

Chapter 18, Implementing Security outlines a number of areas where the system administrator must guard the system against intrusion or misuses. Topics include protection for passwords, `setuid` programs, special devices, ordinary files, and electronic mail. Security across networks (uucp, Internet, and the Andrew File System) is also discussed.

Chapter 19, Understanding the Andrew File System and the Andrew Toolkit explains the operation and maintenance of the distributed Andrew File System, and gives information on using the Andrew Toolkit, a set of user applications.

Please note that for information on the Network File System and the X Window System you should refer to the following sources:

- For information on the Network File System, see the article entitled "IBM Network File System Administration Guide" in the IBM/4.3 documentation.
- For information on the X Window System, see the article entitled "The X Window System" in the IBM/4.3 documentation.

6. Typographical Conventions

Throughout this guide, sample command lines and shell scripts are displayed in an effort to make concrete the procedures involved in normal system administrative tasks. To understand the examples, you should bear in mind several important typographical conventions. Text that is to be typed "as is", such as a command name or a directory name, is set in a constant-width, typewriter-like font, `like this`. Output from commands, if any, is also listed in this font. For example, a C shell command that will move you to your home directory would be displayed like this:

```
# cd ~
```

Notice that the shell prompt ('#') is included in this example to underscore the fact that this is a command given directly to the shell.

Text that represents a parameter that needs to be appropriately substituted before a command line can actually be executed, such as a size or a part of a file format, is set in an italic font, *like this.* The word or words italicized indicate the kind of substitution required. For example, a command to move you into an arbitrary temporary directory might be schematically given as follows:

```
# cd temporary-directory
```

When you want to actually execute such a command, you would need to substitute a suitable value for the string '*temporary-directory*'. For example, an actual command line adhering to this format is the following:

```
# cd /tmp
```

Another possible command line might be:

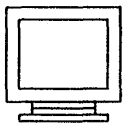
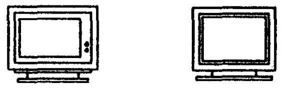
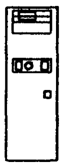
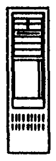
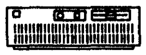
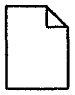
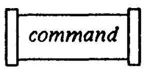
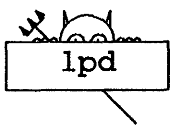
```
# cd /usr/tmp
```

And there are no doubt other possible substitutions, if your system contains other temporary directories.

It is important to note that some other publications you consult, in particular the manual pages, may not conform to the conventions followed in this guide.

7. Pictorial Conventions

This guide contains numerous illustrations designed to facilitate your understanding of the operation and administration of the IBM/4.3 system. In these illustrations, a number of standard items appear, the principal ones of which are summarized in the following table:

Pictorial Element	Description
	An IBM Academic Information Systems Experimental display or other high-resolution display capable of bit-mapped graphics.
	A system console monitor or other terminal providing serial communications with a host computer.
	An IBM 6150 RT Personal Computer.
	An IBM 6152 PS/2.
	A table-model RT PC.
	Any file located in the file system hierarchy, usually (but not always) a plain text file.
	Any command or command pipeline. Usually, the standard input, if any, goes in the left side and the standard output, if any, emerges from the right side.
	A daemon process (i.e., a process that sits waiting for something to occur and takes appropriate action when it does).

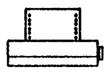
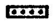
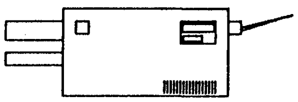
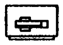
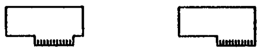
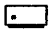
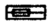
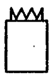
	The IBM 5152 Personal Computer graphics printer or other impact printing device (such as a line printer).
	A modem, used for connecting a terminal or host computer to some computer through telephone lines.
	A 3812 Pageprinter, or other high-quality output device.
	A streaming tape drive.
	Any add-in expansion or adapter card.
	A hard disk drive.
	A floppy diskette drive.
	The kernel, `/vmunix` (the "king" of programs).
	Any serial port.
	A socket (endpoint for communication).

Table 1-1: Pictorial Conventions Used In This Guide

8. Other References

This manual is designed to serve as an introduction to system administration and to provide a general reference to guide you in performing typical administrative tasks on the IBM/4.3 system. It does not provide information on specific types of hardware devices that you may wish to connect to your system, so you will need to consult the hardware installation instructions accompanying such devices. This guide does not discuss the procedures to be followed in installing the IBM/4.3 system software. For this information, consult the document "Installing and Operating Academic Information Systems 4.3", in Volume 2 of the IBM/4.3 documentation. Nor is this guide intended as a replacement for the manual pages (in the BSD documentation or the IBM/4.3 documentation), which provide complete (though terse) information on the syntax of administrative commands and on the options available for each command. As you investigate the procedures involved in typical administrative tasks, you should read through the relevant chapters of this guide and have the relevant pages of the reference manuals open in front of you.

CHAPTER 2

Starting Up and Shutting Down the System

1. Introduction

The IBM/4.3 system is designed to allow continuous operation of the system. Barring unforeseen events such as power failures or system crashes, the IBM/4.3 system will continue in operation for weeks and even months, until the system manager shuts it down. A system shutdown is usually necessary to install additional system hardware or peripheral devices, to perform routine preventive maintenance, or to transport the machine to another location. A shutdown may also be necessary simply to conserve electricity when the system would otherwise be unused. This chapter outlines the steps required to bring the system into operation and to take it out of operation, also called "booting" and "halting" the system.

A machine with the IBM/4.3 system installed on it is be in one of several modes at any time. When you turn the machine on, the machine will run a boot program and then attempt to start up the IBM/4.3 system into one of two available operational states or "run levels." The current run level determines what processes and daemons are active, as well as what file systems are available and who (if anyone) can log into the system. During a normal boot process, described below, the IBM/4.3 system checks its disks and comes up in a run level called "multi-user mode." This is the normal mode of operation on the system, allowing numerous users to log in to the machine and execute commands. The other run level is "single-user mode" which is used primarily for administrative tasks. Finally, there is a special mode of operation called the "standalone utility shell" which allows you to perform administrative or diagnostic functions without the aid of the IBM/4.3 system.

The IBM/4.3 system start-up and shutdown procedures are reasonably automatic. For example, once the machine has been turned on and if correctly configured, the system will go directly from a powered-off state into multi-user operation without operator intervention. Similarly, the administrator can bring the system from a multi-user state to a quiescent state (suitable for powering down or rebooting) with a single command. Nevertheless, you must understand the various steps involved in these processes well enough that you will be able to intercede and provide corrective action if necessary.

This chapter also discusses how to recover from a system crash, when for some reason the operating system ceases to operate at all. If your system crashes unexpectedly, you will also need to refer to the discussion of file system creation and maintenance procedures in a later chapter, since it is possible that the crash caused damage to the file system.

2. The Boot Program

When you first turn on (or "power on") your machine, the console screen will remain blank for a few seconds as the machine runs some internal hardware consistency checks. When the system has completed these diagnostics, it will load a boot program into memory and execute it. If everything goes normally, a message similar to the following will appear on the screen:

```
4.3BSD UNIX Standalone Boot Program $Revision: 1.7 $
Default hd(0,0)vmunix (just press Enter or wait ~30 seconds
or press <CTRL>-C for menu)

:
```

(Note that the revision level may differ in your message.) The colon at the end of the message is the "boot prompt." It indicates that the boot program is awaiting input from the console operator. If you do nothing, the boot program will wait for the specified amount of time, then undertake its default action, which is to run the command:

```
hd(0,0)vmunix
```

This command instructs the boot program to look in the root partition of hard disk number 0 (i.e., /dev/hd0a) for a program named vmunix. The program vmunix is the "kernel" or heart of the IBM/4.3 operating system that, when running, will manage the resources of the machine. If the boot program finds vmunix there, it will load vmunix into memory and exit, turning control of the machine over to the new program.

It is possible that your system was configured with its root file system in some other location, or indeed in several locations. For example, if the root partition (and hence the kernel) is located on disk number 1 (i.e., /dev/hd1a), then to start up the system you would type:

```
hd(1,0)vmunix
```

If your root partition isn't located on hard disk 0, the boot program may ask you the location of the root device once again, by displaying the question:

```
root device?
```

In this case, the answer should be hd1 (or wherever the root partition is actually located). The default location for the root file system at installation is hard disk 0, so the default action should succeed in starting up the operating system and initiating multi-user operation. If it does, you can skip ahead to see what procedures will automatically be executed and how you can tailor those procedures to suit your local needs and preferences. If the boot program does not successfully start up the operating system, you will need to do some troubleshooting. Fortunately a standalone utility shell is provided for precisely this purpose.

3. Standalone Utility Shell

The standalone utility shell allows you to perform some simple administrative tasks without having the IBM/4.3 system running. You will probably use the standalone shell primarily to format and check disks prior to creating file systems on them, but a number of other functions are supported as well. For example, from the standalone shell you can list the contents of any directory in the system (using a standalone version of the ls command). This is useful if the kernel, /vmunix, becomes damaged and you need to find a backup copy of it in the file system to boot the system.

There are two ways to enter the standalone utility shell: by loading it from a diskette, or by loading it from the hard disk. To load the standalone shell from the hard disk, you would give the following command in response to the boot prompt (usually a colon, :):

```
: ^c
```

(Hold down the Control key and type the letter 'c'.) This command invokes the program sautil, which presents a menu similar to the following:

```
4.3BSD UNIX Standalone Maintenance Program $Revision: 1.7 $

Choice          Description

1   boot - boot standalone program or kernel
2   format - format hard disk
3   dump - display disk or diskette (hex)
4   cat - display a file contents (ASCII)
5   ls - print directory of UNIX filesystem
6   copy - copy all/part of disk or floppy
```

```
7   debugger - display memory, etc.
8   iplsource - set boot order in nvram
9   minidisk - display/change minidisk directory
10  dosboot - boot standalone program or kernel from DOS diskette
11  convert - convert R70 ("hd70r") disk to E70

Enter the menu choice desired then press Enter.

Choice ?
```

You choose the function you want by typing its number. For example, to list the contents of a directory, you type '5'. You are then prompted for the location and name of the directory to be listed. To list the root directory, you type:

```
hd(0,0).
```

The output of this selection looks something like this:

```
2          .
2          ..
4          .profile
768        bin
5          boot
1163       dev
384        etc
1152       lib
3          lost+found
2304       mnt
6          sys
16         tmp
1920       usr
11         vmunix
Directory:
```

To list some other directory, for example `/etc`, replace the dot (`.`) with the name of that directory:

```
hd(0,0)/etc
```

The standalone shell reads the correct disk partition and lists the contents of the specified directory. For complete details on the functions available under the standalone utility shell, see `sautil`(8R).

4. Single-User Mode

When IBM/4.3 is in single-user mode, only the console operator can use the system. No logins are allowed and only the console terminal is active. Single-user mode is the recommended run level for most administrative tasks such as file system checking (with the `fsck` program) and backing up or restoring the disks, since file system activity is at a minimum. Generally, no mounted file systems are available in single-user mode, although the system administrator may mount them manually, if desired.

There are several ways to enter single-user mode. If you have just powered up the machine and the boot program is running, you can enter single-user mode by typing the following command:

```
hd(0,0)vmunix
```

Note that this is the same command that the boot program would run by default, except that *you typed it in* and did not wait the specified 30 seconds for the boot program to issue it. The boot sequence will proceed as normal, though in some cases the boot program may ask you for the location of the root device:

```
root device?
```

Typically the correct response is 'hd0'. Once you have entered the response to this prompt, the boot process will continue and the system will come up in single-user mode.

If you neglect to type in the appropriate boot command within the 30 second period, you can still bring the system up in single-user mode by typing a ∧c (hold down the Control key and type the letter 'c') after the system prints the date.

It is also possible to enter single-user mode from multi-user mode. This is usually done during the sequence of shutdown operations or whenever the system administrator wants to ensure that maintenance tasks (such as backups) will operate on quiescent file systems. If the system is running in multi-user mode and you want to bring it into single-user mode, type the command:

```
# kill 1
```

Before doing this, however, make sure that no one else is logged on the system, or you are likely to disrupt their work. The complete sequence of commands you should issue to ensure this is given in the section below on shutting down the system.

5. Multi-User Mode

Multi-user mode is the run level that permits multiple users to access the fullest range of system resources; consequently, it is the mode that the system will be in most of the time. In multi-user mode, numerous daemons are active to oversee system functions such as spooling print jobs, executing periodic commands, logging system messages, and handling network communications. All terminal lines specified as active in the file `/etc/ttys` have `getty` processes started on them so users can log in from those terminals. In addition, all file systems listed in the file `/etc/fstab` will be mounted.

The process of bringing the system up to multi-user mode is handled largely by the `init` program, which is invoked as the last step of the boot process. It is the `init` process which runs the multi-user start-up files described below and which then starts a login process on various terminals.

5.1. Multi-User Start-Up Files

When IBM/4.3 enters multi-user mode, it consults a file, `/etc/rc`, for a list of the commands necessary to start up processes that distinguish single-user from multi-user mode. (`rc` stands for "run commands.") The file `/etc/rc` is a Bourne shell script that contains whatever commands the system is to run as it enters multi-user mode. It is called the "multi-user start-up file." Since `/etc/rc` is a shell script, it is quite easy to modify using your favorite text editor. You can even program it (for example, to run certain commands conditionally) using the standard Bourne shell constructs. However, be sure you know what you are doing when you modify this file; a syntax error or a command that loops forever may cause the system to fail to enter multi-user mode. (In that situation, the only recourse is to bring the system up in single-user mode to correct your mistakes.)

Any command can be put into the `/etc/rc` file, but it is best to keep the file fairly small by including only those commands absolutely necessary to resume multi-user operation after a system boot or reboot. Typically the first commands in `/etc/rc`, after some introductory comments, are the following two:

```
HOME=/; export HOME
PATH=/bin:/usr/bin
```

These commands set the home directory and search path, so that subsequent commands do not have to include their full path specifications. Following this, the system decides whether it should check the disks (by running the fsck program):

```
if [ -r /fastboot ]; then
      rm -f /fastboot
      echo Fast boot ... skipping disk checks > /dev/console
elif [ $1x = autobootx ]; then
      echo Automatic reboot in progress... > /dev/console
      date > /dev/console
      /etc/fsck -p > /dev/console
      case $? in
      0)    date > /dev/console
            ;;
      2)    exit 1
            ;;
      4)    /etc/reboot -n
            ;;
      8)    echo "Automatic reboot failed!" > /dev/console
            exit 1
            ;;
      12)   echo "Reboot interrupted" > /dev/console
            exit 1
            ;;
      *)    echo "Unknown error in reboot" > /dev/console
            exit 1
            ;;
      esac
else
      date >/dev/console
fi
```

As you can see, the disks will not be checked if the file /fastboot exists. This file is created by the program /etc/fastboot (and also by the program /etc/fasthalt) in order to have some easy way to indicate to /etc/rc not to check the disks. The set of commands following the second then is run only if the system is being automatically rebooted (in which case the rc script was executed with the argument autoboot). The disk checking program fsck is run with the -p option to "preen" the disk of any inconsistencies. Depending on the exit status of the fsck command (the value of the variable '$?'), the rc script will either continue executing commands, or exit with an error status.

The next set of commands in a typical /etc/rc file is designed to ensure that the password file, /etc/passwd, has not been corrupted. The password file is essential for logging in, so it is important that a reliable version of it exist before users attempt to log in. If the file /etc/ptmp exists, the password file has very likely been corrupted; /etc/ptmp is a temporary file created and usually removed by the command vipw.

```
if [ -s /etc/ptmp ]; then
      if [ -s /etc/passwd ]: then
            ls -l /etc/passwd /etc/ptmp > /dev/console
            rm -f /etc/ptmp
      else
            echo 'passwd file recovered' > /dev/console
            mv /etc/ptmp /etc/passwd
      fi
elif [ -r /etc/ptmp ]; then
      echo 'removing passwd lock file' > /dev/console
      rm -f /etc/ptmp
fi
```

This section of the /etc/rc script illustrates an important application of the script, namely to clean up the file system and several configuration files before entering multi-user mode. Here, if the file /etc/ptmp exists, it will be removed. Other similar tasks usually assigned to /etc/rc include cleaning out temporary file directories, removing lock files, initializing a list of mounted file systems, and updating certain data base files. This is all accomplished by including commands like the following:

```
/etc/umount -a
cp /dev/null /etc/mtab
/etc/mount -a  > /dev/console 2>&1
/etc/swapon -a > /dev/console 2>&1
/bin/ps -U     > /dev/console 2>&1
rm -f /etc/nologin
rm -f /usr/spool/uucp/LCK.*
chmod 666 /dev/tty[pqrs]*
```

Toward the end of the file /etc/rc, you will find the following command:

```
sh /etc/rc.local
```

This instructs the system to read and execute any commands found in the file /etc/rc.local. Typically this file contains any multi-user start-up instructions that are local to your machine. For example, not all IBM/4.3 systems are attached to a local area network, so the commands necessary to start up network operation are usually put into /etc/rc.local instead of /etc/rc. If your machine is attached to a network, you will probably find commands similar to the following in /etc/rc.local:

```
hostname poisson
ifconfig un0 inet `hostname` netmask 255.255.255.0
ifconfig lo0 localhost
hostid `hostname`
```

The netmask is used to subnet a class B network. For example, the IP address 129.33.222.8 contains a network number 129.33.222 and a host number 8. You do not need to include the Internet address in the ifconfig command. Ifconfig uses the host name to look up the address in /etc/hosts.

After the local start-up script has been executed, control returns to the /etc/rc file. Three principal steps remain: preserving editor files, starting local daemons, then starting network daemons. In part, the remaining section of the script looks something like this:

```
echo preserving editor files        > /dev/console
(cd /tmp; /usr/lib/ex3.7preserve -a)

echo -n standard daemons:           > /dev/console
/etc/update;     echo -n ' update'  > /dev/console
/etc/cron;       echo -n ' cron'    > /dev/console
/etc/accton /usr/adm/acct;
                 echo ' accounting' > /dev/console

echo -n starting network daemons:   > /dev/console
if [ -f /etc/rwhod ]; then
     /etc/rwhod;
     echo -n ' rwhod'               > /dev/console
fi
if [ -f /etc/inetd ]; then
     /etc/inetd;
     echo -n ' inetd'               > /dev/console
```

```
    fi

    exit 0
```

The /etc/rc file should end with the exit(0) command as shown, so that init knows that the script has successfully accomplished its goals. Init will then proceed to start a getty process on the terminal lines and take logins. Multi-user start-up and initialization has been completed. At this point, a login banner should appear on the system console and on all terminals attached to the system that are configured to accept logins.

6. How to Shut Down the IBM/4.3 System

To shut down the IBM/4.3 system, you must do more than just turn off the computer. A deliberate shutdown procedure must be followed to ensure that the file system is in a clean and uncorrupted state and that the disks are not being accessed when the power is turned off. This primarily involves notifying users that the system is to be shut down, terminating all active processes, switching to single-user mode, unmounting any mounted file systems, and updating the disks. Fortunately, the utility program shutdown is available to provide the system administrator with an automated shutdown procedure.

6.1. Shutting Down Automatically

The shutdown program automatically leads the system administrator through the steps required to bring the IBM/4.3 system either to its powered-off state or to a state from which it can be rebooted. The simplest way to invoke the command is as follows:

```
    # shutdown now
```

The argument now requests an immediate shutdown, so the procedure will begin at once. In the future you can replace now in either of two ways:

+number

to request that the shutdown begin in *number* minutes, or in the format:

hour: *min*

to request that the shutdown begin at the specified time of day, using a 24-hour clock.

Regardless of how the command is invoked, a series of warning messages appear on the terminal screens of all users who are currently logged in, informing them of the impending shutdown. At this time, users should cease running processes and log off. They should also ensure that they have no processes running in the background. You can check to see that users have complied with your request to terminate all processing by running the ps command. A useful format can be obtained with the -aux options, as illustrated:

```
    # ps -aux
    USER       PID %CPU %MEM   SZ  RSS TT STAT   TIME COMMAND
    root     20582 10.2 11.4  290  236 co R      0:00 ps -aux
    root        91  0.2  2.8   76   50 ?  S     34:31 /etc/rwhod
    root        52  0.0  2.7   98   48 ?  S      9:17 /etc/routed
    root        57  0.0  7.7  206  156 ?  I      0:33 /usr/lib/sendmail -bd -q30m
    root        80  0.0  0.8   12    8 ?  I      5:15 /etc/update
    root        45  0.0  2.1   90   36 ?  I      0:03 /etc/syslogd
    root         2  0.0  0.5  800    0 ?  D      0:02 pagedaemon
    root         1  0.0  1.1   68   14 ?  I      0:03 /etc/init -
    root         0  0.0  0.4    0    0 ?  D      0:03 swapper
    root        83  0.0  2.5   58   44 ?  I      0:43 /etc/cron
    root        95  0.0  0.9   86   10 ?  I      0:02 /etc/inetd
    root       100  0.0  4.4  108   84 ?  I      0:01 /usr/lib/lpd
    root     20481  0.0  4.3  192   82 co I      0:03 -csh (csh)
```

As you can see, no one but `root` remains logged into the system and the only processes still running are special processes or daemons. Consequently, it is safe to continue the shutdown procedure.

At five minutes before the specified shutdown time (or immediately, if a shorter interval was requested), the operating system disables further logins by creating the file `/etc/nologin`. The `shutdown` program writes a message there indicating that the system is going to be shut down. If someone attempts to login to the system after this file has been created, the `login` program will print the message contained in `/etc/nologin` on the user's terminal, then exit. This file is removed just before the `shutdown` program exits, so that the system allows logins once it has been rebooted. In addition, the operating system makes certain that the `/etc/nologin` file is removed if it still exists at multi-user start-up time.

At the appointed time, `shutdown` causes the system to enter single-user mode. The system administrator is left with a shell on the system console. Any mounted file systems remain mounted, so you must now unmount them to complete the shutdown procedure. Execute the commands:

```
# cd /
# sync
# sync
# /etc/umount -a
```

The shutdown sequence is almost complete. You still need to update the root file system by executing the `sync` command. The `sync` command is one of the simplest user-level commands in the IBM/4.3 system, since it does nothing more than execute the `sync`(2) system call, which causes all buffered disk updates to be written out to the disk. As you can imagine, however, this command is also one of the most essential administrative commands, since halting the processor without executing it may leave the disks in a non-current state, which can make the file system inconsistent. As just illustrated, it is recommended that you execute this command several times to make absolutely sure that the contents of in-core buffers are flushed to disk. The disk updating is not necessarily complete just because you get a shell prompt back. To be safe, issue the command two or three times:

```
# sync
# sync
# sync
```

To power down the machine, first halt the processor with the `halt` command:

```
# halt
```

Now wait until you see the following response from the system:

```
syncing disks...done
halting (via wait);
```

It is now safe to turn off the machine. To return to multi-user operation without powering down, type an end-of-file indicator instead of the `halt` command:

```
# ^d
```

The system will re-enter multi-user mode. The start-up script `/etc/rc` will be executed, and multiple users will once again be able to use the system.

A number of options are available with `shutdown` to allow the system administrator to override its default behavior. For example, the option `-r` requests that the `reboot` command be executed once the system reaches single-user mode. Similarly, the `-h` option will cause `shutdown` to execute the `halt` command automatically at the appropriate time. For a complete list of the available options, see `shutdown`(8).

6.2. Shutting Down Manually

In some rare instances, you may need to bring the system down manually instead of using the `shutdown` command. You do so by following these steps. First of all, notify any users who may be logged in that you are about to bring the system down. Either inform them personally (if there are only a few users located close by) or use the `wall` command:

```
# /bin/wall
System being brought down in 10 minutes!  Please log out.
∧d
```

This command writes the specified message on the terminal screens of all users who are currently logged in, including the system console. When everyone else has stopped all processing and logged off, enter single-user mode with the following command:

```
# kill 1
```

The system responds:

```
# Feb 16 10:18:49 master syslogd: going down on signal 15
erase ^H, kill ^U, intr ^C
```

To indicate that you have entered single-user mode, the system also changes your shell prompt to the following string:

```
master#
```

(The string `master` is be replaced by the hostname of your machine.) Now issue the `sync` command several times and unmount any mounted file systems. Execute the commands:

```
master# cd /
master# sync
master# sync
master# /etc/umount -a
```

Finally, update the root file system and halt the processor:

```
master# sync
master# sync
master# sync
master# halt
```

It is now safe to power off the system. A record of the shutdown prints on the console screen (perhaps too fast to read). If the `syslogd` error-logging daemon is correctly configured, a record is also placed in the log file `/usr/adm/shutdownlog`. See the chapter "System Messages" for complete instructions on how to do this.

7. System Crashes

A system "crashes" when it ceases operation without having been told to do so. Many conditions, both software- and hardware-related, can cause a system to crash. For example, if the system is unable to write to or read from the paging device, the system will crash; when it does so, it may display the following short message on the system console:

```
panic: hard IO err in swap
```

The notation `panic` is an indication that the system has crashed "voluntarily." This means that the system detected the error while performing one of its internal consistency checks and was unable to recover from the error. More importantly, the system "knew" that it had encountered an error that would make continued operation of the system impossible, and printed the message indicating what had gone wrong. When a system crashes voluntarily in this way, it attempts to save (or "dump") an image of the core memory onto a hard disk for post-mortem crash analysis; whether or not it succeeds, it then automaticallys invoke the reboot procedure to attempt to resume multi-user operation.

It is also possible for a system to crash "involuntarily," in which case the system does not display a message or even try to create a core memory dump. Instead of attempting to reboot itself, the system hangs in a frozen state: no input is accepted from users and no output is produced by the system.

7.1. Recovering from a Crash

In the event of a system crash, you should do the following:

(1) Make sure that the system has actually crashed. Users may not get any response from the system, and hence think that the system has crashed, but there are a number of other reasons for a lack of response. For example, the cable connecting a terminal to the system may have accidentally become disconnected, or the user may unknowingly have typed a *∧s*, thereby stopping any output. Also, users logged in across a local area network may mistake a network problem for a system crash. To make certain that the system has crashed, check the system console. If the system has crashed voluntarily, it most likely displayed a `panic` message on the system console. If the system crashed involuntarily, however, there may be no indication at all on the console terminal. If commands typed on the console keyboard are not echoed, the system has probably crashed.

(2) Make an entry in your system log book indicating the time and date of the system crash. If an error message printed on the console screen, make a note of the message.

(3) Reboot the system if it did not automatically reboot itself.

(4) Run the file system checker, `fsck`, if the reboot process did not run it automatically. This step is essential to maintain a healthy file system. If the system crashed during a disk read or write, or before the in-core buffers could be flushed to disk, it is very likely that the file system was damaged. `Fsck` attempts to correct any inconsistencies it finds. **Do not run `fsck` on any Andrew File Systems present on your system; you will destroy them. Instead, run `vfsck` on them.**

(5) If the system crashed voluntarily and succeeded in producing a core dump, you may want to analyze that dump to determine what caused the crash. See the following section for an overview of this process.

7.2. Analyzing Crash Dumps

In addition to printing a `panic` message on the console screen when it crashes voluntarily, the system also attempts to create an image of the core memory on a mass storage peripheral device. Usually the device chosen is the same as the primary swap area, so that important files in the file systems are not endangered. Also, the memory image is written at the end of the swap area so that the system can try to reboot. To be used for crash post-mortem analysis, this image must then be salvaged from the swap area after the system is rebooted.

You can move the core image from the swap area by executing the `savecore` program. This program performs two main functions. First, it moves the core image from the swap area into a file called `vmcore.1` in whatever directory is provided as an argument to `savecore`. (If the file `vmcore.1` already exists, the `1` is replaced by the lowest integer not already in use.) At the same time, `savecore` saves the namelist in a file called `vmunix.1` (again, where the `1` may replaced by some other numeral). Second, `savecore` sends a reboot message to the system message logging daemon, `syslogd`. The `savecore` command is usually included near the end of the multi-user initialization file, `/etc/rc.local`, so that it is run automatically each time the system goes multi-user. (It is placed in `/etc/rc.local` instead of `/etc/rc` because the desired location of the saved core is usually site-specific.) See `savecore`(8) for further details on this process.

Once moved to some location in the file system, the core image, or "dump", may be used to investigate the cause of the crash. To analyze a dump, you should run the utility adb, as follows:

```
# adb -k vmunix.1 vmcore.1
```

If this core image is the result of a panic, the panic message prints. (This is useful if the system rebooted itself automatically and the original panic message has scrolled off the top of the screen.) Then run the command:

```
$c
```

which asks for a C stack trace from the point of the crash; this may provide some clues as to what went wrong. If you are not experienced with adb, see adb(1).

8. Troubleshooting Hints

Occasionally you may experience problems of various sorts in booting the system or in changing run levels. This section lists some common problems that may trip you up and suggests solutions to them.

- Remember that in single-user mode only the root file system (probably called /dev/hd0a) is mounted. Files on other file systems will not be available, unless they are mounted manually (using the mount command). Generally this is not a problem, since those files will not be listed when an ls is performed. The main exception to this involves files which are symbolically linked to files in an unmounted file system. For example, the file /etc/gettytab as distributed is symbolically linked to the file /usr/ldir/etc/gettytab. When a file is symbolically linked to a file on an unmounted file system, it may appear that the file is available on the mounted file system when in fact it isn't. For example, gettytab will appear in the output of an ls command performed in the /etc directory, but you will not be able to read or edit the file. To make the symbolic link active, you must first mount the file system to which the link points. For more discussion of symbolic links, see Chapter 7.
- The command df prints the amount of free disk space on all normally mounted file systems, whether or not they are all currently mounted. It does so by inspecting the file /etc/fstab and the list of all currently mounted file systems maintained in /etc/mtab. Only mounted file systems are listed in the sixth column. Compare the following two outputs from df:

```
# df
Filesystem    kbytes    used   avail capacity  Mounted on
/dev/hd0a       7413    6672     370    95%     /
/dev/hd0g      41299   24001   15233    61%     /usr
/dev/hd1g      41299   11175   25097    31%     /usr/src
# sync
# umount /dev/hd1g
# df
Filesystem    kbytes    used   avail capacity  Mounted on
/dev/hd0a       7413    6672     370    95%     /
/dev/hd0g      41299   24001   15233    61%     /usr
/dev/hd1g      41299   11175   25097    31%
```

  The command mount also inspects /etc/mtab and hence should provide an accurate picture of which file systems are actually mounted. However, the mount table may have been left in an inexact state if the system was not shut down cleanly. Accordingly, the start-up script /etc/rc initializes the mount table by unmounting all mountable file systems, then truncating /etc/mtab to zero length.

- In some rare instances, the system may not boot correctly from a powered-down state. When you receive the boot prompt and press return, you may see diagnostic messages similar to these:

  ```
  hd0c adapter f00001f0 didn't interrupt
  hd0c adapter f0000170 didn't interrupt
  fd0c adapter f00003f2 IRQ 6 CPU level 4
  [some lines deleted]
  root on fd0
  panic: swap blocks <=0
  ```

  If this occurs, you can either boot the system from a floppy diskette or load the standalone utility shell and execute the boot program from within the utility shell. The second method is generally successful in helping the system "find" the disk controller hd0c.

CHAPTER 3

Reconfiguring the Operating System

1. Introduction

The kernel is the heart of the IBM/4.3 operating system and is responsible for starting processes, managing available core memory, and reading and writing the system's secondary storage media. It services requests received from the user's shell program and manages all hardware interactions. Only by accessing the kernel (by issuing a command to a shell) does a user actually get work done on the IBM/4.3 system.

Because the hardware on one machine may differ from that on another, you need to customize the kernel for each actual hardware environment. For example, there may be a network adapter installed on one machine but not on another; the operating system for the latter machine need not include the network access routines. The process of creating customized kernels is called "kernel reconfiguration." By reconfiguring the kernel of a particular machine, you can control the peripheral devices available to the system, the layout of disk partitions and file systems, network access to the machine, and other hardware-related features.

Although kernel reconfiguration is largely concerned with tailoring the system to the available hardware, it is not limited to hardware components. It also changes the system's idea of the time zone in which it is operating, the maximum number of users allowed to log in at any one time, the internal system identification, and other parameters that are not directly related to the underlying hardware. Moreover, you may need to reconfigure your kernel to use optional software such as the Andrew File System or the disk space quota subsystem. Reconfiguring the kernel enables the operating system to provide the desired functions in your specific hardware and software environment.

The process of reconfiguring the kernel is accomplished in part by the utility program `config`. This tool reads a file describing system parameters and hardware devices, and generates files which are used, together with the IBM/4.3 system source code, to build a new IBM/4.3 kernel appropriate to that environment. Finally, this new kernel must be installed as `/vmunix` and the system rebooted before the new configuration can take effect. This procedure greatly simplifies system maintenance by isolating most system dependencies in a file that is easy to read and modify.

This chapter describes the steps required to create, install, and boot a new kernel. It is intended to accompany, rather than replace, the article "Building IBM/4.3 Systems with Config" in Volume II of the IBM/4.3 documentation. That article provides a terse but complete description of the configuration process. It also provides several important appendices describing the grammar used by `config` to parse configuration files, the rules to assign default values for device parameters, and several other topics.

2. Overview of the Reconfiguration Process

To reconfigure a kernel, you must have the IBM/4.3 system source code available on a machine at your site. As the IBM/4.3 system is distributed, the kernel source is located in the directory `/usr/sys`. However, the source code may be located anywhere in the file system hierarchy, as long as you create a symbolic link from the directory `/sys` to its actual location. The remainder of this chapter gives path names relative to the directory `/sys`.

Within the `/sys` directory is a directory named `conf`. It contains the configuration files you need to modify to reconfigure the kernel. One such configuration file, GENERIC, is supplied with the IBM/4.3 distribution and corresponds to the kernel, `/vmunix`, as distributed. If your kernel has already been reconfigured, you should find additional configuration files in the `/sys/conf` directory.

The illustration below shows the six main steps to generate a newly-configured kernel suitable for booting.

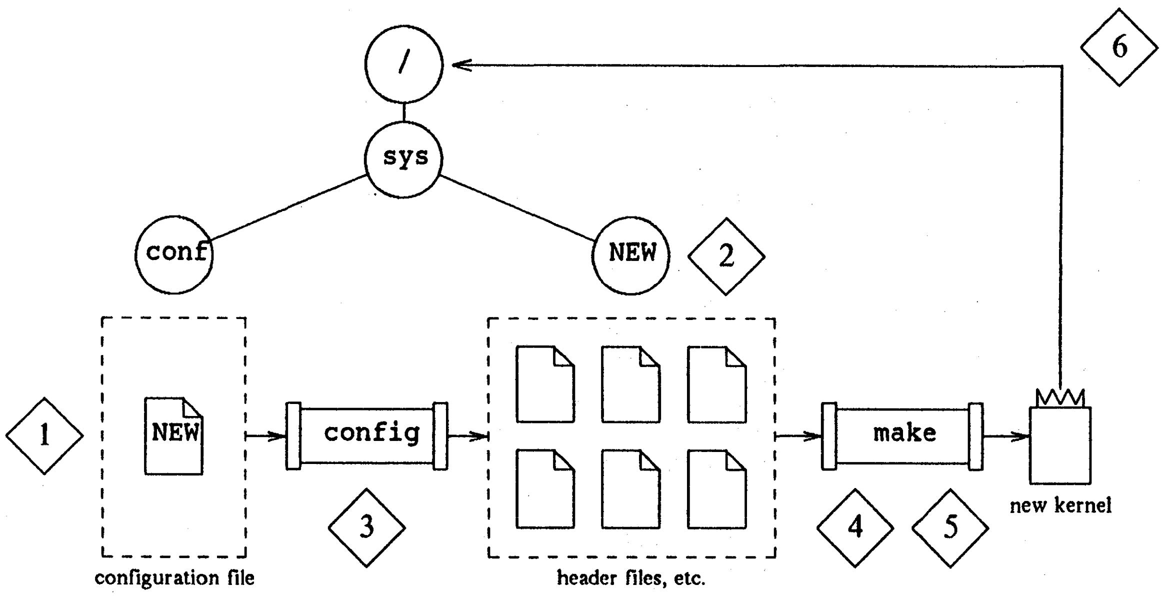


Figure 3-1: Generating a New Kernel

(1) Create a configuration file for the system to be generated. Usually you can simply copy an existing configuration file and make any necessary changes to the copy. For example, if you are reconfiguring your kernel for the first time, copy the template GENERIC and make changes to the new copy. The contents of a configuration file are described below in detail.

(2) Create a directory to hold a number of files that are used in the kernel construction process. (In the following discussion, this directory is called the "target" directory.) By convention, this directory is given the same name as the configuration file and is located in the directory `/sys`, the parent directory of the one holding the configuration files. For example, if your new configuration file is named NEW, you would need to create the directory `/sys/NEW`. As illustrated, it is a good practice to capitalize the name of your configuration file (and the names of its associated target directories). This helps isolate the configuration subdirectories in the `/sys` directory from other files and directories that may be located there.

(3) Run the utility `config` on the new configuration file. `Config` will read that file and generate a prototype `makefile` and several header files (files with names ending in `.h`) in the target directory.

(4) Construct source code dependency rules for the system being created. Move into the target directory and type:

```
# make depend
```

A modified `makefile` will be created for use in the next stage of the reconfiguration process.

(5) This step recompiles the system source code according to the directions now contained in the `makefile`. Type the command:

```
# make vmunix
```

A large number of object modules (files with names ending in `.o`) will be created in the target directory, along with a file called `vmunix`. This is the new bootable system image, or kernel.

(6) Install the newly-created system image as the bootable kernel. After saving a copy of your current `vmunix` kernel, copy the new file `vmunix` located in the target directory into the root directory, `/`.

(7) Reboot your machine to try out the new kernel.

As you can see, the process involved in creating and installing a new kernel is reasonably straightforward and automatic. You should only have to edit the configuration file, which is relatively easy to understand and modify. The next several sections describe in detail the contents and format of the configuration file processed by `config` in step 3 above.

3. Format of the Configuration File

A configuration file consists of an arbitrary number of lines, each of which begins with either a keyword, some white space (indicating that it is a continuation of the preceding line), or a pound sign, # (indicating that the line is a comment and is to be ignored by `config`). There are three kinds of parameters specified in the configuration file, usually in the following order:

- *Global configuration parameters* apply to all of the system images to be generated from this single configuration file.
- *System image parameters* are specific to each separate image to be generated from this single configuration file.
- *Device specifications* indicate the kinds of devices that either are or will be attached to your machine.

3.1. Global Configuration Parameters

The seven global configuration parameters are listed and described in the following table. For more detailed information, please see Chapter 4, "Configuration File Syntax" and Appendix C, "Sample Configuration File" of "Building IBM/4.3 Systems with Config" in Volume II of the IBM/4.3 documentation.

Keyword	Arguments	Description
`machine`	*type*	The new kernel is to run on machine type *type*. This information is used to locate certain data files that are specific to the machine type. It is also used to select rules in constructing the resulting configuration files in the target directory.
`cpu`	"*type*"	The system is to run on a central processing unit (CPU) of the specified type *type*. Multiple *types* are allowed, subject to certain constraints, in which case the new kernel will be configured to run on all of the listed CPUs. Note that the *type* must be enclosed in double quotes.
`options`	*list*	The optional sections of code in the specified *list* are included in the new kernel. For example, the optional quota subsystem is included by adding the value QUOTA in the list. Multiple options may be separated by commas, or they may each follow a separate `options` keyword.
`makeoptions`	*list*	Options may be passed to the `make` utility by including them in the specified *list*.
`timezone`	*number*	The system is to run in the specified time zone. The *number* is the number of hours that the desired time zone is west of Greenwich Mean Time (GMT). For example, Pacific Standard Time is 8 hours west of GMT.
`ident`	*name*	The new system is to be known as *name*. This name is used primarily to select machine-specific sections of code using the `#ifdef` preprocessor directive. Accordingly, *name* should be all uppercase to conform to normal `#define` syntax practices.
`maxusers`	*number*	The kernel is to accommodate a maximum of *number* users at one time. There may be many more user accounts (as specified in the user account data base, `/etc/passwd`), but only this many users may simultaneously access the system.

Table 3-1: Global Configuration Parameters

Each of these keywords must be present on a line in the configuration file except for the `options` keyword (which is optional). Most system configuration files, however, specify some options for inclusion in the system image. One widely-used option is `INET`, which specifies that the kernel include code relevant to network operation. Another common option is XWM, which requests support for the X Window System, upon which parts of the IBM Andrew File System are built.

Note that if you include certain of the available options, you may also need to specify that the system be configured to recognize particular hardware or pseudo-hardware devices. For example, if you request network support by specifying the `INET` option, then you must configure the kernel to support the network pseudo-devices `inet`, `ether`, and `pty`. The appropriate format of such specifications is discussed below in the section "Pseudo-Device Specification."

3.2. System Image Parameters

System image parameters are specified beginning with the keyword `config`. The simplest possible specification is this:

```
    config      vmunix      swap generic
```

This specification instructs `config` to follow its default rules for determining the location of the system swap space, the root partition, the device for processing argument lists, and the device to dump the system image in the case of a system crash. The current IBM/4.3 distribution establishes the default root partition in partition `a` of disk 0, while all the other default parameters are set to partition `b` of disk 0.

You can override `config`'s default behavior by specifying the system image parameters. A moderately complex `config` line might look like this:

```
    config      vmunix      root on hd0     swap on hd0 and hd1
```

The `root` clause specifies the location of the root file system; in this example, the root file system is to be found on device `hd0` (which is the default location). This line also specifies that swapping and paging activity are to occur on both `hd0` and `hd1`. (However, note that swapping and paging will not occur on the second device listed until the system administrator enables the additional areas with the `swapon` command. Usually, the line

```
    /etc/swapon -a
```

is placed into the multi-user start-up file so that the additional swapping and paging areas are enabled automatically at boot time.)

Several `config` keywords may occur in a single configuration file, indicating that multiple system images are to be generated from that file. See below for a further discussion of creating multiple images.

3.3. Device Specifications

The most complicated part of the configuration file, and usually the largest, is the specification of the hardware devices and pseudo-devices that may be attached to the machine. At boot time, the kernel goes through an autoconfiguration phase during which it tests for the existence of the hardware specified in the configuration file. To do this, the system must have some idea of which devices might be present on the system. The system is configured to look for particular hardware devices by supplying the appropriate information to `config`. Note that you can configure a system image for hardware devices that are not actually attached to a system. This flexibility allows you to generate kernels that will accommodate hardware devices that you plan to attach to the system in the future. Only the actual hardware found during the autoconfiguration phase of the boot process will be used by the system.

On the IBM/4.3 system, there are four possible device specification keywords, whose syntax is as follows:

> `controller` *name info*
> `device` *name info*
> `disk` *name info*
> `tape` *name info*

A controller is an adapter that uses DMA or controls one or more disks or tapes. The master controller is named `iocc0`, as explained below. Other controllers are attached to it, as are some other devices (such as serial output connectors). Tape and disk drives are attached to their own appropriate controllers. Aside from the root controller, each device must be attached to some other device, in accordance with the following rules:

- A controller must be attached to another controller.
- A disk or tape must be attached to a controller.
- A non-slave device must be attached to a controller.

The information included in the string *info* specifies where in the logical hierarchy of devices the device in question is attached, and hence typically begins with the keyword `at`. The logical hierarchy of device specifications mirrors the hierarchical nature of the actual physical connections, so it may be helpful to consider the following simple diagram of a machine and its attached hardware.

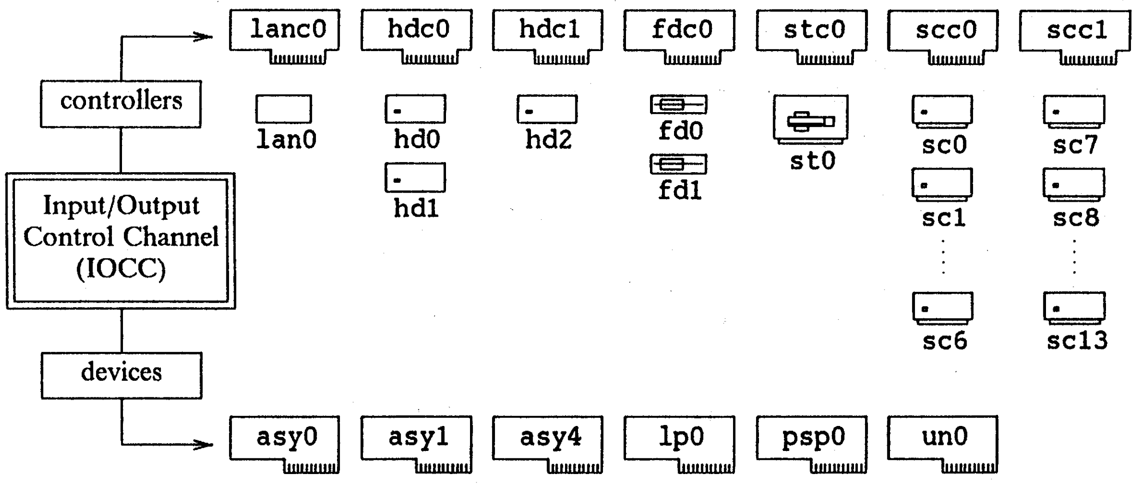


Figure 3-2: A Machine's Controllers and Devices

The top of the physical hierarchy is the input/output control channel, or IOCC, which consists of circuitry on the main system board. The IOCC is the channel by which all peripheral equipment (attached devices, controller cards, and even physical memory) communicates with the central processor. All controllers and some devices are connected directly to the IOCC,

by plugging them into an available expansion slot. Other devices are attached to the IOCC only indirectly, by attaching them to some other controller. In the diagram, there are seven controllers depicted, all of which connect directly to the IOCC. Devices may then be attached to those controllers. For example, the hard disk `hd0` is attached to the hard disk controller, `hdc0`, which resides in one of the available bus slots. Further, a streaming tape drive, `st0`, is attached to the streaming tape controller, `stc0`.

It is relatively straightforward to convert the diagram of the physical set-up into a list of device specifications. By convention, the IOCC has the following specification:

```
controller  iocc0  at nexus  ?
```

Other controllers are described by giving the name of the controller, the name of the controller to which it is attached, the controller adapter base address, and the interrupt level. For example, the first three controllers might have specifications as follows:

```
controller  lanc0 at iocc0 csr 0xf00001c0 priority 12
controller  hdc0  at iocc0 csr 0xf00001f0 priority ?
controller  hdc1  at iocc0 csr 0xf0000170 priority ?
```

The question mark ? indicates that the priority is left unspecified and is to be determined by the operating system at boot time. This is known as "wildcarding" a device specification.

Similarly, devices are specified by listing the name of the device, the name of the controller to which it is attached, the controller adapter base address, and the interrupt level. So the device `lp0` may be specified as follows:

```
    device       lp0   at iocc0 csr 0xf00003bc priority 9
```

Moreover, the devices attached to their own controllers are listed as follows:

```
    disk         hd0   at hdc0 drive 0
    tape         st0   at stc0 drive 0
```

The recommended controller adapter base address and interrupt priority level for a specific device may be obtained by consulting the appropriate manual page in Section 4 of Volume 1 of the IBM/4.3 documentation.

3.4. Wildcarding

As just mentioned, it is possible to leave certain device information unspecified in a device specification. Whatever information is lacking will be obtained, if possible, by the system as it checks its attached devices and controllers in the autoconfiguration stage of the boot process. This process of wildcarding device specifications allows you to omit device location information, thereby generating a more flexible system image.

To appreciate the value of wildcarding device location information, consider the following specifications:

```
    controller   hdc0  at iocc0 csr 0xf00001f0 priority ?
    controller   hdc1  at iocc0 csr 0xf0000170 priority ?
    disk         hd0   at hdc0 drive 0
    disk         hd1   at hdc ? drive ?
    disk         hd2   at hdc ? drive ?
```

The controller and drive number specifications have been omitted from the last two disk specifications to allow disks hd1 and hd2 to be attached to either of the two disk controllers present on the machine. In that case, you can move a disk from one controller to another, if necessary, without having to reconfigure and install a new kernel. This type of wildcarding means that the system will be able to configure itself for any of the following hardware configurations involving three disks:

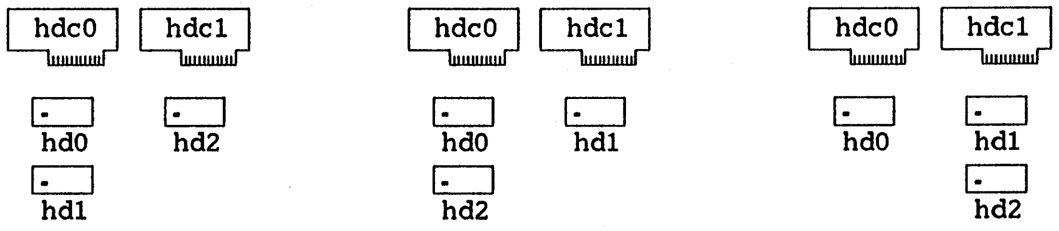


Figure 3-3: Sample Disk Connections in a Wildcarded System

Don't forget that a system can be run without all the devices specified in the configuration file. The configuration file specifies devices which *may* be present on the system. As a result, several more hardware configurations are possible by removing either hd1 or hd2.

3.5. Pseudo-Device Specifications

There are several software subsystems that require software drivers similar to the ones compiled into the kernel as device specifications for hardware controllers and devices, but for which no actual hardware exists. For example, when a user remotely logs in from one system to another using the rlogin command, a "pseudo terminal" is used on the remote host to manage the user's login session. The kernel on the remote system receives input across the network through the pseudo terminal and sends any output there. There is no actual second terminal being used, nor is there an actual physical serial port in use. Rather, the kernel accesses the user's terminal through a pseudo terminal entry in the `/dev` directory and treats the port as if a terminal were attached to it. Because such software subsystems require device drivers but do not access any actual hardware, these devices are collectively called "pseudo-devices." In addition to pseudo terminals, the kernel uses pseudo-devices to communicate across a local area network (using the `inet` and `ether` pseudo-devices) and even with itself (using the `loop` software loopback interface). Also, support for the X Window System is provided in part by configuring the kernel with the `xemul` pseudo-device.

There is a second large class of pseudo-devices that in fact *are* connected to specific hardware devices but which require special handling over and above that provided by device drivers for normal serial lines. As a result, these devices are also treated as pseudo-devices, even though, strictly speaking, they do have associated hardware devices. For example, if a mouse is attached to your machine, allowing you to point at and highlight objects on the screen, the kernel needs to be able to select the method by which it will accept data from the mouse and interpret it. Since these functions exceed those provided by the normal `tty` driver, the kernel must be configured to include the additional sections of code that govern the mouse operation. This is done by specifying the mouse as a pseudo-device. Similarly, some graphics display monitors (or screens) require specialized routines to write data and graphics in an optimal fashion. Once again, the additional code required by the kernel in order to manage the display device is included by configuring the kernel for a further pseudo-device.

The following are standard pseudo-devices available on the IBM/4.3 system:

Name	Description
`pty`	Pseudo terminals
`ms`	Mouse device
`mono`	Monochrome Display Monitor
`aed`	Academic Information Systems experimental display
`ega`	Enhanced Color Graphics Display
`apasixteen`	IBM 6155 Extended Monochrome Graphics Display
`apaeight`	IBM 6153 Advanced Monochrome Graphics Display
`apaeightc`	IBM 6154 Advanced Color Graphics Display
`inet`	Internet Protocol Suite
`ether`	ARP on Ethernet and Token-Ring
`loop`	Software Loopback Interface
`xemul`	X Window System support
`tb`	Digitizing Tablet
`ap`	IBM 3812 Pageprinter

Table 3-2: Standard Pseudo-Devices

Section 4 of Volume 1 of the IBM/4.3 documentation contains a complete list of available pseudo-devices.

In order to configure the kernel to understand how to communicate with these devices and to allocate them memory and buffer space, the configuration file must contain an entry for each type of pseudo-device. The general form of a pseudo-device specification is as follows:

`pseudo-device` *name* [*number*]

As you can see, these entries are simpler than those for normal devices, since they do not need either a base address or interrupt level specification. The only allowable argument, which is optional for most pseudo-devices, is an indication of how many of the named devices to create. For example, by default the kernel is configured to allow 32 pseudo terminals. If more of them are needed (perhaps because a particular machine will be used heavily for remote logins), then an argument may be supplied. The standard pseudo-devices on the IBM/4.3 system are configured as follows:

```
pseudo-device  pty
pseudo-device  ms
pseudo-device  mono
pseudo-device  aed
pseudo-device  ega
pseudo-device  apasixteen
pseudo-device  apaeight
pseudo-device  apaeightc
pseudo-device  inet
pseudo-device  ether
pseudo-device  loop
pseudo-device  ether
pseudo-device  xemul 4
pseudo-device  ap
```

The argument on the `xemul` entry indicates that at most four separate invocations of the X Window System may be active at a time. Note that if the `inet` pseudo-device is listed, then the `INET` option must also be listed in the options section at the beginning of the `config` file.

4. Creating Multiple System Images

As indicated earlier, the `config` utility allows several system images to be generated from a single configuration file. The multiple system images are configured for identical hardware and global configuration parameters, but they may differ in the location of the root file system or in the location of the swapping and paging devices.

To create multiple system images from a single configuration file, that file must contain multiple `config` lines, each specifying a different system name. For example, you might include the following two lines:

```
config  vmunix        root on hd0
config  swvmunix      root on hd0     swap on hd0 and hd1
```

Reconfiguration proceeds as described in the overview at the beginning of this chapter, except that in step 5 you must now give the two commands:

```
# make vmunix
# make swvmunix
```

The result is the creation of two different kernels in the target directory, named `vmunix` and `swvmunix`. You may install both of them in the root directory and then later boot the system from whichever image you choose.

5. A Sample Kernel Reconfiguration

This section takes you through a kernel reconfiguration process step by step. For clarity, we will make a very simple change to the kernel, to allow use of the optional quota subsystem. With the quota system you can impose disk space quotas on individual users and individual file systems to help ensure that the system does not run out of space. A more complete description of the operation of the quota system is contained in Chapter 14, "Quotas." Before making the changes outlined here, you should read that chapter to determine whether you should install the quota system on your machine. Even if you do not install this subsystem, it may still be helpful for you to read the following walk-through to acquaint yourself with the exact steps required to configure and install a new kernel.

To begin, move into the directory that contains the configuration files, `/sys/conf`:

```
# cd /sys/conf
```

In that directory you should see a file named GENERIC, which contains the system description for the system as distributed by IBM. To modify the file so it includes the quota subsystem, copy it and modify the copy.

```
# cp GENERIC GEN_QUO
```

The name GEN_QUO suggests the new kernel will be like the generic one, with the addition of the quota system. (You can of course choose any name you like.) Now edit the file you just created (using `vi` or an editor of your choice) and add the following line to it. Place the new line immediately after the last line beginning with the `options` keyword. (There are about six such lines.) The amount of white space between the two words is irrelevant.

```
options         QUOTA
```

Now save your changes and exit to the shell.

Next, create a directory to hold the various files used in the configuration process. As indicated above, by convention this directory should be given the same name as the new configuration file and located in `/sys`. Type:

```
# mkdir ../GEN_QUO
```

Now run the `config` program with the name of the new configuration file as an argument. Type:

```
# config GEN_QUO
```

This step should finish rather quickly, at which time `config` will print the following reminder:

```
Don't forget to run "make depend"
```

When you see this message, move into the target directory:

```
# cd ../GEN_QUO
```

Several files should now exist in that directory, including a file named `makefile` used by the program `make`. You do not need to edit any of those files; instead, continue the reconfiguration process by typing:

```
# make depend
```

This step builds rules used by `make` to recognize interdependencies in the source code. This process takes ten minutes or longer, depending on the machine you are running it on. When it completes, type the following command:

```
# make vmunix
```

At this point, the system will recompile the system source code to create a new bootable kernel that incorporates the disk quota subsystem. This process is somewhat more lengthy than the previous `make` command and may take an hour or more to complete.

When the `make vmunix` command completes, the current directory will be populated with a large number of object modules (files with names ending in `.o`) and a new kernel, named `vmunix`. This is your new bootable system image. You must now install it as the boot program, `/vmunix`, as follows:

```
# cp /vmunix /vmunix.old
# cp vmunix /vmunix
```

The first of these two commands saves your current system image under a different name, so that it can be booted if for some reason the new kernel does not operate as expected. The second command installs the kernel you just created as the bootable system image. You can see that the new `/vmunix` indeed includes routines not in the previous one by looking at the sizes of the two files:

```
# ls -l /vmunix*
-rwxr-xr-x  1 root        731136 Feb  9 11:54 vmunix
-rwxr-xr-x  1 root        720896 Feb  9 11:54 vmunix.old
```

The additional size accounts for the quota-related routines. To complete the entire process, reboot your system:

```
# sync
# sync
# reboot
```

You will be able to tell that the system is booting from your newly-configured kernel by inspecting the boot banner that appears:

```
4.3BSD UNIX(GEN_QUO) #0 Fri Feb 5 13:07:17 PST 1988
    root@master:/usr/sys/GEN_QUO
```

You have successfully reconfigured your kernel to include the disk quota subsystem. To continue the quota configuration process, refer to Chapter 14.

6. Kernel Management Tips

The kernel is the most important program in the entire IBM/4.3 system. It is therefore crucial that you exercise care in modifying your kernel and installing the modified version as your bootable system image. In addition to the various pieces of advice given throughout this chapter, the following tips will help you maintain the smooth operation of your system:

- Never build or install a new kernel that has not been successfully configured. If any error messages occur during the `config` processing, try to determine what specifications are causing trouble and then correct them.
- Always install a new kernel as `/vmunix`. Many programs available on the system expect the currently executing system image to be called `/vmunix` and are likely to give very strange (and not very useful) results if you install the kernel under a different name.
- When you install a new kernel, always save a copy of the old kernel. Then, if the new kernel fails to boot or operate as expected, you will be able to boot from the previous kernel. In particular, you should save the generic distribution version of the operating system (whose configuration file is `/sys/conf/GENERIC`) permanently as `/genvmunix` so that the system can be booted from it in case of emergency.

This page intentionally left blank.

CHAPTER 4

Managing User Accounts

1. Introduction

The IBM/4.3 system allows simultaneous access to machine resources by numerous users, who may be connected to the system directly through terminal lines and modems, or remotely through a local area network. In either case, the IBM/4.3 system controls the use of its resources with a simple scheme involving *user names*, *passwords*, and *groups*. To gain access to a particular machine, a person must be able to provide a valid user name and the correct password associated with that name. Moreover, to use certain files and peripheral devices, a user must have the correct *permissions*. By maintaining a closely-monitored list of login names and passwords and suitably determining file permissions, a system administrator can tailor access to system resources with a great deal of flexibility and control.

The tasks for managing user accounts discussed in this chapter include choosing login names and passwords, adding accounts for new users, removing old or inactive accounts, and restoring forgotten passwords. Refer to later chapters in this manual for help on related tasks such as setting permissions, backing up and restoring user files, adding terminals and peripheral devices, and establishing quotas on system disk space.

2. Overview of a User Account

A user *account* is a collection of files, directories, programs, and other items that are associated with a user name (or login name). The following diagram depicts the main files and directories relevant to a typical user account.

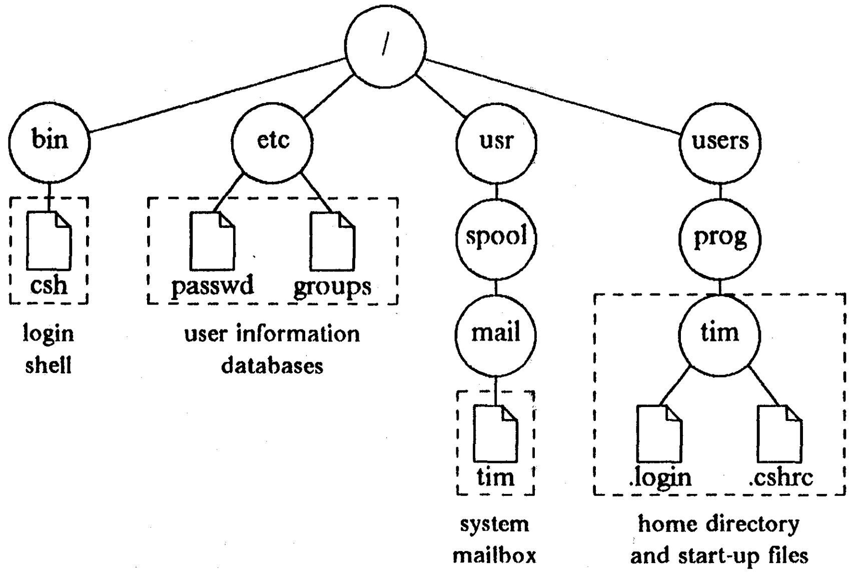


Figure 4-1: Elements of a User Account

In this example, the account is assigned to a programmer with login name `tim`. Accounts are created and removed by the system administrator, primarily by adding or deleting lines from the account data base file, `/etc/passwd`. Although many persons can share a single account, it is generally preferable to have one account per user. Then, if several persons must be given access to the same set of files, the users can be put into the same *group*. It is recommended that each person be assigned a separate account, especially if the system accounting procedures are in use and users are charged for resource consumption (CPU time, disk space, etc.).

To establish a new user account, the system administrator will need the following six pieces of information:

- login name
- password
- user identification number
- group identification number
- home directory
- login shell

The function and proper format of each of these items is explained in the following subsections.

2.1. Login Name A login name (or user name) is a string of characters that uniquely identifies a user to the operating system. The login name should be no more than 8 lowercase alphanumeric characters and is usually chosen by the system administrator in consultation with the user. Typically, the user's last name, suitably made all lowercase, is taken as the login name. For instance, a user whose last name is Smith might be given the login name `smith`. The main exception to this practice occurs when a number of related accounts need to be created for use by class members or department personnel; in such a case, a standard practice is to begin each such account name with a five-letter prefix followed by a hyphen and an arbitrary two-letter sequence. (For example, the members of a Philosophy class may be given account names of the form `philo-aa`, `philo-ab`, `philo-ac`, and so on.) Whatever naming scheme is adopted, a login name must be unique on the system. A user's current login name may be displayed by the `whoami` command.

2.2. Password A *password* is a string of characters that the user must provide during the login process to be allowed access to a particular machine. (The password is not echoed on the screen when typed by the user, so other people in the immediate area will not be able to see it.) The password should be at least six characters long and composed of a reasonably random combination of letters, digits, and special characters. This helps ensure that unauthorized users do not easily guess the passwords of other users.

2.3. User Identification Number The *user identification number* (or uid) is a numerical indication corresponding to the login name. It is used by the operating system and by certain utilities to control access to files and directories and can largely be ignored by the average user. When setting up a new user account, the system administrator must assign each new user a unique user identification number. This number must be an integer between 0 and 65535 and is typically taken to be the integer following the largest existing uid. The user identification number 0 is reserved for the superuser (whose login name is '`root`').

2.4. Group Identification Number *Groups* are used to control access to files and directories; this is useful to allow members of a project to work on a set of files without each of them having to log in using the same user name. A user must belong to at least one group, and may belong to several. The *group identification number* (or gid) is a numerical indication of the group a user initially belongs to. As with the user identification number, this number must be an integer between

0 and 65535. Traditionally, system staff (including the superuser, `root`) are given a group identification number of 10. A user may display the current group memberships by entering the `groups` command.

2.5. Home Directory Each user needs a place to store personal files and directories. The top directory allocated to a specific user is known as that user's *home directory*. In addition to personal files and directories, a number of start-up files (for example, `.login`, `.cshrc`, `.mailrc`, `.newsrc`, and similar files) will be put there. Generally, a part of some file system is dedicated to holding user home directories, so that these directories are collected together for easy backup and restoring. The location of users' home directories is, however, one of the least standardized features of the IBM/4.3 system. Sometimes part of the `/usr` file system is used, while many installations prefer to dedicate an entire file system, or even multiple file systems, to user directories. You should investigate the existing local practices, if any, to find out where on your system user home directories are located. In this guide, examples will generally assume that the file system `/users` is the root directory of user home directories.

2.6. Login Shell Once a user has provided a correct user name and password, the system launches a process for the user known as the *login shell*, which will persist until the user logs out of the system (or until the user changes the login shell). The shell interprets user commands and sends appropriate instructions to the operating system for execution. In this way, the shell lies between the user and the system (or "kernel"). Typically the login shell for a user is set to either `/bin/csh` (the C shell) or `/bin/sh` (the Bourne shell). In fact, however, practically any IBM/4.3 program can be assigned as a user's login shell; it is perfectly legal, for example, to specify the login shell as the program `dc` (in which case that user will perceive the system as a very fast and precise desk calculator). Within certain limits, a user can change the login shell without assistance from the system administrator, as illustrated below in the section "Customizing User Accounts".

3. Adding New Users

There are three major steps involved in adding a new user account to the IBM/4.3 system. First, a line must be added to the file `/etc/passwd` summarizing the necessary user information listed above. Second, a home directory must be created to give the new user a place to store personal files and directories. Third, several other files need to be created for the new user, including start-up files and a system mailbox.

3.1. Editing the Password File

The file `/etc/passwd` is the primary storehouse of account-oriented information on the IBM/4.3 system. It contains a number of lines that encapsulate the six pieces of account information listed above (one line per user or system account). The general format of a line in the file `/etc/passwd` is:

login-name : *password* : *uid* : *gid* : *user-info* : *directory* : *shell*

Format 4-1: `/etc/passwd`

There are seven fields here, each separated from the next by a colon, '`:`'. A sample line in `/etc/passwd` might therefore look like this:

```
smith:1Kd6jEOIUPoJE:341:30:Bert &,117E,8396407:/users/prog/smith:/bin/csh
```

The first field is the user's login name and the second is that user's password. Note that the password appears only in an encrypted form, so that even though users on the system can look into `/etc/passwd`, they cannot easily get a usable password from it. The fourth field contains user information; generally this includes three items separated from one another by a comma: the user's real name (where the ampersand is replaced by the capitalized user name), the user's office number, and the user's phone number. (These three subfields are used by the `finger` program.) The last two fields are the home directory and login shell for the user.

While setting up a new user account, the system administrator must edit the file `/etc/passwd` to add a line of this form. Since the password file may be updated by ordinary users (for example, using the commands `chsh` and `chfn`), it must first be *locked* before it can be edited by the system administrator. This may be accomplished using the `vipw` program. So type:

```
# vipw
```

The `vipw` command is the recommended way to make all changes to the password file for another reason too: when the password file is updated, the `vipw` command automatically runs the command `mkpasswd`. `mkpasswd` generates hashed password data base files, `/etc/passwd.pag` and `/etc/passwd.dir`, which are used by the C-language library routines `getpwnam()` and `getpwuid()`. The use of these hashed data files by the password lookup functions (instead of a linear search of the normal password file) improves response time and system performance.

Once the file `/etc/passwd` is opened for editing, add a line containing the information for a new account. This line should look just like the example given above with suitable changes made, except that *you should leave the password field empty*. There is no point in putting in a password for the new user, since `/etc/passwd` needs to contain the encrypted password. For example, you might add the following line to this file:

```
smith::341:30:Bert &,117E,8396407:/users/prog/smith:/bin/csh
```

When you are done adding a new account entry to `/etc/passwd`, save your changes and exit to the system. Then type:

```
# passwd smith
```

where '`smith`' is the login name of the new user. At this point, the `passwd` program will ask you for the new password (and have you enter it a second time, to make sure you didn't mistype).

3.2. Creating a Home Directory

Once an entry has been added to `/etc/passwd`, the system administrator must create the home directory listed in the sixth field of that entry. This is done with the `mkdir` command. For example:

```
# cd /users/prog
# mkdir smith
```

Next, you will need to pass ownership of the new directory to the new user. Type:

```
# chown smith smith
# chgrp prog smith
```

In this example, `smith` is the login name of the new user and `prog` is that user's default group (a group of programmers). For more information on the `chown` and `chgrp` commands, see the sections "Changing the Owner of a File" and "Changing the Group of a File" below. To make sure that the directory has been created and given the correct ownership and group, type:

```
# ls -ld smith
```

The output of this command should look something like this:

```
drwxr-xr-x  2 smith        512 Nov 11 21:08 smith
```

3.3. Editing the Groups File

You should now inform the system that you have added a new user to an existing group. Group membership information is stored in the file `/etc/group`. Each group is listed on a single line in the following format:

group-name : *password* : *gid* : *members*

Format 4-2: `/etc/group`

Each line is broken into four fields by the colons. The first field gives the name of the group. This can be (almost) any string, but it is usually something descriptive like '`prog`' for a group of programmers or '`doc`' for a group of documentation personnel. (The only real restriction is that group names must be 8 characters long, or fewer.) Some group names are already used for system services and administrative functions. For example, there is a group called '`wheel`' that contains the superuser, `root`, and another group called '`daemon`' that is used by daemon processes.

The second field in `/etc/group` entries is an encrypted password that, if present, is required of all users requesting to change into a group. Usually this field is left blank.

The third field is the numerical group identification number of the group. This is the number that appears in the fourth field of an entry in `/etc/passwd`.

The fourth field, finally, is a comma-separated list of the members of the group. Each user should be listed as a member of the group whose gid appears in the user's `/etc/passwd` entry, and a user may be listed as a member of up to seven other groups (for a total of up to eight group memberships). Be careful not to assign a user to more than eight groups, since otherwise that user will be unable to log in.

For example, a typical line from `/etc/group` might look like this:

```
    doc:*:204:judy,tim,tenli,dick
```

This indicates that the users having login names `judy`, `tim`, `tenli`, and `dick` belong to the `doc` group, which has a group identification number of 204. Any one of those four persons may view and alter files that belong to the group `doc`, even if that person does not own the file.

3.4. Creating Start-Up Files

The system administrator should now create several start-up files that will automatically be read by the user's login shell at login time. Which start-up files need to be created depends on which shell has been specified as the login shell in the user's entry in `/etc/passwd`. If the login shell is specified as `/bin/sh`, then a file called '`.profile`' should be created in the user's home directory. If the login shell is specified as `/bin/csh`, then several files must be created; in the remainder of this chapter, we shall assume that the login shell is `/bin/csh`.

Every installation should have prototype start-up files stored in some central location, so that the system administrator may simply copy them into the new home directory. A common place to find such prototype files is the directory `/usr/skel`. Assuming this is the case, the system administrator would execute the commands:

```
    # cd /users/prog/smith
    # cp /usr/skel/.login .login
    # cp /usr/skel/.cshrc .cshrc
    # chown smith .login .cshrc
    # chgrp prog .login .cshrc
```

When the new user first logs in, these prototype start-up files will be run by the shell. They can then be altered to suit the user's individual preferences.

If the login shell for a particular user is /bin/csh, then there is one further file that should be installed into that user's home directory, called '.logout', which is an analogue of the .login file. When that user logs out, the instructions in the .logout file, if it exists, will be executed. Usually there isn't anything very complex in this file. Here is an example:

```
# a typical .logout file
echo 'Bye Bye'
clear
if ($?prompt) stty 0
```

As before, you can simply copy a prototype into the new home directory:

```
# cp /usr/skel/.logout .logout
# chown smith .logout
# chgrp prog .logout
```

There is one further start-up file that you may want to create in the new home directory, called .hushlogin. When a user logs in, the file /etc/motd and the time of last login are usually printed on the screen. If, however, the file .hushlogin exists in the home directory, then both of those steps are skipped. This mechanism allows you to make the login process slightly "quieter" than normal and is most useful if the user logging in is a non-human user such as uucp who is not likely to be interested in such information.

3.5. Creating a Mailbox

Every user should have a place to store incoming electronic mail. This file is called the user's system "mailbox" and it is generally located in the directory /usr/spool/mail. You should move into that directory, create an empty file whose name is the user's login name, and change the ownership, group, and permissions on this file so that only the user can read any mail put there. Execute the following commands:

```
# cd /usr/spool/mail
# touch smith
# chown smith smith
# chgrp prog smith
# chmod 660 smith
```

As explained in the following chapter, the last command will prevent anyone but the user from reading that user's mail. The installation of a new user account is now complete.

3.6. Account Installation Summary

The following sequence of commands is a complete summary of the steps required to add a new user account to the system. For purposes of illustration, we shall assume that the new login name is 'smith' and that the home directory is /users/prog/smith. You should note that some installations have a script available (typically called adduser) that will perform this sequence automatically for you. If such a script is available on your system, you should use it instead of the following procedures.

(1) Select a login name, a password, a user identification number, a group identification number, a home directory, and a login shell for the new user. Find out the office number and telephone number of the new user, if appropriate.

(2) Log in as root and execute the command:

```
    # vipw
```

Add a new line to the end of /etc/passwd in the format detailed above. Do not include any entry in the password field. Write your changes to the file and quit the editor.

(3) Initialize the password field by running the command:

```
# passwd smith
```

Respond to the ensuing prompts with the password chosen in step (1) above.

(4) Execute the command:

```
# vi /etc/group
```

and add the new user name to the fourth field of the group entry whose gid was specified in the line added to /etc/passwd in step (2) above. If the new user is to be a member of any other groups, you should add the user name to the fourth field of those group entries as well.

(5) Create a new home directory for the user:

```
# cd /users/prog
# mkdir smith
# chown smith smith
# chgrp prog smith
```

(6) Create some system start-up files in the new home directory. If the login shell is /bin/csh, give the commands:

```
# cd /users/prog/smith
# cp /usr/skel/.login .login
# cp /usr/skel/.cshrc .cshrc
# cp /usr/skel/.logout .logout
# chown smith .login .cshrc .logout
# chgrp prog .login .cshrc .logout
```

If the login shell is /bin/sh, give the commands:

```
# cd /users/prog/smith
# cp /usr/skel/.profile .profile
# chown smith .profile
# chgrp prog .profile
```

(7) Create a mailbox for the new user:

```
# cd /usr/spool/mail
# touch smith
# chown smith smith
# chgrp prog smith
# chmod 660 smith
```

(8) Give the new login name and password to the new user. The new user may now log into the system.

4. Customizing User Accounts

Once an account has been created and assigned to a particular user, the user may alter some of the parameters associated with the account without further assistance from the system administrator by running the passwd command. This command is used primarily to alter the account password, but it may also be used to change the login shell (using the -s option) or the user information field entries (using the -f option). The passwd command therefore incorporates the functions previously performed by the commands chsh and chfn.

To display the current account settings, you may simply look at the relevant line from the file /etc/passwd:

```
% grep nat /etc/passwd
nat:uY79FnRa8/tOk:364:50:Nathan Daniels,a106,,:/users/prog/nat:/bin/csh
```

A more readable summary can be obtained by using the `finger` command. For example:

```
% finger nat
Login name: nat                     In real life: Nathan Daniels
Office: a106,
Directory: /users/prog/nat          Shell: /bin/csh
On since Apr 18 11:39:03 on ttyp0 from ibmpa
No Plan.
```

To alter the login shell, type:

```
% passwd -s nat
Old shell: /bin/csh
New shell: /bin/sh
```

The program listed the current login shell and requested a new one. The programs that you may provide in response to the prompt for a new shell are listed in the file `/etc/shells`. If this file does not exist, then only the two shells `/bin/csh` and `/bin/sh` may be specified.

A user may alter the user information displayed by the `finger` command by running the `passwd` command with the `-f` option:

```
% passwd -f nat
Default values are printed inside of '[]'.
To accept the default, type <return>.
To have a blank entry, type the word 'none'.

Name [Nathan Daniels]:
Room number (Exs: 597E or 197C) [a106]:
Office Phone (Ex: 6426000) []:
Home Phone (Ex: 9875432) []:
```

Notice that the command has prompted for the four items typically specified in `/etc/passwd`, the user's name, room number, and home and office phone numbers.

Finally, a user may advertise to the user community either a plan, or a project, or both, by creating one or both of the files `.plan` and `.project` in the home directory. Any text entered into these files will be displayed by future invocations of the `finger` command. Note however that a user who wants to announce plans or projects in this way must have the home directory readable and executable by everyone on the system in order for the `finger` command to be able to read the appropriate files. The mechanism for setting the access permissions in this way is discussed in the next chapter.

5. Removing Users

It will sometimes become necessary to restrict certain users from accessing machine resources or to remove their user accounts altogether. This section explains the steps you will need to take in order to accomplish these tasks. Before limiting or denying machine access for a particular user, however, you should first determine the precise nature of the access restriction. The complete deletion of a user from a system involves both removing the appropriate user entry in `/etc/passwd` and removing all files and directories owned by the user. If you want merely to suspend account access temporarily, you simply need to modify the password field of the user entry in `/etc/passwd`. These procedures are detailed in the following two sections.

5.1. Suspending an Account

A user account may need to be suspended temporarily for a wide variety of reasons, ranging from preventing misuse of machine resources to enforcing a freeze on further development of a project.

For whatever the reason, the operating system can be instructed to disallow logins for a specific account simply by altering the password field in the appropriate line in /etc/passwd. Most commonly, the encrypted password is replaced by the word 'VOID' or by some other impossible password such as the string 'XXX'. Once this is done (using the vipw command), the user will be unable to log into the system.

The access restriction can be lifted by reversing this process: remove the word 'VOID' from the password field (leaving it empty) and then immediately run the passwd command to reinitialize the user's password.

5.2. Deleting an Account

The first thing that a system administrator should do to permanently remove a user from the system is to make a complete backup copy of all files and directories owned by the user. Even if you are certain that the user is done with those files, it is remotely possible that they will be needed again in the future by someone else, or that the user account will need to be reconstructed. You should also transfer ownership of any files that are currently in use by other members of the system. It is highly recommended that the administrator send mail to all users informing them of the impending account deletion and requesting a response from anyone using files or directories belonging to that user.

You can obtain a list of all files and directories in the directory hierarchy owned by a specific user (for example, smith) by executing the following command:

```
# find / -user smith -print
```

Most of the files in the list will probably be located under the user's home directory, although there may be a number of files scattered elsewhere in the directory structure (such as the user's mail box in /usr/spool/mail). You may remove all files and directories owned by a specific user (once again, smith) by executing the command:

```
# find / -user smith -exec rm {}\;
```

Be sure to run this command only after you have completely backed up the files and directories of the account you are removing.

Next, you will need to remove the line in the file /etc/passwd that lists the account information for the user being deleted from the system. As before, use the command vipw to edit that file. Then edit the file /etc/groups to remove the user's login name from the fourth field of any line that contains it. The user is now completely removed from the system.

5.3. Account Removal Summary

The following sequence of steps will allow you to remove a user account from the system.

(1) Back up all files and directories owned by the user whose account you are deleting.

(2) Find and remove all files and directories owned by the user:

```
# find / -user smith -exec rm {}\;
```

(3) Execute the command:

```
# vipw
```

and remove the line containing the account information for the user you are deleting.

(4) Execute the command:

```
# vi /etc/groups
```

and remove all references to the user being deleted.

6. System Accounts

As distributed, the IBM/4.3 system already contains a number of accounts that are designed for use by the system itself, by utilities running in the system, or by the superuser and other administrative personnel. For example, the login account `root` is provided for use by the superuser to perform system maintenance tasks, and the account `uucp` is provided for use by the uucp system. Generally you should not alter either the user identification numbers or the group identification numbers of such accounts, nor should you alter the group memberships of those accounts. If you change uid's and gid's of system accounts in a haphazard manner, some utilities will no longer function correctly.

As distributed, the IBM/4.3 system includes entries for the following accounts:

Name	Uid	Description
`root`	0	the superuser account
`daemon`	1	account for daemon processes
`operator`	2	account for file system backups
`uucp`	66	account for uucp transactions
`nobody`	32767	the user with least privileges

Table 4-1: Predefined IBM/4.3 System Accounts

Both the `daemon` and the `operator` accounts are provided to allow the system or a user to perform tasks that require some enhanced privileges but that do not need full `root` privileges. For instance, anyone logged in as `operator` has full read permissions on all file systems. The `operator` account is intended to allow a user to perform file system backups and retrievals without having full read and write privileges on the file systems. The `nobody` account is an account with the least possible file system privileges; it is useful to provide a minimally dangerous uid argument to some of the entries in `/usr/lib/crontab`. See the chapter "Periodic Command Execution" for full details. Incidentally, the strange-looking user identification number for the `nobody` account is simply the uid midway between the smallest (0) and the largest (65535) possible uid values.

The following groups are also pre-defined in the IBM/4.3 system:

Name	Gid	Description
`wheel`	0	only users in this group may `su` to `root`
`daemon`	1	owns files in spool directories
`kmem`	2	only this group can read `/dev/kmem` and `/dev/mem`
`sys`	3	owner of system source code
`tty`	4	owner of user terminals
`operator`	5	has read access to disks
`staff`	10	owner of non-system source code

Table 4-2: Predefined IBM/4.3 System Groups

Once again, you should take care not to alter these values, lest certain utilities break.

7. Additional System Accounts

You may wish to insert additional entries into `/etc/passwd` and `/etc/group` in order to allow users to perform some simple tasks without having to know the superuser password or having full user accounts on a machine. For instance, it is useful to have an account named '`sync`' so that a user (perhaps even the system administrator) can synchronize the disks without having to go through the entire log in process. To accomplish this, you can add the following line to the file `/etc/passwd`:

```
sync::7:10:synchronize the file systems:/:/bin/sync
```

(You may need to change the uid and gid to conform to local customs.) Similarly, it is very often useful to have a login account named '`who`' so that users can log in just long enough to see who is currently on the system. To do this, add the following line to the file `/etc/passwd`:

```
who::7:10:show who is logged in:/tmp/:/bin/who
```

(Once again, you may need to change the uid and gid to conform to local customs.) When a user logs in as '`who`', no password will be requested and the current users will be listed. Then the system will log that user off and print a new login banner.

8. Restoring a Lost Password

Users sometimes forget passwords to their accounts, so a system administrator will need to know how to restore a password. The process is quite simple: invoke the command `passwd` with the user's login name specified as an argument. Since you are the superuser, you will not be asked for the previous password. Instead, you will be asked for a new one, and then asked to retype it. You have now reinitialized the user's password.

This page intentionally left blank.

CHAPTER 5

File and Directory Permissions

1. Introduction

The IBM/4.3 system controls access to files and directories by maintaining a set of *permissions* for each file or directory in the file system hierarchy. The permissions of a file or directory determine who on the system is able to do what with the file or directory. If a file is an ordinary text file, the permissions determine who can look at (or read) and modify (or write) the file. If a file is a program or executable shell script, the permissions determine who in the system is able to run that program or script. By the nature of the tasks assigned to a system administrator, the superuser is able to look at and modify *any* file or directory in the system; in addition, the superuser can execute any program or shell script. An ordinary user, however, may look at or alter only those files and directories for which he has the correct permissions; if the permissions on a file or directory are not set appropriately, then the user will be unable to access that file or directory.

File and directory permissions therefore provide a general mechanism whereby a user can share files with other users or protect them from reading and tampering by other users. They also provide a means by which the system administrator can prevent unauthorized users from executing commands that would disrupt the system (such as unmounting a file system) and from modifying files that are necessary for the system to function properly. Permissions are an important component of system security and it is essential that a system administrator understand the operation of file and directory permissions thoroughly. This chapter discusses the permissions that can be assigned to files and directories under the IBM/4.3 system, as well as several other topics related to the permissions mechanism.

2. Types of Permissions

There are three types of permissions assigned to a file or directory at the time of its creation, called *user* permissions, *group* permissions, and *other* (or *world*) permissions. User permissions specify what actions the *owner* of a file may undertake. Group permissions specify what actions a person who is in the same *group* as the owner of the file may take, whether or not that person actually owns the file. Finally, world permissions specify what actions everyone else who has access to the system may undertake. These three types of permissions may be set independently of one another.

For the most part, it is best to let individual users determine the permissions on the files and directories they own. It is crucial, however, that system files and directories have their permissions and ownerships set correctly. If certain files in the system do not have the correct permissions, ordinary users may be able to acquire superuser privileges and do unfriendly things to your system. Consult the chapter on system security for a more complete discussion of the issues involved.

3. Looking at Permissions

The information about the permissions governing a file or directory is stored in the corresponding i-node, along with information about the owner of that file or directory its date of creation, and so on. To see what permissions are set for the members of a particular directory, move to the directory and execute the command:

```
    # ls -l
```

This instructs the system to produce a "long" listing of the files and subdirectories in that directory. The output will look something like this:

```
    total 18
    drwxr-xr-x  2 monroe           512 Nov 11 21:08 Figs/
    -rw-r--r--  1 monroe           315 Nov  4 20:35 README
    -rw-r--r--  1 monroe          1621 Nov  3 20:47 ch.01
    -rw-r--r--  1 monroe           121 Nov  3 19:31 ch.02
    -rw-r--r--  1 monroe           285 Nov  3 19:27 ch.03
    -rw-r--r--  1 monroe          6527 Nov 15 17:07 ch.04
    -rw-r--r--  1 monroe          6262 Nov 11 18:17 ch.05
    -rw-r--r--  1 monroe           627 Nov  3 20:35 ch.06
    -rw-r--r--  1 monroe          1471 Nov  3 21:07 ch.07
    -rw-r--r--  1 monroe           188 Nov  3 19:29 ch.08
    -rw-r--r--  1 monroe           148 Nov  3 19:29 ch.09
    -rw-r--r--  1 monroe           128 Nov  3 19:30 ch.10
```

The first line of the output indicates the number of disk blocks occupied by all the files and subdirectories in the listed directory. The first 10 characters of each subsequent line in this output indicate the permissions for each file. The ten characters are subdivided into four fields, interpreted as follows:

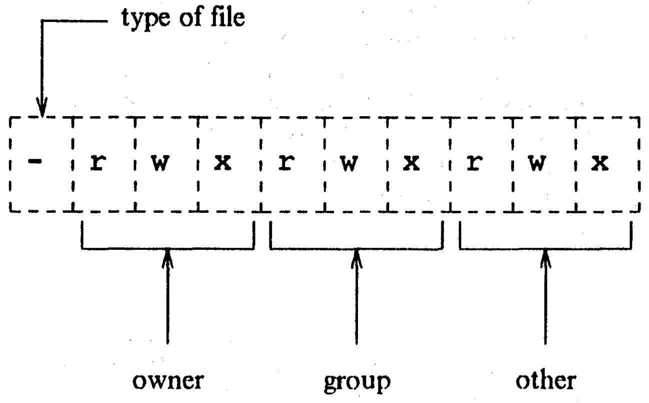


Figure 5-1: Interpreting the Permissions Fields

The first column indicates the "type" of the directory entry and may have any one of the following four characters:

d This indicates that the item listed is itself a *directory*.

- This indicates that the item listed is an ordinary *file*.

l This indicates that the item listed is a *symbolic link* to a file located elsewhere in the directory hierarchy, possibly on a different file system.

s This indicates that the item listed is a *socket*.

b This indicates that the item listed is a *block* special file.

c This indicates that the item listed is a *character* special file.

The last two file types typically reside in the directory of special files, /dev, and are not used for files or directories located elsewhere in the file system. For administering ordinary user files and directories, you can ignore these two file types.

The three remaining fields in the permissions columns indicate, for the owner, the group of the owner, and everyone else, whether the file or directory is *readable*, *writable*, or *executable* by that person. The meanings of these terms differ slightly depending on whether they apply to a file or a directory.

3.1. File Permissions

For *files*, the characters in the permissions fields have the following meaning:

`r` If the first character in the owner, group, or other field is the letter '`r`', then that file is readable by the appropriate users. Users with read permission may look at the file (for instance, using the `cat` command). They may also copy the file.

`w` If the second character in the owner, group, or other field is the letter '`w`', then that file is writable (or changeable) by that user. Users with write permission may edit or even remove the file.

`x` If the third character in the owner, group, or other field is the letter '`x`', then the file is executable by that user. In general, files are made executable only if they are programs or shell scripts.

`s` If the third character in the owner field is '`s`', then the file is an executable file (binary program or shell script) that has been configured to run with the effective uid of the owner of the file. If the third character in the group field is '`s`', then the file is an executable file that has been configured to run with the effective gid of the group of the file. For more information on `setuid` or `setgid` files, see below.

`t` If the last character in the permissions fields (i.e., the third character in the *other* field) is '`t`', then that file has its "sticky bit" set. If the file is an executable program, this indicates that the text portion of the program will remain in the swap area so that it can be located quickly in case the program is executed again.

`-` If the character in a position in the owner, group, or other field is '`-`', then the corresponding permission is not active for that user.

For example, a file with the permissions

```
-rwxrwxrwx
```

is readable, writable, and executable by everyone who has access to the directory in which the file is stored. (This does not quite mean that just anybody could remove the file or change it, since the permissions of the directory in which the file is located may be set so as to restrict others from entering it.) Similarly, a file with the permissions:

```
-rwx------
```

is readable, writable, and executable by only the owner of the file.

3.2. Directory Permissions

For *directories*, the characters in the permissions fields have the following meaning:

`r` If the first character in the owner, group, or other field is the letter '`r`', then that directory is readable by the appropriate users. Users with directory read permission may list the files in that directory (for instance, using the `ls` command).

`w` If the second character in the owner, group, or other field is the letter '`w`', then that directory is writable by that user. Users with write permission in a directory may create files and subdirectories in that directory. They may also rename files in that directory (for instance, using the `mv` command) and even *remove* files from that directory.

x If the third character in the owner, group, or other field is the letter 'x', then the directory is executable by that user. This means that a user may enter that directory (for instance, with the cd command) and look at the contents of files.

t If the last character in the permissions fields (i.e., the third character in the *other* field) is 't', then that directory has its "sticky bit" set. This indicates that the directory is an "append-only" directory. To remove or rename a file located within such a directory, a user must have write permission on the directory and be the owner of the file, regardless of whether the file itself allows write permission to other users. (The superuser may also remove or rename such files.)

- If the character in a position in the owner, group, or other field is '-', then the corresponding permission is not active for that user.

For example, a directory that has the permissions:

```
drwxrwxrwx
```

can be looked into, written into, and moved into by anyone on the system. On the other hand, if the permissions for a directory are:

```
drwxr--r--
```

then the owner of the directory may do anything to it, while the group and other users can only list the contents of the directory. In particular, users other than the owner cannot move into that directory or create files or directories in it.

4. Changing Permissions

You may change the permissions of a file or directory by using the chmod (change mode) command. For example, the command:

```
# chmod go-rwx /src
```

will prohibit anyone other than the owner (who is probably root) from accessing any files and subdirectories located under the directory /src. Note that only the owner of a file or the superuser may change the mode of a file.

The first argument to the chmod command may be specified in two different ways, "symbolically" or "absolutely". When the new mode is expressed "symbolically" (as illustrated above), the first argument has three parts: *who* is affected by the change, the *operation* to be performed, and the *permission* to be altered. The available values for these parts are as follows:

who may be either 'u' for the user or owner of the file, 'g' for anyone in the same group as the owner of the file, and 'o' for everyone else. The special abbreviation 'a' is available, standing for 'ugo'.

operation
is '-' if the specified permissions are to be removed for the users indicated, '+' if they are to be added, and '=' if the permissions are to be set to the listed permissions, clearing all unlisted permissions.

permission
may be any combination of 'r' for read permission, 'w' for write permission, 'x' for execute permission, 's' for set owner or group id, and 't' for sticky bit set. The permission 'X' may also be specified instead of 'x' to set execute permission only if the file is a directory or if some other execute bit is already set.

Alternatively, the permissions may be set absolutely using the following numerical scheme:

4000	set user id on execution
2000	set group id on execution
1000	set sticky bit
0400	read by owner
0200	write by owner
0100	execute by owner
0040	read by group
0020	write by group
0010	execute by group
0004	read by others
0002	write by others
0001	execute by others

Table 5-1: Determining Permissions Numerically

Octal 4 indicates that the specified person has read permission on the file or directory in question, while 2 indicates write permission and 1 indicates execute permission. For example, the command:

```
    # chmod 700 /src
```

sets the protection modes on the directory `/src` to octal 700, which indicates that the directory may be read by the owner, written into by the owner, and searched by the owner (400+200+100), and may not be accessed by any other person (except the superuser). So this command has the same effect as the one specified symbolically above.

When a mode is specified symbolically, several useful effects may be obtained by omitting part of the symbolic specification. For example, the command:

```
    # chmod go= /src
```

removes any permissions which may have existed on the directory `/src` for everyone other than the owner of that directory. The command:

```
    # chmod +X /bin/passwd
```

makes the `passwd` program executable by everyone on the system if it is currently executable by anyone.

There is a **-R** option available to the `chmod` command that instructs `chmod` to recursively descend any directories listed as arguments and set the mode as specified. For example, if the `/src` directory contains any subdirectories (as is quite likely), then the command:

```
    # chmod -R go-rwx /src
```

will cause the entire directory structure under `/src` to become unavailable to anyone other than its owner.

5. Changing the Owner of a File

By default, the owner of a file is the user who creates it. The creation may occur by invoking an editor and adding some text to a file, by compiling some source code into an executable program, by copying an existing file, or by many other means. In whatever manner a file is created, the operating system inspects the user identification number of the creator to determine the owner of the new file.

The superuser may change the owner of a file or directory by using the `chown` (change owner) command. For example, to change the owner of the file `data.1` from `smith` to `root`, issue the command:

```
    # chown root data.1
```

The new owner of the file or directory may also be specified by the uid. So the previous example could have been written:

```
    # chown 0 data.1
```

Only the superuser may execute the `chown` command. This restriction simplifies certain accounting procedures and prevents users from circumventing disk space quotas (if the quota subsystem is in effect). See `chown`(8) for more complete details.

6. Changing the Group of a File

The group membership of a file or directory is taken to be the group membership of the directory in which the file or directory is created. Recall that under the IBM/4.3 system, a user may belong to several groups at once. For example, a user may belong to both the `prog` and `doc` group (programming and documentation, respectively). If this user has a subdirectory owned by `doc`, then all files and directories created within that directory will belong to the group `doc`. Similarly, if this user has a subdirectory owned by `prog`, then all files and directories created within it will belong to the group `prog`. Note, however, that simply moving a file from a directory belonging to one group into a directory owned by another group will *not* automatically change the group membership of the file. To change the group membership of a file from the default membership given at the time of creation, you must explicitly instruct the system to do so.

You may change the group of a file or directory by using the `chgrp` (change group) command. This command works just like the `chown` command, except that you need to give it a valid group name as the first argument. For example, to assign a file to the `prog` group, type:

```
    # chgrp prog data.1
```

The user invoking the `chgrp` command must be a member of the group into which the file or directory is to be moved. The only exception to this rule is the superuser, who may move any file into any group at all.

Note that the group membership of a file may be changed at the same time as the ownership by providing an optional group identification number or group name to the `chown` command. For example, to transfer ownership of a file owned by `root` to a normal user, the superuser might give a command like:

```
    # chown nat.prog data.1
```

Upon successful completion of this command, the file `data.1` will belong to the user `nat` and the group `prog`. See `chgrp`(1) and `chown`(8) for more complete details.

7. Setuid Programs

At times, it is necessary to allow users to modify certain files and directories without allowing them to have full read and write privileges on those files and directories. For example, it is important to allow users to change their passwords often in order to promote security and reduce unauthorized use of the system. But it is a bad idea to let everyone have full read and write permissions on the password file, `/etc/passwd`, since by suitably altering a few numbers, such users could give themselves superuser privileges. The IBM/4.3 system solves these conflicting desires by allowing certain programs to be run by anyone on the system, while temporarily adopting the permissions of the owner of the program. Such programs accomplish this by setting the effective user identification number to the owner of the file; for brevity, such programs are said run to "`setuid`".

The effective user identification number of a process is the principal operative factor involved in determining file access permissions. When a process asks to perform some operation on a file, the system ensures that that operation can be performed by inspecting the effective user id of the process and comparing it to the access permissions of the file. If the effective uid is that of the superuser or of a user allowed access permission to the file, then the requested operation is performed. Initially, the effective user id is identical to the user's real user id (as specified in /etc/passwd), but it may be changed temporarily if the program is setuid. In a nutshell, the setuid mechanism is simply a way to change from the user's real uid to an effective uid.

Since the effective user id of a user of a program is governed by a bit in the permissions field, a program that runs setuid to the owner of the file is also said to have its setuid bit set. This is accomplished by means of the chmod command. For example, to set the setuid bit on the file /bin/passwd, execute the following command:

```
# chmod 4755 /bin/passwd
```

You can determine that a program is setuid by inspecting its permissions:

```
# ls -l /bin/passwd
-rwsr-xr-x  3 root          27648 Jul   7  1987 /bin/passwd
```

The permission 's' in the user's execute column (where an 'x' would otherwise appear) indicates that the program is indeed setuid to the owner of the program, in this case root. Since the passwd program has its setuid bit on, it may be executed by anyone with an account on the system; but that person will be assume superuser privileges *only for the duration of the program.* This is what allows the passwd program to update the file /etc/passwd even though ordinary users cannot write that file.

Another useful application of the setuid bit involves multi-user games that maintain a common data base listing scores or other necessary information. A game program that needs to update this list might run setuid to the owner of the program, so that other users would not be able to change the list except by playing the game and having the game change the list.

8. Setgid Programs

There are also programs in the IBM/4.3 system that are configured to execute with the effective *group* id of the program. Such programs are called "setgid" programs and are useful for allowing controlled access to files owned by a particular group. For example, under the IBM/4.3 system, a terminal device is owned by the user logged in it but belongs to group tty. In addition, such terminal files are writable only by the owner of the file and members of the group tty, as illustrated by the following command output:

```
# ls -lg /dev/ttyl
crw--w----  1 monroe   tty        6,    0 May 15 11:39 /dev/ttyl
```

Since normal users are typically not members of the group tty, they will not be able to write on the terminal of another user. This prevents an obnoxious user from scrambling the terminal screens of other users by redirecting command output into their terminal files.[1]

Some commands, however, are designed specifically to facilitate user-to-user communication by allowing two or more users to write on the terminal of the other(s) and carry on an interactive "conversation". To accomplish this, such programs are configured with the setgid bit set on. For example, programs like talk and write run setgid tty. You can see this, once again, by inspecting the permissions on these programs; for example:

[1]This protection also helps circumvent at least one major security loophole on systems that do not incorporate it. Some intelligent terminals can be instructed to echo characters back to the host operating system, as if they had been typed on the keyboard itself. A suitably obnoxious user who knows a little bit about how your terminal operates could instruct *your* terminal to issue a command like '/bin/rm -rf *', thereby removing all of your files! By assigning all logged-in terminal lines to the group tty, however, and restricting write access to such files, the IBM/4.3 system effectively prevents such potential mayhem.

```
# ls -l /bin/write
-rwxr-sr-x  1 root     tty         34816 Aug 11  1987 /bin/write
```

Notice that the group execute field contains an 's', indicating that the program is `setgid tty`. A user who desires to write on the screen of another user can execute the `write` command, which sets that user's effective group id to the group `tty` for the duration of the process.

While programs that run `setuid` or `setgid` perform important functions in the IBM/4.3 system, they are also a source of great concern for security reasons. If, for example, a `setuid root` program allows the user to spawn a sub-shell (as some editors do), then the user would be able to acquire superuser status simply by running the program and spawning a subshell. More directly, if one of the shell programs were accidently installed `setuid root`, then any user running that shell would automatically inherit superuser powers. The system administrator must therefore exercise special care in setting the `setuid` or `setgid` bit on user-accessible programs. Refer to Chapter 18, "Security", for a more complete discussion of the security problems associated with `setuid` and `setgid` programs and some useful ways of tracking down potential problems.

9. Sticky Bits

There is one further feature of the IBM/4.3 system that is connected with the permissions mechanism of files and directories, called the "sticky bit". The sticky bit was originally devised as a way to improve system response in launching large executable binary files and it was only recently extended to apply also to directories. Setting the sticky bit of an executable file actually has very little in common with setting the sticky bit on a directory, except that both operations have the effect of making something persist, or "stick around", longer than would otherwise happen. The two different functions of the sticky bit are discussed in the following two subsections.

9.1. Sticky Executable Files

An executable file normally consists of three principal parts, called the text segment, the initialized data segment, and the uninitialized data segment (or bss). There is also a short header at the beginning of the file and, if the file has not been stripped, some relocation information, a symbol table and a string table at the end of the file. When the program is executed, the text segment and the two data segments are copied from the file system into memory. If the file (i.e., program) is large, this can take a relatively long time.

Generally, the text segment of an executable file can be shared by different invocations of the file, thereby reducing the amount of memory consumed by users of the program. In this case, the text segment is said to be sharable. The IBM/4.3 system provides a mechanism that allows the system to save a copy of the text segment of a sharable executable file. If the sticky bit is set on a sharable executable file, then the text segment of that file will not be removed from the system swap area, even though the executable program has ceased running. This allows the system to find the text segment quickly without having to traverse the file system, thereby improving response time on future executions of that command. The text segment will remain in the swap area forever, or at least until the operating system is rebooted.

As distributed, the IBM/4.3 system includes just one executable file that has its sticky bit set, namely the screen editor `vi` (and its links, `e`, `ed`, `ex`, `edit`, and `view`). You can list all the files on your system that have the sticky bit on by executing the following command:

```
# find / -perm -1000 -exec ls -l {} \;
```

To set or unset the sticky bit, use the `chmod` program. For example:

```
# chmod +t /usr/ucb/vi
```

will set the sticky bit on the `vi` command if for some reason it is not already set. Care must be exercised in setting the sticky bit on executable files, however, since indiscriminate setting of the sticky bit may cause the system to run out of swap space and therefore crash. Most sites restrict the use of the sticky bit to programs like `vi` which have a relatively large text segment and which are used by large portions of the user community. Fortunately, only the superuser is allowed to set the sticky bit on a sharable executable file.

9.2. Sticky Directories

Recently, the function of the sticky bit has been extended so that directories can now be made sticky. A directory whose sticky bit is set is called an "append-only" directory. More accurately, a sticky directory is one in which deletion of files is restricted: a file located in a sticky directory can be removed or renamed only by the owner of the file (who must of course also have write permission in that directory), by the owner of the parent directory, or by the superuser. For instance, if the permissions for a directory are:

```
drwxrwxrwt
```

then anyone on the system can read and write files in the directory, move into the directory, and create subdirectories within it. However, normal users will be unable to remove or rename any files with that directory that they do not own, since the sticky bit is set.

Any user may create a sticky directory by first creating the directory (for example, with `mkdir`) and then altering its protection modes (using `chmod +t`). For system-wide directories, the sticky bit must be set by `root`. It is most useful to set the sticky bit on directories such as `/tmp` and `/usr/tmp` which must be publicly writable but in which users should not be allowed to remove or rename files that do not belong to them.

This page intentionally left blank.

CHAPTER 6

Managing Terminals and Modems

1. Introduction

When the IBM/4.3 system is operating in multi-user mode, it is able to manage login sessions and run processes for many users at the same time. Often these users will communicate with the system by giving commands at a terminal connected to the machine directly through a serial line. It is also common for users to gain access a machine indirectly through a modem that is communicating with another modem connected to the machine. These possibilities are depicted in the following diagram:

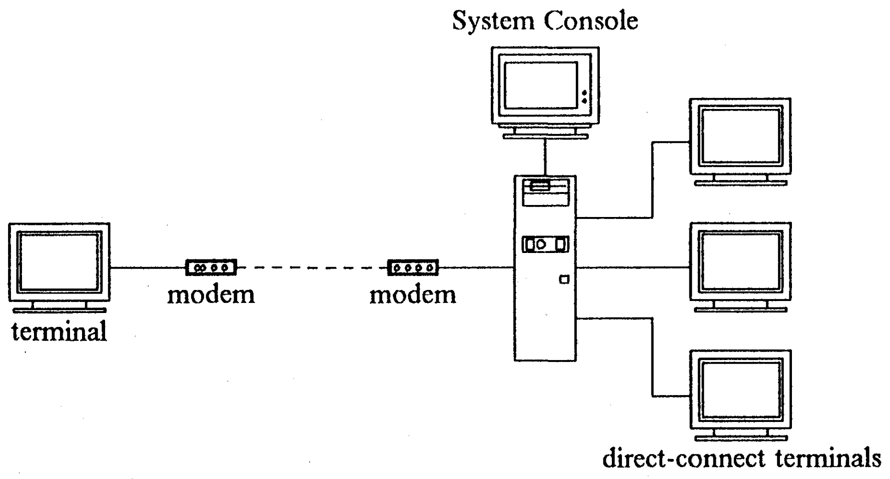


Figure 6-1: Communicating Over Serial Lines

It is easy to add terminals and modems to the IBM/4.3 system in order to allow multiple users to access machine resources simultaneously. The process consists of two main stages, the hardware installation and the software configuration. First, a serial cable attached to the peripheral device (terminal or modem) must be physically connected to a serial port that the operating system "knows" about. If the cable is not attached to a planar serial port, an expansion card that contains additional ports must be added to the machine. It may be necessary to reconfigure the kernel to recognize the additional hardware. Then, the system must be instructed to allow users to login on the appropriate serial port by editing several configuration files. Once these steps have been successfully accomplished and the system is launched into multi-user mode, a login banner will appear on the terminal screen and users will be able to login through that port.

It is also possible for users to login to a machine across a network. In that case, the process is even simpler, since no additional physical connection needs to be made. Once the kernel and software configuration is accomplished, a user may remotely login to a machine using either the `rlogin` or `rsh` command.

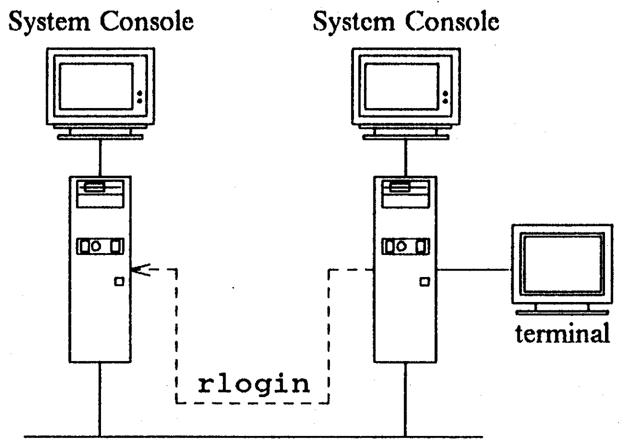


Figure 6-2: Communicating Over A Network

If you plan to allow remote access to your machine, you should refer to Chapter 11 on local area networks for instructions on installing and configuring a network. Then continue with the sections in this chapter on kernel reconfiguration, creating pseudo terminals, and setting up logins.

2. Kernel Configuration

The IBM/4.3 system can support terminals connected to the machine either directly through a serial port or indirectly through a network connection using pseudo terminals. In either case, however, the kernel must be configured to recognize the appropriate hardware or pseudo-hardware. Depending on the machine on which the IBM/4.3 system is running, there will be either one or two planar serial ports. Additional serial ports may be obtained by installing either a multi-port asynchronous communications adapter card (which contains four independent channels) or a serial/parallel adapter card (which contains a single serial port). The allowable number and proper bus placement of these cards is hardware-specific, and you should refer to the installation instructions for this information. On the IBM RT PC, for example, up to four multi-port cards may be installed, along with up to two serial/parallel adapter cards. Together with the two planar serial ports, these provide a total of 20 serial ports, as depicted in the following diagram:

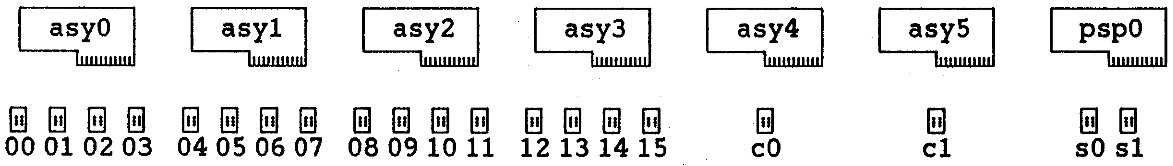


Figure 6-3: Serial Ports on the IBM RT PC

The name of each port is listed in an abbreviated form underneath the corresponding icon; the complete name is formed by prefixing those two characters with the base name '`/dev/tty`'. The four serial channels on the card `asy2`, for example, are named:

```
/dev/tty08
/dev/tty09
/dev/tty10
/dev/tty11
```

Note that the device names for the two planar serial port can be confusing. The device /dev/ttys0 corresponds to the serial port labeled 'S1' on the back of the RT, while device /dev/ttys1 corresponds to the serial port labeled 'S2'.

For the operating system to recognize any of these devices at boot time, the kernel must be configured to probe for it. This configuration is accomplished, as you have already seen, by including the appropriate lines in the kernel configuration file and then regenerating and installing a new kernel. The generic kernel on the IBM RT PC, for example, recognizes three multi-port asynchronous communications adapter cards (asy0, asy1, and asy4) one serial/parallel adapter card, and the two planar serial ports. The generic kernel therefore can support up to 15 serial ports. If you need more than this, or if you wish to locate the cards at different addresses, you will need to reconfigure the kernel. Refer to the discussion of Chapter 3 for complete instructions on doing this.

As noted above, if network logins are to be allowed, the kernel must also be configured to support pseudo terminals and the INET option must be supplied. The generic kernel supports network activity and up to 32 pseudo terminals.

3. Creating Special Files

Once the necessary hardware adapter cards have been installed into the machine and the kernel has been reconfigured to recognize them, you will need to create some *special files* in the /dev directory. The kernel communicates with serial devices by writing to and reading from its corresponding special file. A long listing of some of the special files recognized by the generic kernel is as follows:

```
crw-------  1 root       1,   0 Jan  9 22:18 tty00
crw-------  1 root       1,   1 Jan  9 22:18 tty01
crw-------  1 root       1,   2 Jan  9 22:18 tty02
crw-------  1 root       1,   3 Jan  9 22:18 tty03
crw-------  1 root       1,  16 Jan  9 22:18 ttyc0
crw-------  1 root       1,  20 Jan  9 22:18 ttyc1
crw-rw-rw-  1 root      12,   0 Jan  9 22:18 ttys0
crw-rw-rw-  1 root      12,   1 Jan  9 22:18 ttys1
```

You will notice that there is one special file for each serial port. More importantly, instead of giving a byte-count indicating the size of the file, this listing instead shows the *major device number* and the *minor device number* of the special file. Part of the reason for this is that these special files are all zero length, so there would be very little point in indicating that. More importantly, the kernel distinguishes the various serial devices from one another by looking at the major and minor device numbers. When, for example, the kernel wants to write data to a special file, it needs to access a device driver in order to know exactly how to do that. The device driver code is compiled into the kernel. The major device number indicates what kind of peripheral device the special file corresponds to and hence functions as an index into the kernel code driving the devices. In the examples listed above, both the ports on the multi-port adapter card and the serial/parallel adapter card have a major device number of 1, indicating that the same device driver is to be used in each case. The planar serial ports have a different major device number, primarily because those ports support both asynchronous and synchronous serial communication, while the multi-port card and the serial/parallel card supports only asynchronous communication.

The minor number is passed to the device driver as an argument and is usually used to allow the system to distinguish among several devices of the same type. Typically the minor number allows the kernel to address multiple ports on a single adapter card, or ports on multiple adapter cards.

Special files are created with a script, MAKEDEV, located in the /dev directory. MAKEDEV is provided largely to simplify device creation by insulating the system administrator from the mundane details of figuring out major and minor device numbers (which are specific to each system).

MAKEDEV accepts a number of arguments describing which collections of devices are to be created; once it has figured this out, it issues the correct sequence of `mknod` instructions to create the appropriate devices.

For example, to create the special devices corresponding to all 20 possible serial ports on the floor model IBM RT PC, give the command:

```
# /etc/MAKEDEV asy0 asy1 asy2 asy3 psp com1 com2
```

There are several other possible arguments that may be supplied to the MAKEDEV script. For instance, providing the argument '`std`' will cause MAKEDEV to create special files for all the "standard" devices necessary to run the IBM/4.3 system, such as the files corresponding to core memory (`/dev/kmem`) and the system trash basket (`/dev/null`). The only serial device file created in the standard configuration is `/dev/console`, which provides a logical device for the system console terminal. Also, if you execute the script with the argument '`local`', then another script, MAKEDEV`.local`, if it exists, will be executed. Any additional commands that may be required to configure the special device files should be put into MAKEDEV`.local`.

If you are planning to dedicate any of the available serial ports to receive dial-up logins over a modem, it is recommended that you change the names of the corresponding special files to the form `ttyd`*x*, where *x* is some digit or letter (i.e., *x* is in the range [`0-9a-zA-Z`]). The reason for this is that some programs expect a dial-up line to have a name of that form and may not operate correctly if a dial-up line has some other name. For example, if you plan to attach a modem to the serial port whose special file is `/dev/tty00`, then you should execute the following command, after you have created the special files:

```
# ln /dev/tty00 /dev/ttyd0
```

It is recommended that you place the appropriate commands into the file MAKEDEV`.local` instead of running them from the shell.

4. Creating Pseudo-Terminals

If you plan to allow users to login to your machine from remote machines on a local area network, you will need to create a number of device special files for pseudo terminals. Pseudo terminals are necessary for certain programs that expect input from terminal devices and which expect to send output to terminal devices. They are used in network operations (such as `rlogin`) since sockets do not provide the semantics required to manage terminals across a network. Thus, when a user logs into a remote machine, the remote shell takes input from and sends output to a pseudo terminal associated with the user's network login.

Unlike normal hard-wired terminals, there are actually *two* device files that need to be created for each pseudo terminal accessible by the system, a *master* device and a *slave* device. The function of the master device is to collect input from the socket managing the network communication and pass it as input to the slave device. From the slave device, the data are passed to a process reading the slave device as standard input (such as a login shell). If and when the process generates any output, that output is passed as input to the slave device, which then passes it to the master device and from there across the network to the real terminal of the user. The two levels are necessary so that a process running on a remote machine, including the user's login shell, can treat its standard input and output as a normal terminal. In the case described, the slave device appears to processes on the remote system exactly like a real terminal. The way in which pseudo terminals mimic real terminals is depicted in the following illustration:

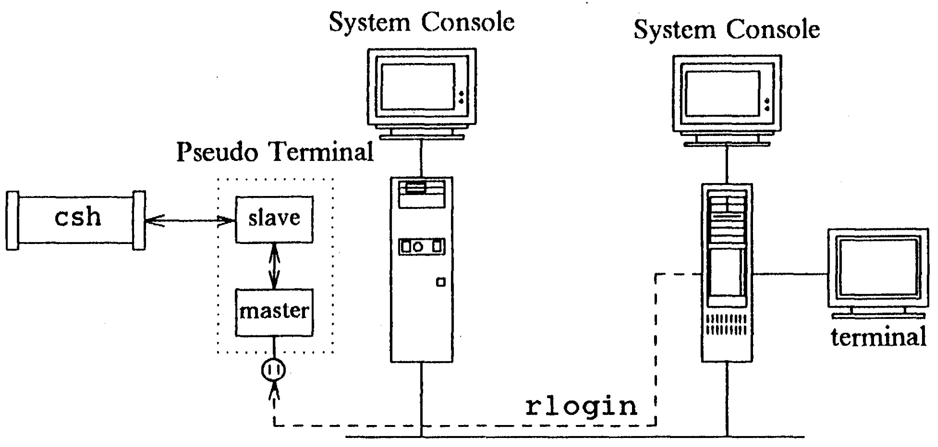


Figure 6-4: Real and Pseudo Terminals

Slave devices have names of the form `/dev/tty`*xy*, where *x* is `p`, `q`, or `r`, and *y* is a single digit or a letter in the range `a` through `f`. Each value of *x*, therefore, gives rise to sixteen possible slave devices. For example, the first sixteen slave devices are named `/dev/ttyp0` through `/dev/ttypf`. The naming of the master devices exactly mirrors that of slave devices, except that the three characters `tty` are replaced by the characters `pty`. So the first sixteen master devices are named `/dev/ptyp0` through `/dev/ptypf`.

To create both the master and slave pseudo terminals, execute the MAKEDEV script with the argument `pty`*x*, where *x* is `0`, `1`, or `2`. The added character tells MAKEDEV which range of pseudo terminals to create. For example, the command:

```
# /dev/MAKEDEV pty0
```

will cause MAKEDEV to create 32 new entries in the `/dev/` directory, corresponding to the sixteen slave devices and the sixteen master devices named above. If you plan to use the machine heavily for network activity or other activity requiring pseudo terminals, you may want to create additional groups of pseudo device entries. In that case, execute the command:

```
# /dev/MAKEDEV pty1 pty2
```

This command will create the master and slave devices for *x* set to the character `q` and for *x* set to `r`.

5. Testing the Hardware Configuration

Once you have reconfigured the kernel to recognize the serial ports available to the system and created the corresponding special device files, you can attach a terminal or modem to one of those ports. The cabling necessary to connect the device to one of the serial ports is machine-specific and should be provided by your hardware supplier. If not, you will need to refer to the documentation accompanying the adapter card and the serial device.

To test the correctness of the configuration and cabling, try sending some data to the special file. For instance, if a terminal is attached to a serial port whose special device file is `/dev/tty03`, execute the command:

```
# date > /dev/tty03
```

Another command useful in supplying random data to send to a serial port is the command `lptest`, which prints the traditional "ripple" pattern of characters on the standard output. So you might run the command:

```
# lptest > /dev/tty03
```

If all the steps covered up to now have been successfully accomplished (i.e., kernel reconfiguration, adapter card installation, special device creation, and hardware cabling), you should get some output on the terminal screen.

6. Login Configuration

If you wish to allow users to login over a particular serial line, the operating system needs to be instructed to monitor that line and wait for input from the terminal or modem attached to it. As the superuser, you can always send output through a serial line to the device attached to it. For the operating system to accept input from the line, however, and to know that that input is to be interpreted as a login attempt, the system must be specifically configured to do so.

The operating system enables a serial line for logins by launching a separate `getty` process for each serial line on which logins are to occur. The function of `getty` is to open and initialize the assigned serial line, and then wait until someone enters a login name. The position of the `getty` process in the login sequence can be seen in the following illustration:

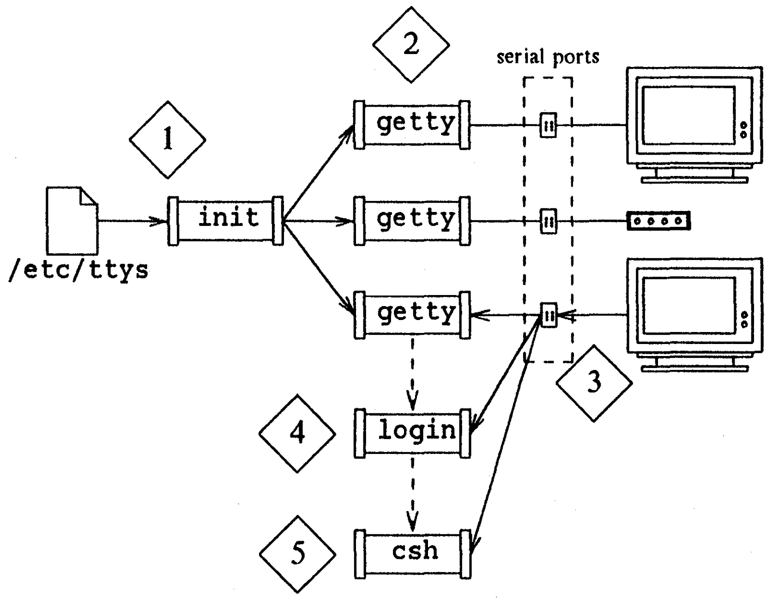


Figure 6-5: The Login Sequence

As indicated, the `init` process is responsible for creating the various `getty` processes that may be running. The `init` process knows which serial lines are to have `getty`s started on them by consulting the configuration file `/etc/ttys`. Once a login name is received by `getty` (step 3 in the diagram), `getty` invokes the `login` command to prompt for the password and verify the correctness of the login name and password. If the login name and (encrypted) password match those listed on some line in the file `/etc/passwd`, then the appropriate login shell will be started up for that user and that shell will interpret all further input from the serial line. When the user eventually logs out of the system, the `init` process is notified and it creates a new `getty` to monitor the serial line for new login attempts.

7. Configuring /etc/ttys

The file `/etc/ttys` is the central storehouse of information regarding which serial lines are to have `getty` processes created on them. The format of this file has changed significantly from the 4.2 release of the BSD system to the 4.3 release. The new `/etc/ttys` file incorporates information previously scattered through three different files, `/etc/ttys`, `/etc/ttytype`, and `/etc/securetty`. By collecting all the relevant terminal information into a single file, the IBM/4.3 system simplifies your job as system administrator.

The general format of `/etc/ttys` can be illustrated as follows:

device *startup-program* *type* *status*

Format 6-1: `/etc/ttys`

The *device* specification is simply the name of the device's special file as found in the `/dev` directory. The *startup-program* is the full path name of the process to be launched by `init` on the corresponding device. Usually the startup program is `getty` invoked with an argument that specifies the baud rate to use. In fact, however, the startup program can be any process at all, as you will soon see.

The *type* specification indicates what kind of device is connected to the serial port. You must select a terminal type from among those listed in the file `/etc/termcap`. As distributed, the IBM/4.3 system contains terminal descriptions for a large number of common terminal types, as well as entries for other kinds of connections such as `dialup` (for modem communications) and `network` (for network communications). If there is no listing in the data base `/etc/termcap` for a terminal you wish to connect to your system, refer to the section below entitled "Terminal Capability Descriptions" for assistance on constructing one.

The final column in the file `/etc/ttys` (disregarding any comments, which are introduced by the pound sign '#') indicates the *status* of the associated terminal line. If the `init` program is to launch a `getty` process (or some other startup program) on a specified device, the status field must contain the value '`on`'. If the status field contains the value '`off`', then no such process will be started up. If the status field is '`on secure`', then `root` logins are permitted on that line; otherwise they are not allowed. This provides some measure of security against unauthorized logins across certain terminal lines using the `root` uid. It is recommended that publicly available terminals and modems not contain the qualifier '`secure`'. The system console, on the other hand, should allow `root` logins.

The startup command does not have to be a `getty` process. To allow logins across a local area network, for instance, there should be no command at all listed, largely because network login services are provided by the daemon `rlogind` using pseudo terminals instead of serial lines. As a result, there is no need to launch `getty` processes to await login attempts across a network.

8. Sample /etc/ttys Entries

Here are some sample lines from an `/etc/ttys` file:

```
console  "/etc/getty ibm"       ibm5151 on secure
tty00    "/etc/getty std.9600"  vt102   on           # Room 312
tty01    "/etc/getty std.9600"  ibmaed  on           # Room 314
ttyd0    "/etc/getty std.1200"  dialup  on
ttyd1    "/etc/getty std.1200"  dialup  off
ttyp0    none                   network
ttyp1    none                   network
```

The first entry lists the information applying to the console terminal. Since it will be used for administrative purposes by the system administrator, the status is listed as '`on secure`'. The argument to the `getty` process indicates the speed at which the line should be opened and sets certain other parameters. The meaning of the argument is discussed below in the section "Configuring `/etc/gettytab`."

The second line illustrates a standard terminal hookup. It is exactly like the first line except that `root` logins are *not* allowed over `tty01` and it ends with a useful comment indicating the location of the terminal in question. We recommend adding such comments to the configuration file, since otherwise the location of a terminal may be difficult to determine.

The fourth and fifth lines illustrate the configuration for a dial-up line. The first of the two entries is set to accept logins while the second is not. Presumably, the second modem will be used for outgoing phone connections.

The last two entries in the samples show how to set up network logins. In this case, the two pseudo terminals `ttyp0` and `ttyp1` will accept logins over the network. In an actual `/etc/ttys` file, there will be many more lines like the last two (as you have already seen above).

9. Configuring /etc/gettytab

The startup command listed in the `/etc/ttys` file for a particular serial line is usually `getty`. As indicated above, `getty` opens the corresponding special device file and awaits input from that line. To do this job correctly, however, `getty` needs to know certain things about the serial line, such as the baud rate of the peripheral device attached to the line, the parity setting, whether the device can generate lowercase letters, and a number of other parameters. `getty` gets this information from the file `/etc/gettytab` by way of the argument that was provided in the file `/etc/ttys`. For example, consider the entry:

```
tty00     "/etc/getty std.9600"     vt102     on              # Room 312
```

When `init` launches a `getty` process to monitor `/dev/tty00`, it does so by executing the command:

```
/etc/getty std.9600 tty00
```

The first argument '`std.9600`' indicates that the terminal attached to `/dev/tty00` is a "standard" terminal operating at 9600 baud. To see exactly what that means, you need to look at the corresponding entry in `/etc/gettytab`. On most systems, it is no more complicated than this:

```
2|std.9600|9600-baud:\
        :sp#9600:
```

The format of the entries in `/etc/gettytab` resembles those in `/etc/termcap` (with which you are probably already familiar). The first line lists the names of the entry; notice that the second name listed is exactly the argument provided to `getty` in the entry in `/etc/ttys`. The remaining lines list the terminal line capabilities. The sample entry above has just a single capability listed, namely '`sp`' representing the speed of the line in question.

The configuration for terminal type `std.9600` is so simple because `getty` automatically reads and acts on any terminal line characteristics listed in the entry whose name is `default`. This fact is extremely useful in allowing you to tailor the `getty` parameters for all terminal lines in one fell swoop. For example, a typical entry for the `default` type looks like this:

```
default:\
        :ap:fd#1000:
        :im=\r\n\r\nIBM 4.3 (%h) (%t)\r\n\r\r\n\r:
        :sp#1200:
```

This specifies that the terminal can use any parity (`ap`), that the system should delay 1000 milliseconds after printing a form feed (`fd#1000`), and that the default speed is 1200 baud. This entry also indicates the initial banner (`im`) that is to be printed on the terminal screen so that users know that the system (i.e. `getty`) is awaiting their input. When `getty` is launched (by `init`), it will print a banner that looks like this:

```
IBM 4.3 (poisson) (tty00)

login:
```

To change the banner, simply alter the string assigned as the value of `im` in the `default` entry in `/etc/gettytab`. Note that the escape sequence `\r` is mapped to a carriage return when the banner is output on the terminal screen and that the escape sequence `\n` is mapped to a newline character. `getty` also performs certain other substitutions on the string specified in the `im` capability. The substring `%h` is replaced by the hostname of the machine. Similarly, the substring `%t` is replaced by the name of the serial port to which the terminal or modem is attached. The second part of this banner is the default login message, as specified by the `lm` capability. Since there is no `lm` capability listed in the `default` entry, the default message is used.

A large number of other line characteristics can be configured in the entries in `/etc/gettytab`. A complete list may be obtained by consulting the manual pages for `gettytab`(5). In almost all cases, the default values of the various capabilities have been selected so that `gettytab` entries can be very short and simple. As a result, except for changing the initial banner output by `getty`, you may never need to alter any `gettytab` entries or add entries to it.

10. Converting 4.2 /etc/ttys to the 4.3 Format

As noted above, the format of the file.`etc/ttys` has changed drastically from the 4.2 release of the BSD system to the 4.3 release (upon which IBM/4.3 is based). If you are converting from 4.2 to IBM/4.3, you may wish to employ the following procedure to help automate the construction of a new `/etc/ttys` file.

First of all, install the following script on your system; you can use any name, but the C shell script that calls this `awk` script assumes it is called '`ttys.scr`'.

```
# awk script to help convert 4.2 ttys file to 4.3 format

BEGIN { FS = "	" }

FILENAME == "-" {
	FS = "	"
	if ( NF > 1 ) { getty[$1] = $2 }
	else          { getty[$1] = $1 }
}

FILENAME == "/tmp/ttys" {
	port = substr( $1, 3, length($1)-2 )
	list[port] = port
	if( substr( $1, 1, 1 ) == 1 ) {
		on[port] = "on"
	}
	else { on[port] = "off"
	}
	pty = substr( $1, 6, 1 )
	if ( substr( $1, 3, 3 ) == "tty" \
	  && ( pty == "p" || pty == "q" || pty == "r" ) ) \
	     { comm[port] = "none" }
	else { comm[port] = substr( $1, 2, 1 ) }
	type[port] = "unknown"
	secure[port] = ""
}

FILENAME == "/etc/ttytype" { type[$2]  = $1 }

FILENAME == "/etc/securetty" { secure[$1] = "secure" }
```

```
END { for ( port in list ) {
      if ( getty[port] == "none" ) {
           printf( "%s
      } else {
           printf( "%s
                          getty[comm[port]] )
         . }
      printf( "%s%s%s0, type[port], on[port], secure[port] )
      }
}
```

This script assumes that your old (4.2) version of the file /etc/ttys is located in the file /tmp/ttys. In addition, the 4.2 versions of the files /etc/securetty and /etc/ttytype are located in their original locations. To use this awk script, run the following simple script:

```
#! /bin/csh -f
# conv_ttys: construct 4.3 ttys file from 4.2 files

/usr/bin/egrep "\^[0-9a-zA-Z]" /etc/gettytab \
     | /bin/sed -e "s/|/ /g" -e "s/://g" -e "s/\//g" \
     | /bin/awk -f ttys.scr - /tmp/ttys
             /etc/ttytype /etc/securetty \
     | sort
```

Once this C shell script and the previous awk script are in place on your system, you can create a 4.3 version of the /etc/ttys file simply by running the command:

```
# conv_ttys > /etc/ttys
```

This script uses awk and several other utilities to merge the previous /etc/ttytype, /etc/securetty, and /etc/ttys (which is stored in /tmp) to create a new /etc/ttys that is in the format required by the IBM/4.3 system.

11. Terminal Capability Descriptions

The IBM/4.3 system can allow virtually any kind of terminal to be used as a communications device, whether it is a hard-copy teletype machine, a normal ASCII character terminal, or a high-resolution bit-mapped graphics display device. The system accomplishes this by isolating terminal-dependent information into a common data base of terminal capability descriptions (called /etc/termcap) that applications programs can read to determine the appropriate display characteristics of any particular terminal. For instance, when a user instructs the system to clear the screen by issuing the command clear, the program will consult the terminal description data base to see what characters must be sent over the communications line to have the terminal clear its screen.

There are two main ingredients to achieving successful communication between the system and a user at a terminal. First, the administrator must make certain that the system prompts the user to identify the type of terminal being used. This can be accomplished quite easily by including the appropriate lines in the user start up file (usually either .login or .profile). If the user in question has the C shell as the login shell, then the following few lines of code will prompt the user for the terminal type:

```
echo "Please confirm (or change) your terminal type:
set noglob; eval `tset -e^H -s ?$TERM`
```

If the start-up shell for a user is the Bourne shell, the following few lines of code will accomplish the same goal:

```
echo "Please confirm (or change) your terminal type:
```

```
export TERM
TERM=`tset - -e^H`
```

In either case, the system will set the environment variable TERM to a string that uniquely identifies the terminal type.

The second main administrative activity associated with adding terminals to a system is making sure that the file `/etc/termcap` contains correct terminal descriptions for all terminals that will be connected to the system. The `/etc/termcap` data base supplied with the IBM/4.3 system contains terminal descriptions of a very large number of terminals, so it is likely that a description of a reasonably common terminal already exists there. To see if such a description exists (and, if it does, what its abbreviation is), try to scan for a pattern that might identify the terminal. For instance, if your terminal is manufactured by Televideo, try the command:

```
# grep televideo /etc/termcap
```

Most likely, you will get copious output, like this:

```
v1|tvi912|912|920|tvi920|old televideo:\
v2|912b|912c|tvi912b|tvi912c|tvi|new televideo 912:\
v3|920b|920c|tvi920b|tvi920c|new televideo 920:\
v4|tvi912-2p|tvi920-2p|912-2p|920-2p|tvi-2p|televideo w/2 pages:\
vi|tvi925|925|televideo model 925:\
vj|tvi925vb|925vb|televideo model 925 visual bells:\
vn|tvi925n|925n|televideo model 925 no standout or underline:\
vk|tvi925vbn|925vbn|televideo model 925 visual bells no so or ul:\
va|tvi950|950|televideo950:\
vb|tvi950-2p|950-2p|televideo950 w/2 pages:\
vc|tvi950-4p|950-4p|televideo950 w/4 pages:\
vd|tvi950-rv|950-rv|televideo950 rev video:\
ve|tvi950-rv-2p|950-rv-2p|televideo950 rev video w/2 pages:\
vf|tvi950-rv-4p|950-rv-4p|televideo950 rev video w/4 pages:\
vg|tvi924|924|televideo model 924:\
vo|tvi924vb|924vb|televideo model 924 visual bells:\
vp|tvipt|televideopt:if=/usr/lib/tabset/stdcrt:\
vh|tvi910+|910+|televideo 910+:\
va|ims950|ims televideo 950 emulation:\
```

This output indicates that a number of different kinds of Televideo terminals are recognized by the system. If a user is working at a Televideo model 920c, for example, then the appropriate terminal type abbreviation would be `v3`.

If the `grep` command listed above produces no output or none of the lines lists your terminal type, it is possible that a terminal description does in fact exist in `/etc/termcap` but doesn't contain the manufacturer's name. If so, you should look at the complete file to make certain that the terminal isn't listed under a slightly different name. So try:

```
# page /etc/termcap
```

If you still do not find an entry that seems to describe your terminal, then you will have to obtain a new capability description for the terminal and insert it into `/etc/termcap`. It is beyond the scope of this manual to give complete instructions for writing a terminal description from scratch; if you need to do this, consult the manual entry `termcap`(8). Generally it is possible to put together a working termcap description rather quickly by consulting those manual pages and the manual accompanying the terminal in question. Another possibility, of course, would be to obtain a working `termcap` description for the terminal in question from someone who has already written one. The USENET newsgroup `comp.terminals` serves in part as a way of exchanging terminal descriptions, so you may wish to inquire there.

12. Terminal Management Tips

- Remember that no terminals are active during single-user mode, other than the system console. In addition, no logins will be allowed over any modem lines that are attached to your system. This will prevent your machine from receiving incoming uucp or network transmissions.
- It is possible to edit the file `/etc/ttys` during normal multi-user operation in order to activate or deactivate terminal lines "on the fly". To cause `init` to reread `/etc/ttys`, send it a hangup signal:

  ```
  # kill -HUP 1
  ```

  Newly-activated serial lines will have a `getty` process (or whatever startup command is specified) launched. Similarly, lines that have been switched from '`on`' to '`off`' will have any existing shell program killed. Thus it is not necessary to reboot your system simply to reconfigure terminal lines.
- To prevent `init` from launching new `getty` processes when users log out, send it a terminal stop signal, as follows:

  ```
  # kill -TSTP 1
  ```

  This is most useful for letting user activity dwindle down in preparation for an upcoming system reboot. If you change your mind and wish to resume full multi-user operation, send `init` a hangup signal, as illustrated above.

CHAPTER 7

Creating and Maintaining File Systems

1. Introduction

It is probably fair to say that a good understanding of the structure of the IBM/4.3 file system is the most important element of successful system administration. As a system administrator, you will need to manage the space used by system and user files, add disks when the existing space becomes inadequate, and balance the system's use of the available storage space. Since the IBM/4.3 system treats practically everything as a *file* of some sort or another, it is essential to maintain a clean and uncorrupted system of files. This chapter will help you understand how the IBM/4.3 operating system stores files and how it manipulates them to perform typical functions. It also explains how to create and maintain file systems and how to fix them in case of corruption.

You should note that the details on file system organization and maintenance given in this chapter do *not* apply to any Andrew File Systems that may be present on your system. In particular, use of the file system consistency checker `fsck` on Andrew File Systems will result in irreparable file system damage. For instructions on the maintenance of Andrew File Systems, refer to Chapter 19, "The Andrew System", and to the separate document *The IBM Andrew File System*. This chapter applies only to normal, undistributed IBM/4.3 file systems.

2. Overview of IBM/4.3 Files and File Systems

Logically, a file is an arbitrary sequence of bytes. Every file is located somewhere within the IBM/4.3 file system and typically has a name by which it can be accessed (perhaps by giving its name as an argument to some command). As the following diagram makes clear, files are located in a logical *hierarchy* in which certain files, called *directories*, contain other files and directories.

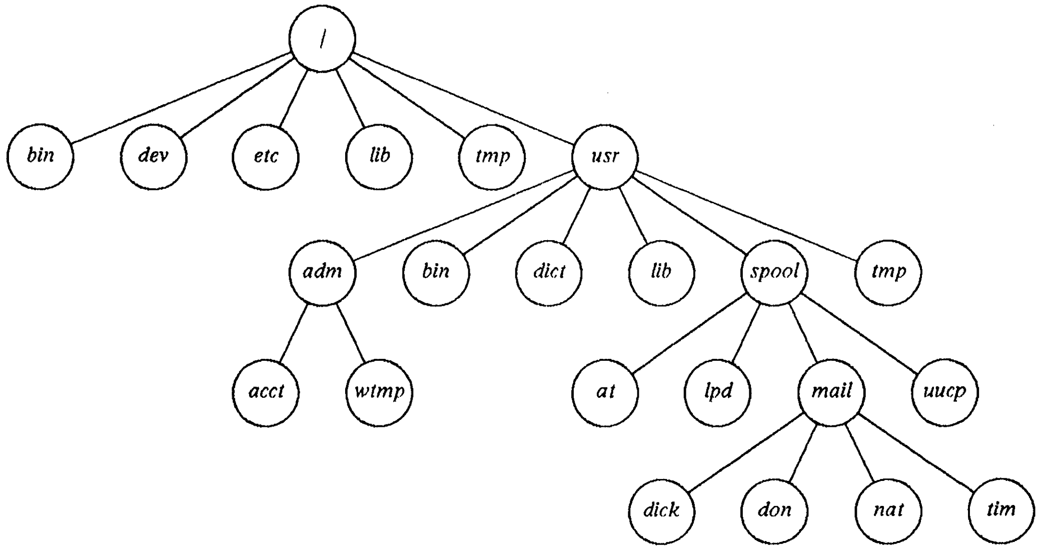


Figure 7-1: Part of the File System Hierarchy

The full or absolute name of a file is simply a listing of the names of all the directories above it, separated by the slash character, '/', followed by the basename of the file (the name it has in its parent directory). For example, the absolute name of the file in the lower right-hand corner of the previous diagram is:

```
/usr/spool/mail/tim
```

This form of the name is also called an absolute *path name*, since it specifies the full path taken to the file from the highest, or "root" directory, /.

Physically, the bytes that make up a file are located on a storage medium such as a rigid (or "hard") disk. A disk is a collection of addresses, each of which can contain a value that the system can usually both read and write. An IBM/4.3 system may have many physical disks attached to it, and each disk may be divided into several file systems. A disk usually contains several *platters* or surfaces, as illustrated in the following picture:

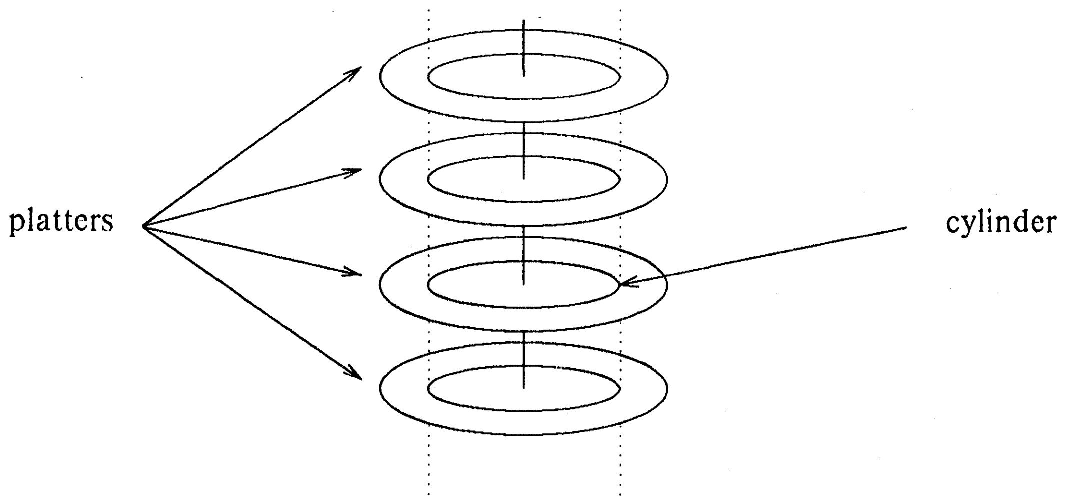


Figure 7-2: Simplified View of a Hard Disk

An important division of a disk is the *cylinder*, which consists of all the data located on the same track on all platters. Each disk has numerous cylinders, one of which is highlighted in the preceding illustration. Not all disks have multiple platters, however; floppy disks, for example, consist only of a single surface (which may usually be written on front and back). For this reason, disks shall be illustrated throughout the rest of this guide showing only a single surface.

The contents of a file are not necessarily stored sequentially on the disk. Although the system tries to avoid file fragmentation by keeping a file's contents in the same cylinder or in contiguous cylinders, the file's contents may very well be scattered in chunks all over the disk. In spite of this apparent chaos, the way that the IBM/4.3 system reduces the logical entity to the physical collections of bytes is really quite simple, and you must understand it thoroughly in order to be able to administer the system well.

A disk platter is divided into groups of cylinders called *partitions*. A partition can contain a file system, a swap area, or it may be completely unused by the system. A partition may even be reserved by the system for its own use. For example, the IBM/4.3 system does not allow access to the entire physical disk, since it reserves the beginning and the end of the disk for maintaining bad sector information about the disk.

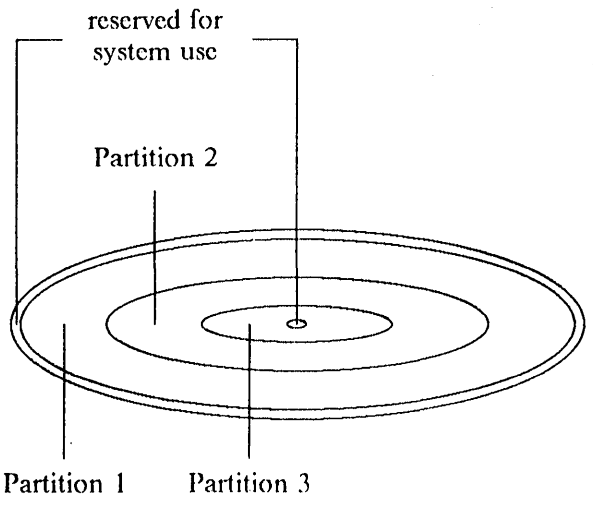


Figure 7-3: How A Physical Disk is Divided into Partitions

The division of a disk into partitions allows several file systems, swap areas, or both to be located on the same physical disk. The partitions are created when the disk is formatted and to repartition a disk requires that you reformat it (thereby losing all information stored on it).

A file system is created in a disk partition with the **mkfs** command (or with the **newfs** command, which is a "friendly" front-end to the **mkfs** command). The **mkfs** command sets the size and format of the file system. Each partition may contain at most one file system.

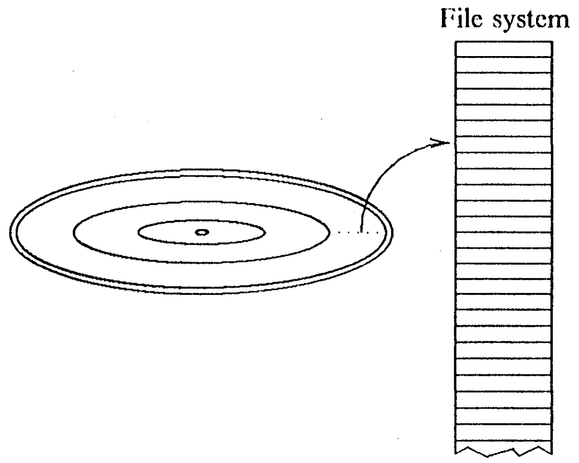


Figure 7-4: Each Partition Contains (At Most) One File System

Every file system is divided into a set of *blocks*. A block is a sequence of bytes having a predetermined size. Most previous UNIX systems used blocks of fixed size, either of 512 or 1024 bytes. The IBM/4.3 system, however, with its "fast file system", allows each file system to have blocks of a different size. The minimum size of a block is now 4096 bytes, and the size of a block may be increased (to any power of 2) if necessary to optimize disk performance.

The first blocks in the file system comprise an entity known as the *superblock*. The superblock contains critical information about the file system, such as the size of the blocks in the file system, the number of blocks contained in the file system, the time that the file system was last modified, and so on. The information contained in the superblock is so important to the system that the superblock is replicated at selected points in the file system to guard against losing the entire file system and all the data in it in the case of corruption of the superblock.

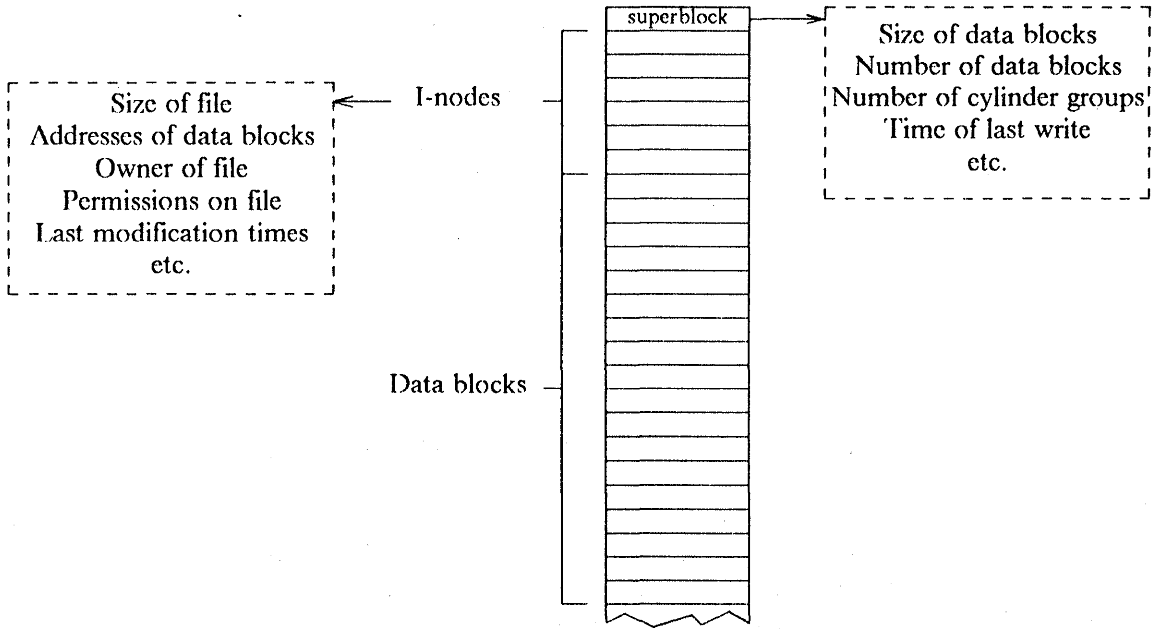


Figure 7-5: General Structure of a File System

Following the superblock in the file system is a section of blocks that contain *identification nodes*, (or *i-nodes*, for short). I-nodes contain information about individual files, such as their location in the file system, their sizes, the time they were created and last modified, and who owns them. A file system contains a fixed number of i-nodes, which is determined at the time the file system is created. I-nodes are not referred to by name, but by the order (or index) in which they are found in the section of i-nodes. The index of a particular i-node in the section of i-nodes is called the *i-node number* (or *i-number*, for short). To display the i-number of a particular file, you can run the `ls` command with the `-i` option. For example, the following command will give a long listing of the `/etc/rc` file, along with the i-number of that file:

```
# ls -li /etc/rc
   874 -rw-r--r--  1 root          2746 Jan 21 03:19 /etc/rc
```

Following the section of blocks which contain i-nodes are the *data blocks*. These are the blocks that actually contain the bytes (or data) which make up files and directories. The data blocks of a file are selected by addresses contained within the i-node of a file, as illustrated in the following figure:

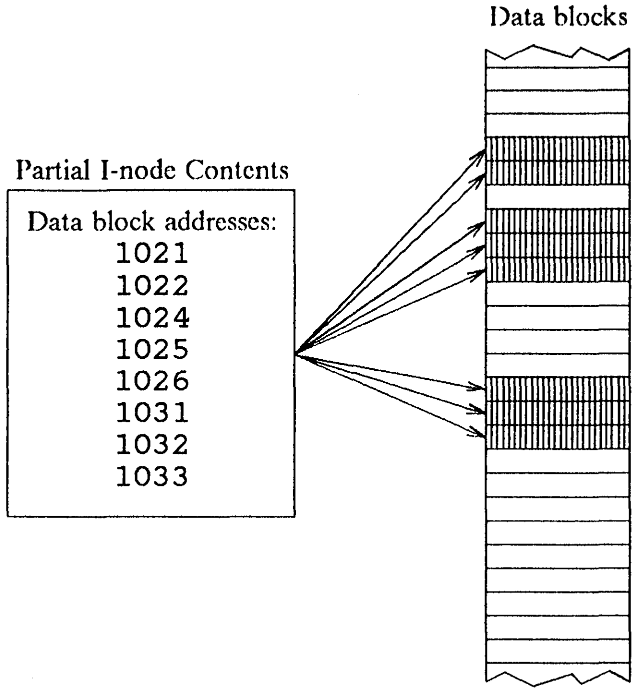


Figure 7-6: An I-node Contains Disk Addresses of Data Blocks in File

In the IBM/4.3 system, an i-node has space for 12 direct data block pointers. That means that up to 12 data blocks can be addressed *directly* by the information contained in an i-node. If a file is larger than 12 × 4096 bytes (about 50,000 bytes), then the i-node is not able to contain all of the addresses of the data blocks occupied by the file. In that case, the i-node will contain, in addition to the 12 direct data blocks, the address of a disk block (called an "indirect address block") that contains more addresses. These additional addresses pick out the remaining blocks of the file, as illustrated in the following figure:

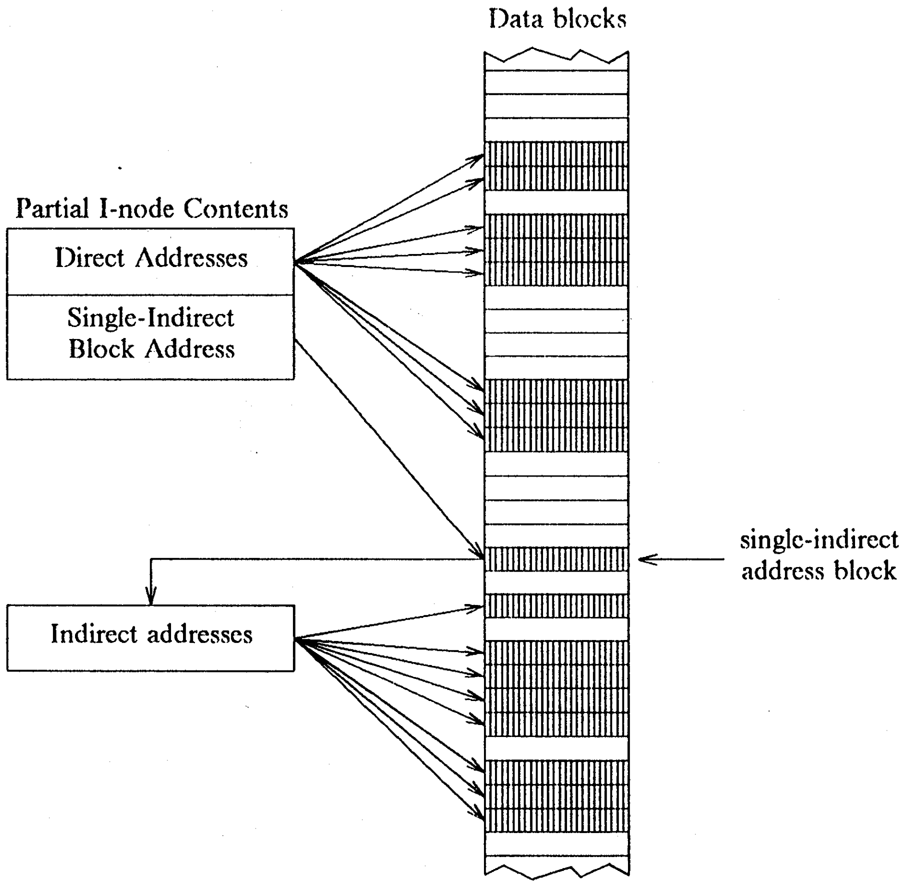


Figure 7-7: Single-Indirect Data Blocks

Exactly how many addresses can be contained in the indirect address block depends on the size of the block, which was determined when the file system was created. Similarly, the size of a file which can be addressed exclusively using direct data blocks depends on the size of a block in the file system. For example, if the block size for a particular file system is set to 8192 bytes, then a file may be as large as about 100,000 bytes before indirect addressing is needed. Since an additional disk read and increased computing time are required to access and interpret the addresses of the indirect blocks, you can improve system performance by increasing the block size on file systems that contain commonly-used files that are in the 50,000 to 100,000 byte size range. In particular, a minimum size of 8192 bytes per block is recommended for the root file system, containing `/bin` and `/tmp`.

If a file is too large to be addressed even using all the addresses that will fit into an indirect address block, then the system will look in the i-node for a second indirect address block, which picks out another group of disk addresses.

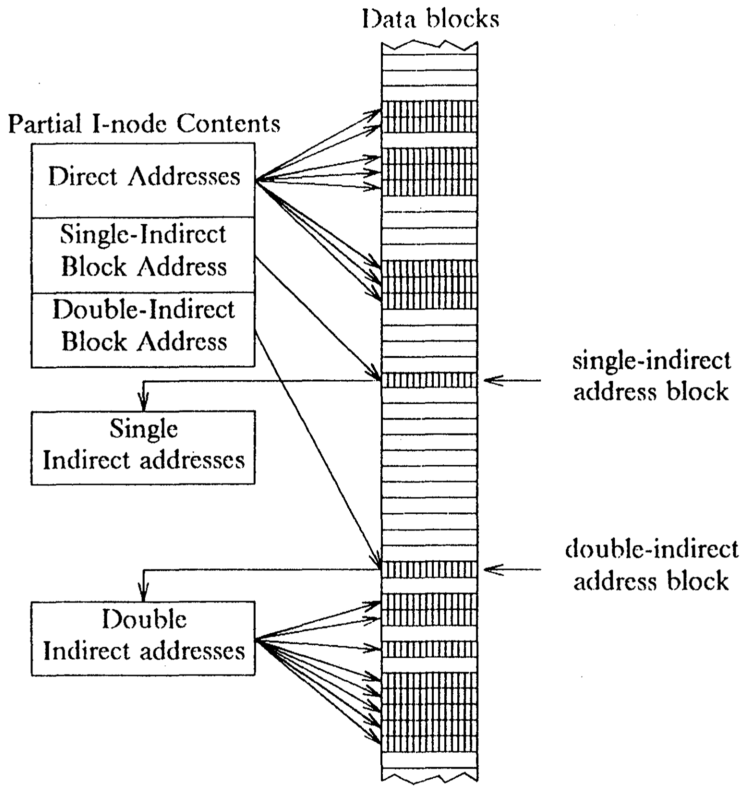

Data blocks
Partial I-node Contents
Direct Addresses
Single-Indirect Block Address
Double-Indirect Block Address
Single Indirect addresses
Double Indirect addresses
single-indirect address block
double-indirect address block

Figure 7-8: Double-Indirect Data Blocks

Because the minimum block size was set to 4096 bytes, it is possible to create files as large as 2^{32} bytes using the direct blocks, the single-indirect blocks, and the double-indirect blocks. An i-node in fact contains one further slot for an indirect address, allowing for the possibility of triple-indirect data blocks. The IBM/4.3 system does not utilize this capability, however, unlike most earlier UNIX releases.

3. Disk Partitioning

Each physical disk can be divided into several partitions, some of which may in fact overlap. Currently, the limit on the number of partitions that may exist on a single disk is eight, and the partitions are consequently named 'a' through 'h'. The general arrangement of these partitions on a physical disk can be illustrated as follows:

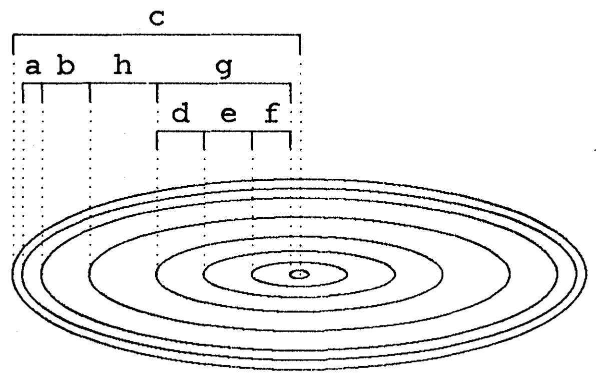


Figure 7-9: The Layout of Disk Partitions

On drives that are smaller than about 200 megabytes, the h partition is usually omitted, in which case the general disk partition layout looks like this:

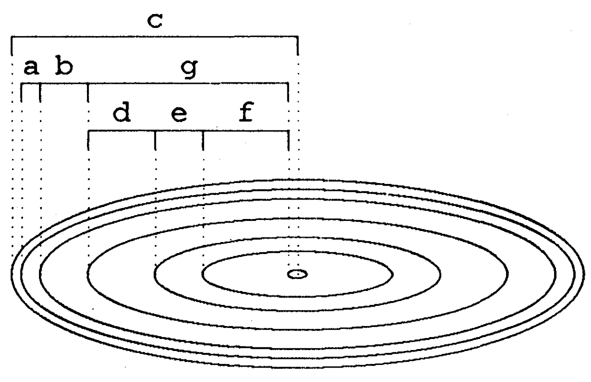


Figure 7-10: The Layout of Disk Partitions on a 70 Megabyte Disk

Similarly, on drives that are smaller than about 60 megabytes, the e partition is omitted as well. The typical partition arrangement on a 40 megabyte disk therefore looks like this:

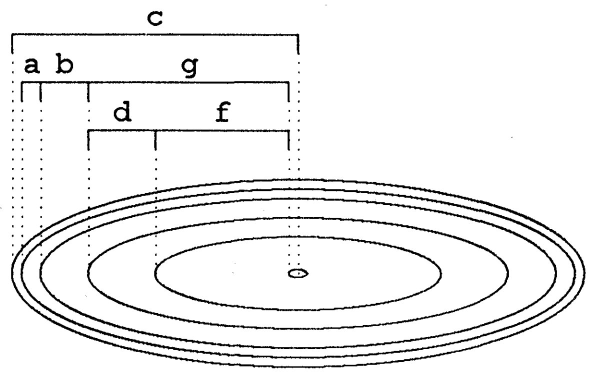


Figure 7-11: The Layout of Disk Partitions on a 40 Megabyte Disk

The a partition is usually used for the root partition, which is where the root file system resides (including the boot program /boot and the kernel /vmunix). The b partition is commonly used as an area for paging and swapping activity. The remaining partitions may be used to hold file systems or additional swap areas, or they may be left unused. In any case, if a file system is created in some disk partition, then it is not possible to create a file system in any partition that overlaps that partition. For example, if a file system is created in the c partition, then you will not be able to create file systems in any of the other partitions. Similarly, if the g partition is used to hold a file system, then you cannot use any of the d, e, or f partitions. So even though all eight partitions may exist on a given disk, not all of them will be actively in use by the operating system.

As you can surmise, the partitions actually in use on a particular disk will fall into one of three basic patterns:

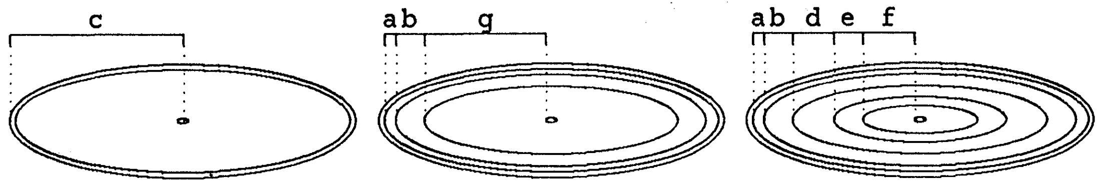


Figure 7-12: The Three Most Common Partition Use Patterns

The first option is to devote the entire disk, minus the reserved space at the beginning and the end of the disk, to a single file system. This commonly occurs when large amounts of space are required to hold users' home directories and personal files. In some cases, the `c` partition itself can be used to access the entire disk, excluding the space at the beginning and end of the disk reserved for bad-sector information. (See the section below, "Using the **`minidisk`** Utility", for further instructions on devoting an entire disk to a single file system.) A second possibility is to have a root file system, a swapping and paging area, and then a large third partition (perhaps for a `/usr` directory). The third basic usage pattern is just like the second, except that the `g` partition is not used, while the `d`, `e`, and `f` partitions are. This allows the system administrator somewhat finer control over the use of the disk, since file systems in the `d`, `e`, and `f` partitions may be mounted and unmounted independently of one another.

When constructing disk partitions, the IBM/4.3 system uses several default configurations in order to determine how large to make a particular partition. The amount of space allocated to each partition on the two most common types of disk in the default arrangement is listed in the following table:

Partition	40MB	70MB
a	15884	15884
b	10032	33440
c	87040	141372
d	15884	15884
e	N/A	55936
f	43826	19404
g	59721	91476

Table 7-1: Standard Partition Sizes

Note that the `a` partition, used for the root file system, is the same size on all drives, 15884 sectors. There is nothing sacred about this number, however, and it can often be useful to have a much larger root partition, especially if you reconfigure your kernel quite often and would like to maintain numerous system images in the root file system.

4. Creating Disk Partitions

The operating system accesses each disk partition through an entry in the `/dev` directory. These partitions are named according to the following scheme: the first part of the name is either '`hd`' or '`sc`', depending upon whether the disk is an internal hard disk or whether it is attached to the system through the SCSI connector. Since there may be several of each of these two types of disk, this two-letter sequence is followed by a single digit, beginning with '0'. Finally, the partition name is completed by adding a single letter designating one of the eight possible partitions, 'a', 'b', 'c', 'd', 'e', 'f', 'g', or 'h'. So, for example, a partition may be named 'hd0a'.

The special device files for hd0 look like this:

```
brw-r-----  1 root      1,   0 Jan  9 22:18 hd0a
brw-r-----  1 root      1,   1 Jan  9 22:18 hd0b
brw-r-----  1 root      1,   2 Jan  9 22:18 hd0c
brw-r-----  1 root      1,   3 Jan  9 22:18 hd0d
brw-r-----  1 root      1,   4 Jan  9 22:18 hd0e
brw-r-----  1 root      1,   5 Jan  9 22:18 hd0f
brw-r-----  1 root      1,   6 Jan  9 22:18 hd0g
brw-r-----  1 root      1,   7 Jan  9 22:18 hd0h
```

The IBM/4.3 system typically uses only three or four of the eight possible partitions. The first partition holds the root file system, which contains a bootable operating system, /vmunix. The second partition is used for paging and swapping activity, while the third is used for the /usr file system:

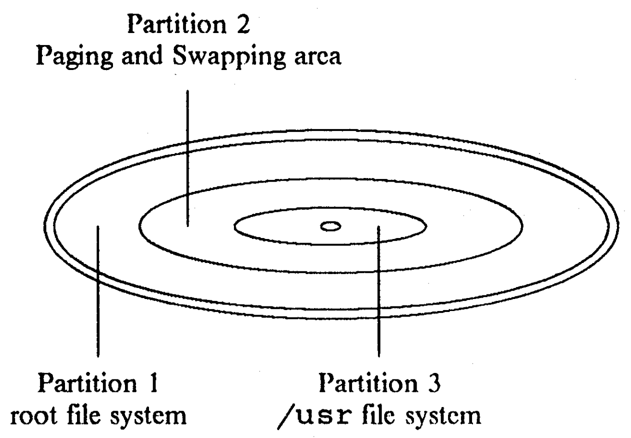


Figure 7-13: The Typical Disk Partitions for IBM/4.3

To create the eight partitions on a new disk, you may use the MAKEDEV script already discussed in the previous chapter. For instance, to make the partitions for the second hard disk, hd1, give the command:

```
# /dev/MAKEDEV hd1
```

The MAKEDEV script will consult the data base /etc/disktab to determine what the default partition sizes are for that type of disk.

4.1. Changing Default Partition Sizes

The IBM/4.3 system is distributed with six default disk partition tables compiled into the kernel, corresponding to the hd40m, hd70m, hd40r, hd70r, hd70e, and hd70c drives. For any of these drives, the sizes of the eight disk partitions can be changed to any reasonable values you care to give them. To do so, you must edit the partition table in the kernel and make similar modifications to the entries in /etc/disktab. The relevant section of the kernel source code is found in the file /sys/caio/hd.c. There you will find initializations of several array structures, each of which contains 8 pairs of integers. For example, the code for the partitions of a hd70c disk looks like this:

```
hd70c_sizes[8] = {
     15884,    1,   /* A=cyl 1 through 104 */
     33440,    105, /* B=cyl 105 through 323 */
     141525,   0,   /* C=cyl 0 through 1023 */
     15884,    324, /* D=cyl 324 through 427 */
     55936,    428, /* E=cyl 428 through 793 */
     18819,    794, /* F=cyl 794 through 916 */
     90729,    324, /* G=cyl 324 through 916 */
     0,   0,
}
```

As you can see, a partition is simply a contiguous chunk of disk cylinders which may or may not overlap other such chunks. You can modify these values as necessary, subject to certain obvious constraints. You cannot for instance modify the `c` partition, since it corresponds to the entire physical disk. Similarly, the `a` partition must start at cylinder 1.

5. Cylinder Groups

Generally, it is inefficient to use a file system structure in which the i-nodes are all located at the beginning of a file system and all the data blocks follow the i-nodes, as illustrated earlier. For instance, it is very common that all the i-nodes for the files in a particular directory need to be inspected in order to complete a user request (such as `ls -l`). In that case, it makes sense to arrange those i-nodes in such a way that they can all be accessed in the smallest amount of time; this can of course be done fairly easily by locating them in the same cylinder or in contiguous cylinders on a disk. But if the i-nodes for a file system are all located at the beginning of the file system, it is more difficult to enforce an i-node "clustering" scheme.

Similarly, many user requests require that the system inspect both a directory and the i-nodes for the files in that directory. But under the simple file system layout illustrated, that is generally impossible, since a directory is just a file and is hence stored in the data block section of the file system. It would improve performance if a directory could be located near the i-nodes it references.

For these and other reasons, this basic picture of the structure of a file system is complicated slightly by an enhancement to the IBM/4.3 fast file system designed to increase file system throughput and reduce file fragmentation. This enhancement is the subdivision of partitions into *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Within each cylinder group is a section of bookkeeping information (mostly occupied by i-nodes) and a section of data blocks, as depicted in the following diagram:

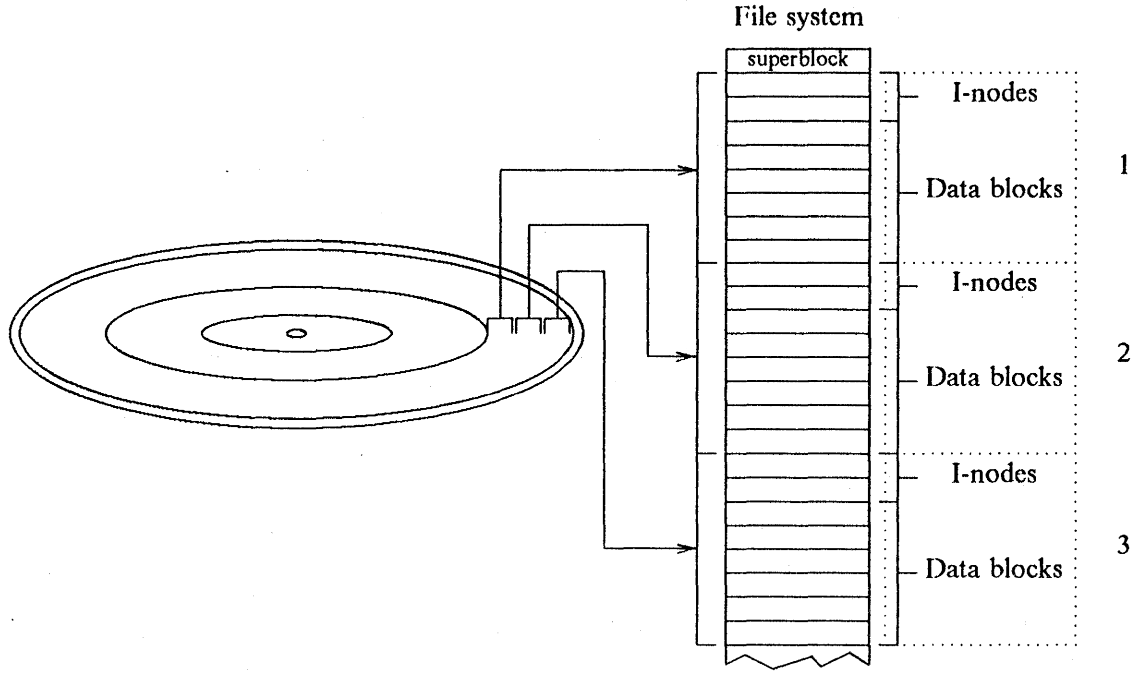


Figure 7-14: The Physical and Logical Positions of Cylinder Groups

As you can see, it is now generally possible for the operating system to situate a directory and the i-nodes for the files in that directory in the same cylinder group. When a new directory is created, the system automatically places it and the associated i-nodes in the cylinder group that has a greater than average number of free i-nodes and the fewest directories already in it. This policy ensures that i-node clustering will happen as often as possible and that directories and i-nodes will be as contiguous as possible. As a result, rotational latency is reduced and file system performance is increased.

6. Block Fragments

The larger block sizes in the IBM/4.3 system provide improved file system performance by allowing larger chunks of data to be transferred to or from the disk in a single disk transaction. The larger minimum block size also reduces the need for indirection and thereby speeds up file reading and writing for files that are large enough that they would have required indirection under a smaller block size. One drawback of this scheme, however, is that a larger block size is apt to waste disk space. In a file system with blocks that are 4096 bytes, for example, a file of exactly 10,000 bytes would need to occupy three disk blocks, accounting for a total of 12,288 byte of space. In that case, as you can see, over two thousand bytes of the disk (more than half a block) are wasted. In a system that employs 512-byte blocks, the same 10,000 byte file could be contained in 20 blocks, accounting for a total of 10,240 bytes. As you can easily see, only 240 bytes of the disk are wasted in the latter system, less than one tenth of the space wasted in the system with a large block size.

In an effort to reduce the amount of space "wasted" in this manner, the IBM/4.3 system introduces a new disk object called the *fragment*. A fragment is a division of a block that can be individually addressed and used to contain part of a file's data. Within certain limits, you can determine how many fragments are to fit into a block on a particular file system, since a block may be divided into either two, four, or eight fragments. This value must be supplied at the time the file system is created, as you will see below. If the file system on which the 10,000 byte file is stored allows eight fragments to a single 4096-byte block, then that file can be stored in two full blocks (8192 bytes) and in four fragments (2048 additional bytes). The four remaining fragments in the subdivided block are available for use by some other file. Under this arrangement, then, the total wasted space is only 240 bytes (i.e., [8192+2048]-10000), which is exactly equal to the wasted space under the system with 512-byte blocks.

The way in which fragments help to reduce "wasted" space is shown in the following illustration.

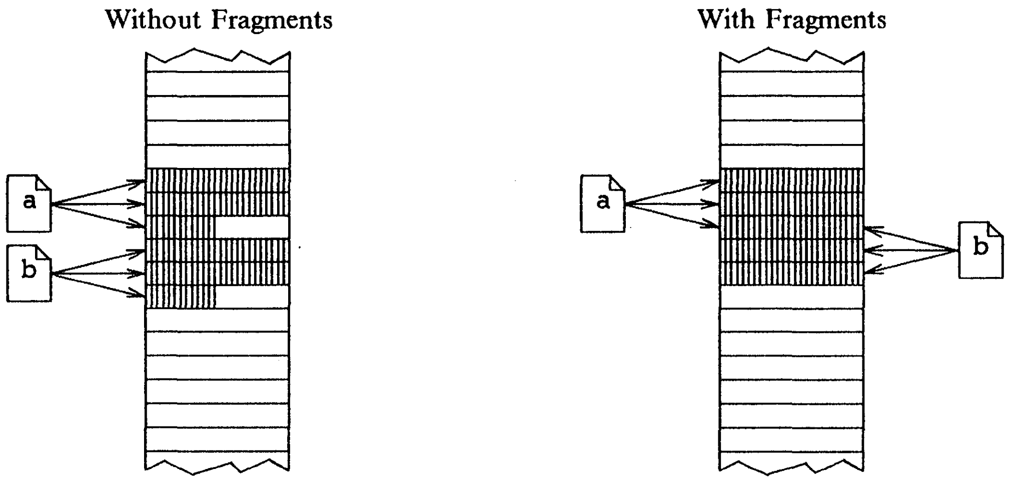


Figure 7-15: How Fragments Help Reduce Wasted Space

Here, the two files a and b are to be stored in a file system. In a file system with no fragments, the two files would each occupy some number of blocks completely, with any remainder in each case assigned to part of a complete block. In a system where fragments are permitted, however, the operating system would put whatever part of the second file that does not fit into complete blocks into the fragments left available by the first file. In this idealized picture, an entire disk block has been saved by allowing parts of files to be placed into disk block fragments.

The division of blocks into fragments is handled automatically by the operating system, and the only time it is of any real concern to the system administrator is at file system creation time. See below for instructions on how to specify the number of fragments per block.

7. Cylinder Group Bookkeeping Information

The bookkeeping information placed into each cylinder group is something of a mixed bag. Mostly it includes the i-nodes for the files and directories whose data blocks are found in the data block section of the cylinder group, but it also includes several other items. First, there is a copy of the file system superblock. The superblock is replicated in each cylinder group in order to protect against the catastrophe that would ensue if the only copy at the top of the file system were damaged. Second, the bookkeeping section of the cylinder group contains a bitmap describing the available blocks in the cylinder group. Previous incarnations of the UNIX operating system (including all current System V releases) maintain a list of free blocks, organized as a linked list. The free list typically begins in the superblock and extends as far into the file system as is necessary to maintain the address of each free data block in the file system. A list block contains some number of addresses of free blocks (usually 50), together with the address of a block that contains addresses of more free blocks (if there are any). Eventually, the address of each free block in the file system will be contained in one of these free list blocks.

Unlike System V and some earlier Berkeley releases, the IBM/4.3 system prefers to keep the superblock structure static; otherwise the redundant copies would need updating whenever the original superblock changed. As a result, the beginning of the free list is not kept in the superblock. Rather, as indicated, the information on free data blocks is kept in a free bitmap.

You should note that this bookkeeping information (superblock copy, i-nodes, free bitmap, etc.) is not always stored at the beginning of a cylinder group. If it were, then it would all be lost if the top platter of a disk were rendered unusable, thereby destroying the redundant copies of the superblock. To protect against that possibility, the bookkeeping information is stored at varying distances from the beginning of the cylinder group. This scheme ensures that not all superblock copies will be found on the top platter.

8. Creating File Systems

Once you have decided how you wish to use the available disk partitions, you may then proceed to create file systems in some of those partitions. The program `newfs` is provided to create new file systems. For instance, to create a file system in the disk partition `hd1g`, give the command:

```
# newfs hd1g
```

This form of the command will create a file system in the named disk partition according to various default parameters governing the number of i-nodes, fragments, cylinders per cylinder group, and so on. Some of these values are obtained from the entry for the appropriate disk type from the file `/etc/disktab`. If you wish to alter the default behavior of `newfs`, you may provide certain arguments. For example, the command:

```
# newfs -b 8192 -f 1024 hd1g
```

requests the creation of a new file system in the named partition having a block size of 8192 bytes (twice the minimum block size) and a fragment size of 1024 bytes. At file system creation time, you may also override the default values for the size of the file system, the number of tracks per cylinder, the threshold of free space, the sector size, and several other parameters. See the manual pages for `newfs`(8) for a complete list of the tunable parameters.

If you are inexperienced with creating file systems or uncertain that you actually want to make a file system, you can run the `newfs` command with the **-N** option, which will cause `newfs` to print out the relevant file system parameters without actually creating the new file system. (This is rather like running `make -n` to see what actions would be performed without actually doing them.)

9. Creating Swap Areas

If your system contains more than one disk, you may wish to assign the system paging and swapping activities to two or more of the disks, thereby increasing performance by "interleaving" the swap areas. To do this, you must first reconfigure the operating system by modifying the system configuration file. For example, you can indicate that the system be able to swap on both `hd0` and `hd1` by including the following line:

```
config  vmunix root on hd0   swap on hd0 and hd1
```

Then you must remake and reinstall the kernel, as explained in detail in Chapter 3.

Next you must edit the file `/etc/fstab` to indicate which partitions on the specified disks are to be used for swapping and paging. Normally the `b` partition is reserved for swapping and paging, so the appropriate `fstab` lines would look like this:

```
/dev/hd0b::sw::
/dev/hd1b::sw::
```

The entry '`sw`' indicates that the relevant partition will be dedicated to swapping and paging. There is nothing sacred about the `b` partition: you may establish swapping areas in partitions other than the `b` partition, and you may use the `b` partition to hold a normal file system.

Nonetheless, we strongly recommend that you follow normal practice and reserve the b partitions on all disks as swapping and paging areas.

So far you have reconfigured your kernel so that paging and swapping are possible on more than one device and you have modified the file /etc/fstab to indicate which partitions on those disks are to hold the swap areas. You have not however actually enabled the additional swap areas. In order to allow the system to boot from a single disk, the system begins by restricting all paging and swapping activity to a single drive. To specify that interleaving is actually to take place, you must execute the swapon command. For example, to turn on the newly-created swap area, give the command:

```
# /etc/swapon -a
```

The -a option causes all devices marked as swap devices in the file /etc/fstab to be made available. Normally this command is included in the multi-user initialization script, /etc/rc.

10. Using an Entire Disk for One File System

Recall that the c partition on a disk is used to access the *entire* physical disk, including the cylinders at the beginning and the end of the disk that are normally reserved by the system for boot programs and bad sector information. Since the system reserves these cylinders for its own use, it is not recommended that you create a file system that occupies the entire c partition. On disks of certain types, doing so would cause important information to be overwritten.

To create a file system that fills the available space on a single disk, you should use the minidisk utility to create a new a partition that extends from the beginning of the a partition to the end of the g partition, as illustrated:

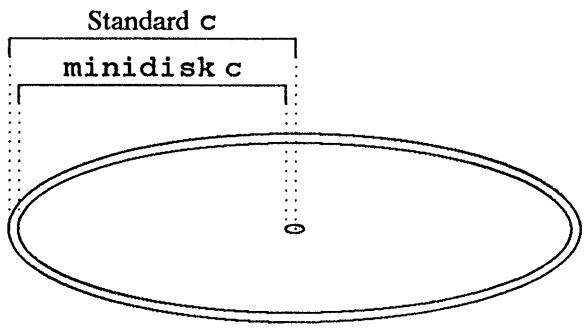


Figure 7-16: Using An Entire Disk for A Single File System

To repartition a disk, you must invoke the minidisk option from the standalone utility shell, reassign partitions, and then reboot your system. The necessary steps are as follows:

(1) Reboot the system:

```
# /etc/shutdown -r now
```

(2) Enter the standalone utility shell:

```
hd(0,6)stand/sautil
```

(3) Select the minidisk option:

```
9
```

(4) Select the disk you want to devote to one file system. In this example, we shall use the third hard disk, hd2:

```
        hd(2,2)
```

(5) If necessary, reinitialize the `minidisk` directory.

(6) Reboot the system.

It is possible to create a file system in the `c` partition of a disk which is not of type `hd70e`, as long as the bad block forwarding table is not too large and the reserved regions of the disk are not used. If you create the file system using the `newfs` program, these restrictions will automatically be enforced. Nonetheless, the procedure outlined above using the `minidisk` utility is the recommended way to place a single file system onto an entire disk.

11. Mounting File Systems

Once a file system has been created, it may be added to the existing directory hierarchy by using the `mount` command. Merely creating a file system in a disk partition does not make it available to the system; the system also needs to be told where in the directory hierarchy the file system is to be located and what permissions the users of the system are to have on the file system. A file system can be mounted *anywhere* in the hierarchy of files, provided that the directory where it is mounted (the "mount point") is empty before the mount occurs. For example, to mount the file system created in partition `hd1g` at the mount point `/usr/src`, run the following command:

```
# mount /dev/hd1g /usr/src
```

The directory `/usr/src` must be empty before this command will succeed. A standard mount point is available on most IBM/4.3 systems in the directory `/mnt`. If nothing is located in that directory and no other file system is mounted there, then you can attach a file system to the directory hierarchy by executing the command:

```
# mount /dev/hd1g /mnt
```

If the message:

```
mount: Device busy
```

appears, then either the named device is already mounted, or the mount point is not empty, or else a user is in that directory. The directory `/mnt` is normally used to mount file systems after they have just been created in order to see that everything went okay, and also to mount file systems temporarily for administrative purposes. You should not normally mount file systems there for other purposes.

To make sure that a file system was properly mounted, try to access a file or directory within the file system. For example, perform a `cd` to the mount point and run `ls`.

Once you have created file systems in disk partitions other than the root partition and put files into those systems, you should modify the file `/etc/fstab` to include that information. The file `/etc/fstab` maintains a list of all file systems known to the system and contains information about the associated device, the usual mount point, the type of file system, the frequency of dumps, and the order in which file system checks are to be done (with `fsck`). The format of this file is as follows:

device-name : *mount-point* : *type* : *freq* : *pass*

Format 7-1: `/etc/fstab`

For a complete explanation of the fields, refer to the manual page `fstab`(5). A few lines from a typical `/etc/fstab` might look like this:

```
    /dev/hd0a:/:rw:1:1
    /dev/hd0g:/usr:rw:1:2
    /dev/hd1h:/usr/src:rw:1:2
```

To mount all currently unmounted file systems listed in the file `/etc/fstab`, execute the command:

```
    # mount -a
```

This command is generally placed into the multi-user start-up file, `/etc/rc`, to mount all available file systems automatically at multi-user initialization time.

12. Unmounting File Systems

To remove a mounted file system from the system is to "unmount" it. This is the reverse process from mounting the file system. Unmounting does not destroy any files or directories located in the file system; it merely makes them unavailable until the file system is mounted once again. A file system may be unmounted by invoking the `umount` command (note the strange spelling). For example, to unmount the `/usr` file system, give the commands:

```
    # sync
    # sync
    # umount /dev/hd0g
```

(assuming that the `/usr` file system was originally mounted on the special device `/dev/hd0g`). The `sync` commands are recommended to flush any output buffers that may contain disk updates. To unmount all currently mounted file systems, you can run the command

```
    # umount -a
```

This will cause `umount` to consult the file system data base file `/etc/fstab` and to unmount all mounted file systems listed therein. If, when you issue an `umount` command, the message:

```
    umount: Device busy
```

appears, then either some user is located in the mount directory (or one of its subdirectories) or there is an open file in the named file system. If you want to unmount the file system, determine whether some user is using it, or whether a process has a file open in the file system. You may need to terminate the process to free up the file system.

13. Maintaining File Systems

The IBM/4.3 file system is designed to achieve maximum throughput on a properly-balanced system. You can help maintain optimal file system performance by taking steps to balance the disk load upon installation and by closely monitoring the use of the system. If certain file systems (such as those containing spooling directories) tend to fill up occasionally, you may consider adding disks or moving the affected file system to a larger disk partition (if one is available). As explained below, a file system that continually operates at near capacity will tend to increase file fragmentation and slow disk transfer operations.

Aside from balancing the disk usage load so as not to overburden a particular disk or disk partition, the most important administrative concerns for maintaining file systems are to ensure that the disks are periodically updated, to minimize disk fragmentation, and to check the integrity and consistency of the file systems on a regular basis. As with most other administrative tasks, you can suitably configure the system so that these tasks are perform automatically. For example, a simple but effective way of helping to ensure a reasonable level of file system integrity is to run the `update` program. The `update` program executes the `sync()` system call once every 30 seconds, thereby updating all file systems currently mounted. So even if you should suffer a system crash, you are guaranteed that the file systems will be reasonably sane. If no hardware damage was done by the crash, then the disk file systems will be at worst 29 seconds "old", and this much inconsistency can usually be fixed easily by running `fsck`, described below. The `update` program is usually launched once, at multi-user initialization time, by placing the following line into the file `/etc/rc`:

```
/etc/update
```

You should not run the program directly.

13.1. Disk Fragmentation

You have seen that the data blocks that collectively make up a file are not necessarily stored contiguously on a disk. When a file is created, the kernel consults the bitmap of free blocks and selects as many blocks as are necessary to hold the file, placing the addresses of the file's data blocks into the file's i-inode. The kernel attempts to reduce disk fragmentation as much as possible, but it simply doesn't matter to the user or to the kernel if the data blocks are next to one another or are widely separated, since it can find them just as easily in either case. Fragmentation of a file's data blocks is a concern only because it may take considerably longer to read those data blocks if they are not located in contiguous sections of the disk.

The IBM/4.3 file system is commonly called a "fast file system" because it implements a disk block management scheme that keeps data transfer rates nearly constant over time. In other words, data transfer rates are not generally sensitive to file system fragmentation through continued use. Instead, data transfer rates are dependent on the total amount of free space in the file system. If a file system has plenty of free space in it, then the disk allocation algorithms of the IBM/4.3 system will avoid disk fragmentation. On the other hand, if there is very little free space remaining on a particular file system, then the system has no choice but to take whatever free data blocks remain; as a result, very great file fragmentation may occur, and data transfer times will increase.

The IBM/4.3 system avoids this kind of performance degradation by making sure that you never fill any of your disks. Under the default configuration, once a disk becomes about 90 percent full, the system simply "pretends" that the disk is full and disallows further allocation of disk blocks to system users (except `root`). Since the total amount of free space in an IBM/4.3 file system can never fall below a certain threshold, the system is always assured that it can find reasonably contiguous blocks to store a file in. So file access times should never suffer, as long as you do not exceed the threshold.

One consequence of the disk allocation scheme of the fast file system is that you should never need to reorganize your disks in order to reduce file fragmentation. If you have administered previous UNIX-based systems, you know that over time a file system will grow more and more fragmented with heavy use, and that response time will eventually suffer. In those systems, special utilities had to be provided to reorganize a disk. Or, if such utilities were not available, the only recourse was to make a full backup of the file system onto secondary media, create a new file system in the disk partition, and then restore the files from the backup. In either case, file system reorganization was a fairly laborious process that would monopolize the disk for quite a while. As you can see, the "wasted" 10 percent of an IBM/4.3 file system is a small price to pay for never having to do this kind of reorganization.

You can in fact modify the free space threshold by running the `tunefs` command on the desired file system. For example, the command:

```
# tunefs -m 8 /dev/hd1g
```

specifies that the file system in `/dev/hd1g` should have the free space threshold reduced to 8 percent of the total available space. (Make sure that the file system is unmounted, if possible, before running this command.) It is possible to set this value to zero, and hence make the *entire* file system usable, but you will of course increase file fragmentation and reduce throughput by doing that. On most systems, the throughput will be reduced by a factor of about 3 over the performance attained at a 10 percent threshold. Accordingly, it is not recommended to set the free space threshold to 0 percent. If a particular file system holds files that are reasonable large, you may reduce the free space threshold to as low as 3 percent without adversely affecting performance. On a file system with lots of small files (such as the file system housing the USENET news articles), a value of 8 to 10 percent appears to be safest.

14. Checking File Systems: Fsck

It is absolutely essential to check and ensure the integrity of all of the file systems on your machine. A file system can become corrupted in a number of ways, most commonly by a system crash. If, for example, a file is being written to the hard disk when the system crashes, the disk is practically certain to contain a mangled version of the file. In addition, the bitmap of free blocks in the file system will very likely be incorrect, since it did not get updated. In such cases, the file system is said to be *inconsistent*, and it is the duty of the system administrator to make sure that the file system inconsistencies are detected and corrected, if at all possible. It is very dangerous to continue to operate on a file system that has become inconsistent, since additional file system activity is likely to exacerbate those inconsistencies.

The IBM/4.3 system contains a program, `fsck`, that is designed to help find and correct file system inconsistencies. It operates by inspecting redundant copies of file system related information and by trying to resolve any conflicts that it discovers among such copies. The file system inconsistencies checked by `fsck` are the following:

- Data blocks claimed by more than one i-node, or by an i-node and the free list.
- Data blocks claimed by an i-node or the free list that are outside the range of the file system.
- Incorrect link counts.
- Size of directory is not in the proper format.
- Bad i-node format.
- Data blocks not claimed by any i-node or by the free list.
- File pointing to unallocated i-node.
- I-node number out of range.
- Excessive number of i-node blocks.
- Bad free block list format.
- Total free block or free i-node count wrong.

The `fsck` file checking program runs in two modes: non-interactively, when it is typically run by the system after a normal boot, and interactively, when it is run by the operator, to identify and correct unexpected inconsistencies, or following a system crash.

14.1. Running Fsck Non-Interactively

When IBM/4.3 is brought up, `fsck` should always be run to check the consistency of the file systems. Doing so helps to ensure the integrity of the file systems; if any inconsistencies are found, they must be corrected before proceeding.

Most of the time (hopefully), `fsck` will not find any inconsistencies. In that case, the output will look something like this:

```
# fsck /dev/hd1g
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
6076 files, 32172 used, 10789 free
              (61 frags, 2682 blocks, 0.1% fragmentation)
```

For explanations of `fsck`'s diagnostics, see "Fsck – The UNIX File System Check Program" (SMM:5).

When run with the `-p` option by the system during a normal boot, `fsck` not only checks for file inconsistencies; it also corrects the following inconsistencies in the file system that it knows arise from an unclean machine shutdown. These include the following:

- Unreferenced inodes
- Link counts in inodes that are too large
- Missing blocks in the free list
- Blocks in the free list that are also in files
- Counts in the super block that are wrong

For each corrected inconsistency, `fsck` prints one or more lines identifying the file system on which the correction took place and the nature of the correction. After successfully correcting inconsistencies, `fsck` prints the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If `fsck` encounters inconsistencies other than those listed above, it exits with an abnormal return status, leaving the system running in single-user mode, and the boot fails. The operator should then run `fsck` interactively to find and correct the inconsistencies.

14.2. Running Fsck Interactively

The operator should run `fsck` interactively if `fsck` finds unexpected inconsistencies (those that it cannot correct) following a system boot, and after a system crash. In this mode, `fsck` lists each problem and suggests the corrective action that it thinks is most likely to restore the file system to a sane state. The operator can then decide whether or not the suggested correction should be made.

15. File System Management Tips

- *Never* run the file system checking program `fsck` on Andrew File Systems. If you do, you will irreparably damage the Andrew File Systems. An alternate file system consistency checker, `vfsck`, is provided with the IBM/4.3 system to provide the equivalent functionality for Andrew File Systems.
- *Never* run `fsck` on a mounted file system other than the root file system (for which you have no choice, since it is always mounted). Running `fsck` on a mounted file system may cause numerous unreferenced file errors relating to the presence of pipes and other objects in the file system.

CHAPTER 8

Backing Up and Recovering Files

1. Introduction

It is absolutely essential that the system administrator maintain a rigorous backup schedule so that current off-line copies of all system and user files are available at all times. Files and directories can easily become corrupted through abnormal system activity (such as a system crash or power failure), or users may accidentally remove files that are important to them. Even a normally careful superuser can destroy an entire file system by mistyping an otherwise innocent command. If a complete backup of all file systems is available, however, the necessary files and directories can be retrieved easily and quickly using IBM/4.3 recovery commands.

This chapter describes the backup and recovery utilities available on the IBM/4.3 system and details typical strategies for using those utilities. Unfortunately, backing up and recovering files is one of the least standardized procedures confronting the system administrator, owing largely to an abundance of utilities that have in the past been used for this purpose. The currently recommended utilities for backing up and restoring files on the IBM/4.3 system are the two programs `dump` and `restore`. It is possible, however, that your site may choose to substitute different backup and recovery procedures for the ones illustrated here, or it may elect to provide a slightly different interface (such as a customized shell script) to these utilities. If either of these possibilities is the case, then you will need to consult local documentation or local experts in addition to this chapter.

If you do employ the programs `dump` and `restore`, then you should log in as `operator`, not as `root`, while performing system backups. There are two main reasons for this. First, the `operator` login account has read permissions on all disks attached to the system, but it does not have write permissions. This allows someone logged in as `operator` to perform backups without having full superuser privileges. By performing backups with reduced privileges, the operator can avoid damaging file systems through uncareful or inadvertent command execution. The second reason for logging in as `operator` to manage backups and recoveries is that the `dump` program and the message daemon `syslogd` are configured to notify the operator in case assistance is needed or errors arise. For example, if one dump tape becomes filled and another is required to continue the backup procedure, the `dump` program will write a message to the terminal screen of anyone logged in as `operator`. If you perform system backups while logged in as `root`, you may not see important diagnostic messages from the `dump` program.

2. Full and Partial Backups

Fundamentally, there are just two types of backup that you can perform on your IBM Academic Operating System 4.3 system, *full* and *partial.* A full backup creates a copy of every file and directory located on every file system on your machine. Such backups are entirely non-selective; they copy everything on the system to the backup medium. A partial backup, on the other hand, is simply any backup that is not full. For example, you may need to perform a partial backup when you have to remove a user from the system and want to make a copy of all the files and directories owned by that user. Or, more typically, you may perform a partial backup to save a copy of every file that has changed since a certain date. This kind of partial backup is known as an *incremental* backup because the file system is backed up in small pieces or increments. Maintaining a series of incremental backups allows the system administrator to have current copies of all files on the system while using a minimum number of backup tapes or floppy diskettes. In addition, it generally takes much less time to perform incremental backups than full backups.

A correctly-designed backup schedule, however, will incorporate both types of backups at different times. For instance, a relatively large computing installation may schedule full backups every month (or even every two weeks) and incremental backups every day. On the other hand, a small system with relatively static file systems might need to perform full backups only once every two or three months and incremental backups just once a week. Until you are fully acquainted with the usage patterns and needs of your particular site, you should follow the backup schedules described below.

3. Backup Media

In its broadest possible meaning, to *backup* a file (or an entire file system) is simply to create another copy of that file in a place where it may be found in case the original copy becomes lost or damaged. Accordingly, a file may be backed up in one or more different ways. The simplest method of backing up a file is just to copy the file into another file in the same directory:

```
    % cp ch.01 ch.01.bak
```

This method has the advantages that any user can "backup" files in this way, and it is easy to "restore" the original file if necessary (since it is right there in the file system). Some disadvantages of such a backup scheme, however, are obvious: if the disk containing the files is for some reason damaged, then both the original file and the copy will be lost. Or if the user mistakenly types the command:

```
    % /bin/rm -f ch.01*
```

then both the original and its copy will be removed. In a nutshell, this method of backing up files sins by putting all its eggs in one basket. Nonetheless, the practice of making a copy of an important file as a temporary "backup" is one employed by most IBM/4.3 system users and (within limits) it is a practice that you should encourage, if for no other reason than to save yourself time restoring files off of backup media.

Perhaps the worst disadvantage of this backup scheme is simply that it can consume large amounts of disk space, which is usually one of the most precious resources on a computer system. As the system administrator, you will need to develop a different strategy for backing up important files, directories, and file systems. Instead of making copies of files in the file systems, you must institute some method of creating copies of file systems that are accessible even if the original file system is irreparably damaged. Typically, four main methods of performing system-wide backups are employed, as illustrated in the following diagram:

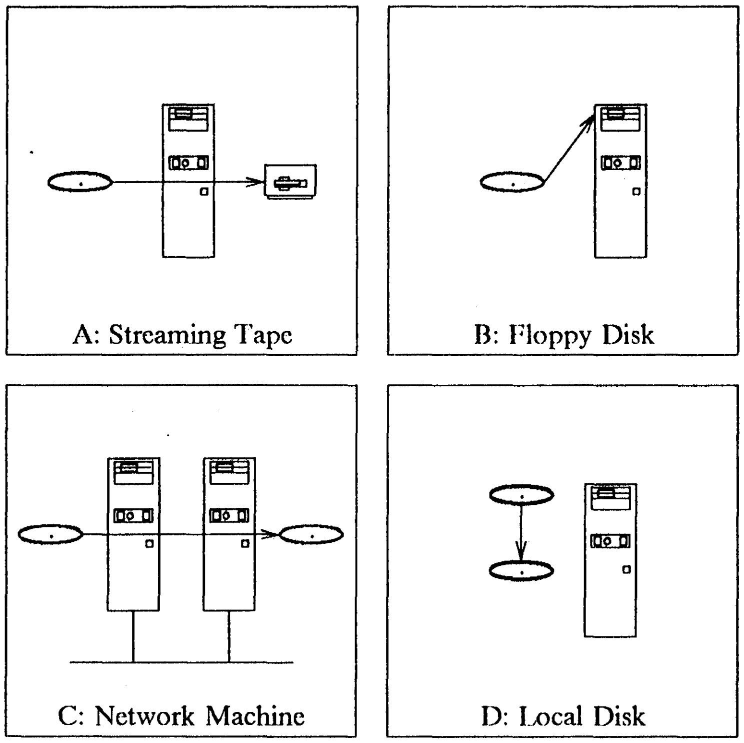


Figure 8-1: Four Common Backup Methods

First, the system administrator may backup files and file systems onto streaming tape cartridges (Figure A). This method is fast and efficient, but you must have a tape drive attached to your machine. If you do not have a streaming tape drive available, you may decide to back up files onto floppy diskettes (Figure B). This option is not nearly as quick as with streaming tape and may require many floppy diskettes to back up all the files located on your system. If your machine is attached to a local area network, it is also possible to make a backup copy of your files by copying them across the network onto a remote machine (Figure C). The files can then remain on the remote disk, or they may be transferred onto streaming tape or floppies from the remote machine. If your local machine has sufficient hard disk space, you may copy all or part of a hard disk onto another hard disk partition on the same machine (Figure D). This is usually the fastest backup method, but the backup files may not be recoverable if the machine is damaged and cannot be booted. All four of these backup methods will be explained in the remaining sections of this chapter.

Which of these four common methods you employ depends heavily on the type of hardware you have available and the amount of time you can devote to file system backups. If the value of the data on your system warrants extra safety precautions, you may choose to employ several of these methods in combination. Conversely, if the data on some file system are of no particular importance or can be easily resurrected in case of corruption or loss, then you may decide never to backup that file system. This happens generally only in cases where an entire file system is devoted to holding temporary or scratch files (such as `/tmp` or `/usr/tmp`), to line printer spooling, or to the storage of USENET news articles.

4. Backup Scheduling

It is recommended that you use the dump program for both full and incremental backups. The dump program is designed to copy to the backup medium all files in a file system changed after a certain date; it does this by relying on the notion of a *dump level* and by maintaining an on-line record of previous dump dates and levels. By definition, a level 0 dump constitutes a full dump. For values of *n* greater than 0, a dump taken at level *n* has the effect of backing up all files modified after any previous dumps at levels lower than *n*. So a level 1 dump will backup all files modified since the previous level 0 dump. Similarly, performing a level 2 dump will backup all files modified since the previous level 1 or level 0 dump, whichever occurred more recently than the other. And performing a level 3 dump will backup all files modified since the previous level 0, 1, or 2 dump, whichever occurred most recently. Hence, the higher the dump level you request, the fewer files you are likely to select, since many files will have been dumped at lower levels.

To see more concretely what this means, imagine that you have just installed the IBM/4.3 system software onto your machine. It is good practice to make a complete backup of all the file systems right away, so that you can archive the installation tapes or return them to a central administrative authority. (Alternatively, you may wish to edit some of the various configuration files on the system before performing this complete backup.) Since your machine has not yet been backed up, you should do a level 0 dump on all file systems, which requests a full dump. After a day's computing activity, some of those files will have been modified. If you then request, for example, a level 3 backup, you will select all files modified after the most recent level 0, 1 or 2 backup. The most recent backup was a level 0 dump, so you will get all files modified in that day's work. On the next day, you might perform a level 2 backup. That will select all files changed after the original level 0 backup, since there has (yet) been no level 1 dump. It is important to note that this level 2 dump will select all the files selected by the level 3 dump, in addition to any that have been changed in the day following the level 3 dump.

Now let's continue this backup process for several more days. On the day following the level 2 backup, you might perform a level 5 backup. That selects all files modified since the most recent backup at a lower level, which in this case is just the files modified since the preceding dump. On the next day, if you perform a level 4 dump, you will get all files modified after the level 3 dump. At the end of the week, you will want to perform a level 1 dump, which will backup all files changed since the original level 0 dump.

This overlapping sequence of dump levels may seem strange until you realize that you must maintain three independent sets of dump media, one for monthly dumps (full or level 0), one for weekly dumps (level 1), and one for daily dumps (all other levels). As you can see, most of the files backed up onto the weekly set of tapes will already have been backed up onto the daily set of tapes. Similarly, most of the files backed up onto the monthly set of tapes will already have been backed up onto the daily and weekly set of tapes. Because of this, the backup scheme described here is sometimes called a "double redundancy" backup scheme. The redundancy inherent in this scheme provides a certain measure of safety in case a streaming tape or floppy should prove defective.

The recommended sequence of full and incremental backups described above is illustrated in the following figure:

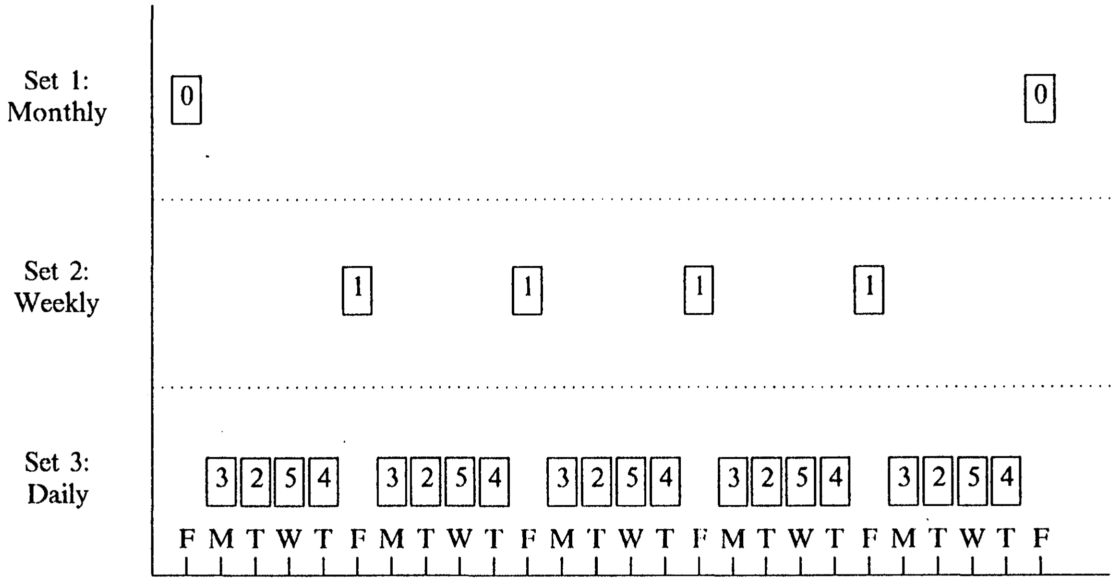


Figure 8-2: The Recommended Backup Schedule

The horizontal axis lists in sequence the weekdays in a given month, and the vertical axis shows the three sets of backup media. The box for any particular day shows which set of tapes to use and what level dump to perform. As mentioned, the month begins with a level 0 dump, a full backup of the entire file system. Ideally this dump would take place at a time of very little system activity, perhaps on a Friday afternoon. Then the following Monday, a level 3 dump is performed using the third set of tapes. The next day, Tuesday, a level 2 dump is performed using the same set of tapes as was used for the level 3 dump. and so on until the following Friday, when a level 1 dump is performed, using the second set of tapes. Eventually, a level 0 dump is performed at the beginning of the new month.

As indicated, monthly dumps are always done at level 0 and weekly dumps are always done at level 1. The level of the daily dumps varies from day to day and should follow a modified "Towers of Hanoi" sequence:

```
3 2 5 4 7 6 9 8 9 9
```

Dumps should always restart at level 3 after a level 0 or level 1 dump, and you should perform weekly dumps often enough that the 9's in this sequence are never reached. The dump schedule depicted in the diagram above is an ideal one that you should strive to maintain, if at all possible. Whatever schedule you adopt, it is important to heed the following two pieces of advice:

- Perform all dumps when file system activity is at a minimum. If you perform a dump on a file system that is actively changing, you risk creating inconsistent dump tapes. The `dump` program makes several passes through a file system, so if things change between passes, the dump may already be out of date. Ideally a file system should be unmounted when backed up.
- Make sure that a file system is "clean" before you make a backup copy of it. The best way to do this is to run `fsck` on a file system before running `dump`.

5. Backing Up Files and Directories

When you want to back up a file system using the `dump` program, you must indicate which level to use when dumping the file system. To indicate to the `dump` utility which level to dump at, you must supply a key character. For example, to perform a complete (level 0) backup, you might issue the following command:

```
# dump 0sfu 2600 /dev/st0 /dev/rhd0g
```

The s key character indicates that the next argument is to be understood as the number of feet in the backup tape. The default is 2300, so to accommodate the streaming tape, the argument 2600 was supplied. Also, the f key character indicates that the corresponding argument is to be taken as the dump device, in this case /dev/st0 (for the streaming tape device). The command syntax is admittedly painful, but users of tar on other systems should have no trouble understanding it. Refer to the manual pages dump(8) for complete details on the available key letters and their meaning.

The dump utility knows which files need to backed up by maintaining a record of all previous backup activity in the file /etc/dumpdates. So, for example, when you request a level 4 dump of a particular file system, the dump program scans /etc/dumpdates to determine the date and time of the most recent dump of that file system at level 0, 1, 2, or 3. When the most recent relevant backup time is found, dump then scans the appropriate file system to see which, if any, files in it have been modified or created since that time. Any such files will then be dumped to the standard output of the dump command (usually a streaming tape drive or a floppy drive). Finally, when dump is finished backing up the selected files, it writes the current date and time into the /etc/dumpdates file, if the u key character was specified, as in the example above. This file must be owned by the user operator if the updating is to complete successfully. In addition, the file should be writable *only* by the owner of the file, or else a careless user may corrupt the file. The format of the /etc/dumpdates file is this:

file-system *level* *date*

Format 8-1: /etc/dumpdates

For example, after a level 0 dump of the two file systems on the hd0 drive, the top of /etc/dumpdates might look like this:

```
/dev/rhd0a          0 Thu Sep 17 12:45:44 1987
/dev/rhd0g          0 Thu Sep 17 12:52:07 1987
```

Recall that the hd0a partition usually contains the root file system and the hd0g partition usually contains the /usr file system. Most systems will have additional file systems available on other disk drives, so there will be more lines in the /etc/dumpdates file like the two listed above.

You can determine which file systems need to be backed up by running dump with the W or the w key character. If invoked with the W key letter, dump will report to the operator which file systems need to be dumped. It prints out the most recent dump date and level for each file system listed in the file /etc/dumpdates. In addition, the W key letter causes dump to highlight those file systems that need to be dumped by prefixing the relevant output lines with the character >. The w key letter causes dump only to report file systems needing to be backed up:

```
# /etc/dump w
Dump these file systems:
  /dev/rhd0a    (     /) Last dump: Level 0, Date Thu Sep 17 12:45
  /dev/rhd0g    (  /usr) Last dump: Level 0, Date Thu Sep 17 12:52
```

Note that the file /etc/dumpdates will not contain accurate information about the most recent dumps unless you always perform dumps with the u key letter telling dump to update that file. If you forget to include that key letter, you can update /etc/dumpdates manually, since it is in a human-readable and -editable format.

Once you have determined which file systems to dump and which level to dump them at, you can proceed to actually dump those file systems. The following three sections describe in detail how to dump file systems onto streaming tape cartridges, onto floppy diskettes, and across a local area network.

5.1. Using Streaming Tape

The IBM/4.3 system supports streaming tape backups using the IBM 6157 Streaming Tape Drive, which accommodates 45-Megabyte cartridges (for example, the DC300 XL/P data cartridge from 3M). Because of the large capacity of such cartridges, you should be able to backup each file system onto a single cartridge (unless you have created non-standard disk partitions). To back up your system onto streaming tape, follow these steps:

(1) Erase each of streaming tape cartridges by inserting it into the streaming tape drive and executing the command:

```
# mt -f /dev/rst0 erase
```

(2) Make sure that you are logged in as `operator`. If necessary, execute the following command:

```
# su operator
%
```

As indicated, the shell prompt should change from the pound sign (indicating superuser privileges) to the prompt of the user `operator`, which is by default the percent sign.

(3) Insert one of the newly-erased tapes into the streaming tape drive and type:

```
% dump 0suf 2600 /dev/st0 /dev/rhd0a
```

This command requests a full dump of the root file system (situated in disk partition `/dev/rhd0a`). Be sure to change the key letter indicating the dump level to the appropriate value. If everything goes okay, you will see messages like the following:

```
DUMP: Date of this level 0 dump: Fri Feb 26 13:32:43 1988
DUMP: Date of last level 0 dump: the epoch
DUMP: Dumping /dev/rhd0a (/) to /dev/st0
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 6624 tape blocks on 0.25 tape(s).
DUMP: dumping (Pass III) [regular files]
DUMP: dumping (Pass IV) [directories]
DUMP: DUMP: 6624 tape blocks on 1 tape(s).
DUMP: DUMP IS DONE
DUMP: tape rewinding
DUMP: level 0 dump on Fri Feb 26 13:34:22 1988
```

If the file system you are dumping does not fit completely onto a single streaming tape, then the last four lines printed out in the example above will not appear; instead, you will see messages similar to the following ones:

```
DUMP: 60.46% done, finished in 0:04
DUMP: tape rewinding
DUMP: Change tapes: Mount tape #2
DUMP: NEEDS ATTENTION: Is the new tape mounted and ready to go?:
        ("yes" or "no")
```

At this point, you should change tapes and then type '`yes`'. The `dump` program will continue:

```
DUMP: Tape 2 begins with blocks from ino 8701
```

Notice that the `dump` program has indicated the number of the first i-node on the new tape. You should, if possible, record this number for future reference (preferably on the tape case). The dump program writes on a tape in order of increasing i-node numbers, and the restore utility restores by i-node number. Continue following the directions from dump until the dump is complete.

(4) Repeat step (3) for each file system that you wish to back up.

As you can see, the dump program requires explicit operator intervention when the end of one tape is reached so that a new one can be mounted. Assistance from the operator is also needed when the dump is completed and whenever it encounters an excessive number of tape or disk errors. (Note that dump will silently absorb up to 32 such errors.) dump always writes these requests for action on the controlling terminal of the person who launched the dump. In addition, you can include the n command line key letter in order to have dump write its requests for action on the terminal screens of *all* persons currently logged in under the operator user name.

5.2. Using Diskettes

The steps involved in backing up file systems onto floppy diskettes are entirely analogous to those involved when using streaming tape. The main difference between using floppies and using streaming tape is that individual floppies hold much less data than streaming tape, so you will need to have a fairly accurate idea of how much data you need to back up and arrange to have plenty of floppy disks on hand. Also, you must make certain that the floppy diskettes are formatted before you use them in a dump.

To back up a file system or selected files onto floppy diskettes, follow these steps:

(1) First of all, estimate how many floppy disks will be required to contain all the data that you wish to archive onto floppies. Each high-density floppy diskette holds 1.2 Megabytes of data, or 1,228,800 bytes. If you are backing up individual files in an incremental dump, you can simply estimate the total number of bytes by looking at a long listing of the files and adding up the fourth column. If you want to back up an entire file system, run the df command to determine how many blocks are used in that file system. Remember that df reports the number of 1K (1024 byte) blocks in the file system, so to determine how many floppy disks are required to hold it, use the following formula:

$$\text{number of 1.2M floppies} = (\text{blocks consumed} \times 1024) / 1228800$$

If the result of this calculation is not integral, then round the result up to the next highest integer. That integer indicates (approximately) how many floppies are needed.

(2) Next, make certain that all the diskettes are properly formatted. Unless you are absolutely certain that the diskettes you are about to use are already formatted, insert each one into the upper floppy disk drive and give the command:

```
# fdformat -h /dev/rfd0
```

When the system asks whether to proceed with formatting, type 'yes' and hit the return key. The system will then format the diskette. Repeat this step for each diskette that needs to be formatted.

(3) Make sure that you are logged in as operator. If necessary, execute the following command:

```
# su operator
%
```

As indicated, the shell prompt should change from the pound sign (indicating superuser privileges).

(4) Now dump the selected file systems onto diskette. For example, to dump the `/usr` file system onto diskette, type:

```
% dump 0sfu 70 /dev/rfd0 /dev/hd0g
```

Notice that this command is exactly like that specified in step (3) for dumping to a streaming tape, except that the capacity of the backup medium must be specified correctly. As above, the `dump` program will pause and prompt you for a new diskette when one becomes full. Be sure to label each diskette with the name of the file system and the sequence number.

(5) Repeat step (4) for each file system that you want to backup.

5.3. Network Backups

It is possible to perform backups of file systems across a local area network, so that in theory you need just one streaming tape drive (or other backup device) for your entire network. To do a remote backup, you should be logged in as `operator` on the machine which you want to backup. Instead of running the `dump` command, however, you will run a very similar command, `rdump`, to route the files to be backed up over the network to a remote machine.

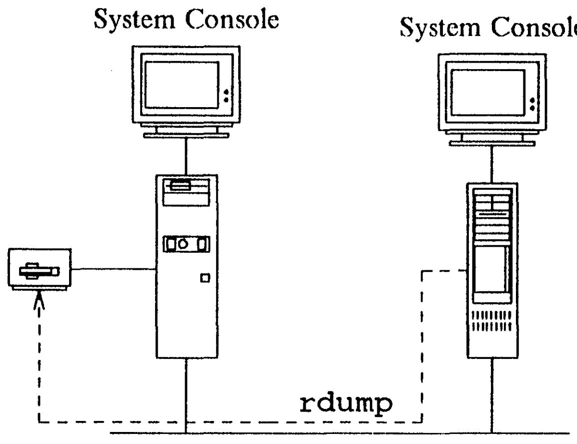


Figure 8-3: Dumping Across A Local Area Network

The utility `rdump` is identical to `dump` except that the `f` key letter should be specified and the file to be dumped to should be of the form:

machine : *device*

This indicates that the specified device on the specified machine is to be used as the standard output of the dump. For example, to make a full (level 0) backup of the local `/usr` file system onto a streaming tape drive attached to the host named '`tuna`', you would give the command:

```
% rdump 0sfu 2600 tuna:/dev/rst0 /dev/rhd0g
```

Before running this or a similar command, make sure that the requested tape drive is not already in use. Of course, you will need to have physical access to the machine `tuna` and the attached tape drive in order to insert and change tapes.

6. Restoring Files and Directories

Once you have created a set of dump tapes using the utility `dump` as described above, you can list the contents of the tape and restore selected files and directories from it using the `restore` utility. `restore` is designed specifically to be used in tandem with the `dump` program and it will not allow you to restore files from archives created using other IBM/4.3 utilities (unless, of course, they call `dump` in some way). Note that you should always run the `restore` program while logged in as the superuser, `root`. If you try to restore files and directories while logged in as `operator`, you will probably be unable to create the necessary files or directories in the file system, since the `operator` account has minimal write access to the system disks (even though it can read all of them).

The command-line syntax of `restore` utility is (not surprisingly) very much like that of the `dump` command. In particular, the following key letters are available to control `restore` functions:

Letter	Description
r	Read the dump tape into the current directory.
R	Restore using a particular tape from a multi-volume dump. This allows `restore` to be interrupted and then restarted.
x	Extract the named files from the backup medium. If no argument is supplied, then the root directory on the tape will be extracted, thereby causing all files on the tape to be extracted.
t	Give a list of files on the tape. If no argument is supplied, then the root directory will be listed, thereby causing all files on the tape to be listed.
i	Enter `restore`'s interactive mode. A prompt will be printed and the program will take commands from the terminal.

Table 8-1: `restore` Function Letters

As you can see, `restore` has both interactive and non-interactive modes. The interactive mode (described more fully below) allows you to browse through a particular backup volume, see what is contained on it, and build a list of files and directories to be restored. This is most useful when you want to restore only selected files, not an entire tape.

6.1. Using Restore Non-Interactively

There are a number of functions that you can perform while using `restore` in its non-interactive mode. For example, to list the files and directories backed up on a particular streaming tape, insert the tape into the tape drive and then execute the following command:

```
    # restore tf /dev/rst0
```

To obtain a listing of a floppy disk, simply change the device name to the appropriate value (probably '`/dev/rfd0`'). The output from this command will vary according to the actual contents of the tape, of course, but the first few lines will look something like this:

```
    Dump   date: Fri Feb 26 15:41:11 1988
    Dumped from: Fri Jan 30 17:43:44 1988
              2       .
              3       ./lost+found
            864       ./bin
           1371       ./bin/yacc
```

```
        1372        ./bin/lex
        1373        ./bin/f77
```

The files are listed together with their i-node numbers. Usually the output will be quite lengthy, so you may want to redirect the output into a file for later reading or printing:

```
    # restore tf /dev/rst0 > list.1
```

The `x` key letter is used to extract a list of files from a tape. For example, if the file `/usr/bin/f77` was accidently damaged (or removed by an overzealous C programmer), you can restore it by inserting the tape containing it into the tape drive and then typing:

```
    # cd /usr
    # restore xf /dev/rst0 ./bin/f77
```

Normally, `restore` will respond with the following message:

```
    You have not read any tapes yet.
    Unless you know which volume your file(s) are on you should start
    with the last volume and work towards the first.
    Specify next volume #:
```

At this point, you should respond by typing the number of the volume containing the file you want to restore. If there is just one volume in the archive, then simply type '`1`'. The requested file, `/usr/bin/f77`, will now be restored from the tape.

Generally, however, it is unwise to restore files by full (or even relative) path name. One good reason for this is that restoring a file by its full path name will clobber the original file, if it still exists. That makes it impossible to compare the original and the backup copy, unless you first move the original into a safe location. Instead, you can instruct `restore` to restore the file under its *i-number*. That is to say, you can tell `restore` to recreate a given file with a file name that is the same as the i-number of the original file. You do this by including the `m` key letter. For example:

```
    # cd /usr
    # restore xmf /dev/rst0 ./bin/f77
```

Once `restore` successfully completes this operation, you will see a file in the current directory whose name is `1373`, which was the i-number of the original file, `/usr/bin/f77`. Note carefully that the *name* of the restored file is the *i-number* of the original file (the one backed up onto the streaming tape). Of course, the restored file is a file and hence has an i-number of its own. That i-number will be identical to the name of the file just in case the original file does not exist at the time that the copy is restored from tape. If, on the other hand, the original still exists, then the restored file will have a different i-number (since the i-number of the original file is still in use). To complete the restore, you need to move the restored file into its preferred place in the file system. In our current example, you would run the command:

```
    # mv ./1373 ./bin/f77
```

The final main key letter used with `restore` in its non-interactive form is the `r` key letter. This is used to restore an entire backup volume. In other words, running `restore` with the `r` key letter causes the entire contents of a backup tape to be read into the current directory. The `r` key letter should therefore be used only to restore a complete (level 0) dump tape onto a clean file system or to restore an incremental dump after a full level zero restore. This activity is advised only when restoring a complete file system because of damage or file system corruption or when reformatting a disk to change the size of a file system.

For instance, suppose that the `/usr` file system, usually located on hard disk partition `/dev/hd0g`, has become hopelessly corrupted by some hardware failure. The first thing you want to do is run `fsck` to check for inconsistencies. Then you want to create a new file system there and do a full restore from your existing level 0 backup tapes:

```
# fsck /dev/hd0g
# newfs hd0g
# mount /dev/rhd0g /mnt
# cd /mnt
# restore rf /dev/rst0
```

After this sequence of commands is executed, the file system mounted at `/mnt` will look like the `/usr` file system looked at the previous level 0 dump. You then need to restore any incremental dumps you made since that level 0 dump. Collect all those incremental dump tapes and mount them in the order of increasing level, *regardless of the order you actually made them in.* That is to say, if you have a level 1 dump taken after the level 0 dump you just restored, then put that tape in the drive and run the command:

```
# restore rf /dev/rst0
```

If you have a level 2 dump taken after the level 1 dump you just restored (or after the level 0 dump, if there is no level 1 dump), then put that tape in the drive and run the `retore` command. Continue in this way with the level 3, 4, and 5 dumps, and any greater level dumps, if any exist, taken since the level 0 dump. Once you have restored the incremental dump tapes, the new file system will be as current as possible. You may now mount the restored file system at its preferred mount point:

```
# sync
# sync
# umount /dev/hd0g
# mount /dev/hd0g /usr
```

The `/usr` file system has now been completely restored from backup media.

6.2. Using Restore Interactively

To use `restore` interactively, supply the `i` key letter in addition to the name of the tape drive (or floppy drive):

```
# restore if /dev/rst0
```

Once you have executed the `restore` command, you will see the interactive prompt:

```
restore>
```

At this point, you can enter any of the ten available interactive commands. The following commands are available from within `restore`:

Command	Argument	Description
`add`	*file*	Add the current directory or the specified argument to the list of files to be extracted.
`cd`	*dir*	Move into the specified directory.
`delete`	*file*	Remove the current directory (or the argument, if specified) from the list of files to be extracted.
`extract`		Extract from the tape all the files that are currently on the extraction list.
`help`		List a summary of all the available commands.

`ls`	*dir*	List the contents of the current directory or of the directory specified as an argument. An asterisk '*' precedes all directory entries that are currently on the extraction list.
`pwd`	*file*	Print the full path name of the current working directory.
`quit`		Exit from `restore`.
`setmodes`	*file*	Set owner, modes, and times on all files currently on the extraction list.
`verbose`	*file*	Toggle the sense of the `v` key letter. When set, the verbose option list i-node numbers along with file names when the `ls` command is performed. In addition, information about each file is printed as it is extracted.

Table 8-2: `restore` Commands

The main advantage of using `restore` interactively is that you can browse a backup volume to see what's on it and selectively restore particular files and directories.

7. Other Backup and Recovery Utilities

As indicated, the two utilities `dump` and `restore` are the recommended programs to use for backing up and restoring either entire file systems or individual files. The scheme of overlapping full and incremental backups allows you to maintain complete backup copies of all files with a minimum of time spent in the backup process. Because `dump` understands the notion of dump levels and maintains an on-line record of dumps in the file `/etc/dumpdates`, it can always figure out what to dump during any particular backup session. Also, in the event of an abnormal condition (such as a tape write error), `dump` knows how to interact with the operator to attempt to circumvent or solve the problem.

It is also possible to use other IBM/4.3 utilities to perform such backup and restore functions. For example, the utility `find` can be used to generate a list of all files that have been modified since a certain date. Those files can then be given as input to other utilities that will write them onto a backup medium such as floppy disk or streaming tape. This section briefly outlines how to use some of these other utilities to perform backup and restore functions.

7.1. Using Tar

The program `tar` can be used to save multiple files onto a single file and then restore them. For example, the command:

```
# tar cf /dev/rst0 /usr
```

will create a tape archive of the `/usr` file system on the device `/dev/rst0` (in this case, a streaming tape drive). Similarly, the command:

```
# tar xf /dev/rst0 /usr/include/sys/fs.h
```

will extract the file `/usr/include/sys/fs.h` from the named archive file and write it back into the file system. The file into which `tar` writes is usually a tape or floppy disk, but it can be any file at all.

The disks and tapes written by `tar` are extremely portable between UNIX-based systems. This makes `tar` a good program to use if you want to move data from one system to another, especially if the target system is not running the IBM/4.3 release of the UNIX system (so that `dump` and `restore` cannot be used). The main constraint with `tar` is that most versions of `tar` are not able to write onto multiple tapes or disks, so that the amount of data moved is limited to what will fit on a single tape or diskette.

As indicated, `tar` can be combined with the `find` command to provide a limited form of incremental backup facility. For example, to find all files that are at most four days old and save them on tape, give the command:

```
    # tar cf /dev/rst0 `find /usr -type f -mtime -4 -print;`
```

This command searches the `/usr` file system and produces a list of all files that have been created or modified within the last 4 days; then it archives each file listed onto the streaming tape drive named `/dev/rst0`.

To restore a file from a `tar` backup medium, you should first make sure that the desired file is found in the archive. Run the command:

```
    # tar tvf /dev/rst0
```

to obtain a listing of the files contained in the named archive file (here, `/dev/rst0`). Then, extract the desired file with the command:

```
    # tar xf /dev/rst0 /usr/include/sys/fs.h
```

The file will be extracted from the archive and placed at the correct spot in the file system. See the manual page `tar`(1) for a more complete specification of the options available with `tar` and for further examples of its usage.

7.2. Using Dd

If you have enough local disk drives, you may want to consider using the utility `dd` to make a copy of an active drive on a spare drive, thus providing a quick "full" backup (as illustrated in Figure D, above). For example, to copy the entire contents of the partition `/dev/hd0g` (which probably contains the `/usr` file system) to a spare partition `/dev/hd2g`, you would give the command:

```
    # dd if=/dev/rhd0g of=/dev/rhd2g
```

`dd` simply copies the input file (specified with the '`if=`' flag) to the output file (specified with the '`of=`' flag). Notice that `dd` operates on *raw* devices, so you must prefix the letter '`r`' to the file names, as illustrated in the example.

To restore a file system backed up in this manner, simply invert the input and output files:

```
    # dd if=/dev/rhd2g of=/dev/rhd0g
```

Notice that it is not possible to restore a single file, or a single directory, when using `dd` in this way, since it treats the entire partition as a single file.

A very common use of `dd` is to make a backup copy of the root file system, so that the system can be booted even if the original root file system somehow becomes corrupted. For example, if you are not using the `a` partition of the `hd1` drive, you can create a backup of the root file system by issuing the following command:

```
    # dd if=/dev/rhd0a of=/dev/rhd1a
```

You should note however that root file systems should be backed up in this way *only* if the two drives are of identical types and are partitioned in the same manner. Otherwise you risk overwriting important data on the `hd1` drive. You should also note that future system releases may include some sort of disk labels stored directly on the disk itself. If the portion of the disk that you move using `dd` includes the disk labels, you will effectively rename the destination disk. This may or may not adversely affect system operation. See the manual page `dd`(1) for a more complete specification of the options available with dd and for further examples of its usage.

8. Backup and Recovery Tips

- If for some reason you need to restore an entire file system from streaming tape (or other backup medium), make sure that you do a full (i.e., level 0) dump immediately thereafter. If you neglect this important step, later incremental dumps on that file system will fail.
- Remember that tape read and write errors happen occasionally and that they can prevent you from extracting certain files from a backup medium. In order to have maximum confidence in the integrity of a set of dump media, the following measures are recommended. First, find the name of the last file that was dumped:

  ```
  # restore tf /dev/rst0 | tail
  ```

  The last file listed in the output is the one you want. Next, instruct restore to restore that file:

  ```
  # restore x filename
  ```

  But mount the tapes in *increasing* order (in other words, insert the tapes in the same order that you originally inserted them when you made the dump). This strategy forces restore to read each tape completely. If there are problems with any of the backup media, you will discover them while performing this check.

This page intentionally left blank.

CHAPTER 9

Managing the Line Printer System

1. Introduction

The line printer system is a collection of programs and files that manage the spooling and processing of print jobs submitted by IBM/4.3 system users. Virtually any printer can be used in conjunction with the line printer system, ranging from very slow and unintelligent impact printers to very fast and intelligent laser printers and phototypesetters. The line printer system is able to manage multiple printers (which are usually of different types), multiple spooling queues, and both local and remote printers. It also provides facilities for maintaining accounting information about print jobs and for restricting printer access to a group of users or to a group of networked machines.

This chapter explains the configuration and operation of the line printer system. It illustrates the normal use of the system and provides troubleshooting advice to help you debug a malfunctioning system. This chapter also provides an introduction to constructing printer capability descriptions and printer interface programs, the two central items required to add a new type of printer to the line printer spooling system.

2. Overview of the Line Printer System

Typically, a user requests that a document be printed either by piping the output of a command to the `lpr` command or by invoking the `lpr` command with a filename as an argument. The `lpr` command then enters the request into a print queue. Under IBM/4.3, there is nothing mysterious about a print queue; it is simply a subdirectory of the main printer spooling directory (`/usr/spool/lpd`) that has the same name as one of the printers available to system users. For instance, if you have two printers available to users of your system, called `pr1` and `pr2`, then there will be two print queue directories, `/usr/spool/lpd/pr1` and `/usr/spool/lpd/pr2`. The overall operation of the line printing system is illustrated in the diagram on the following page.

You will notice that for each print request, `lpr` inserts *two* files into the spooling directory. For example, the diagram shows that a new print request for printer `pr1` actually consists of two files, here named `cfA002` and `dfA002`. The files with names beginning with `cf` are called *control files* and those with names beginning with `df` are called *data files*. The data file contains the data to be printed and is usually identical to the file that was originally given as input to `lpr`. The control file is created by `lpr` and contains instructions on how to process the data in the data file, as well as information relating to the owner of the print request, the host it originated on, the name of the corresponding data file, and so on.

The principal provider of line printing services on IBM/4.3 is the daemon `lpd`. When first started up (usually at boot time), `lpd` examines the file `/etc/printcap`, a data base file that contains information on the printers attached to the local machine and on printers accessible across a local area network. Lpd determines which printers have requests queued for them by inspecting each spool directory for data and control files. For each directory in which waiting requests are found, `lpd` forks a copy of itself to monitor the processing of those requests. It then listens for further requests for services, both from the local domain (by monitoring the socket `/dev/printer`) and from the network domain (by monitoring the socket corresponding to the entry `printer` in the file `/etc/services`).

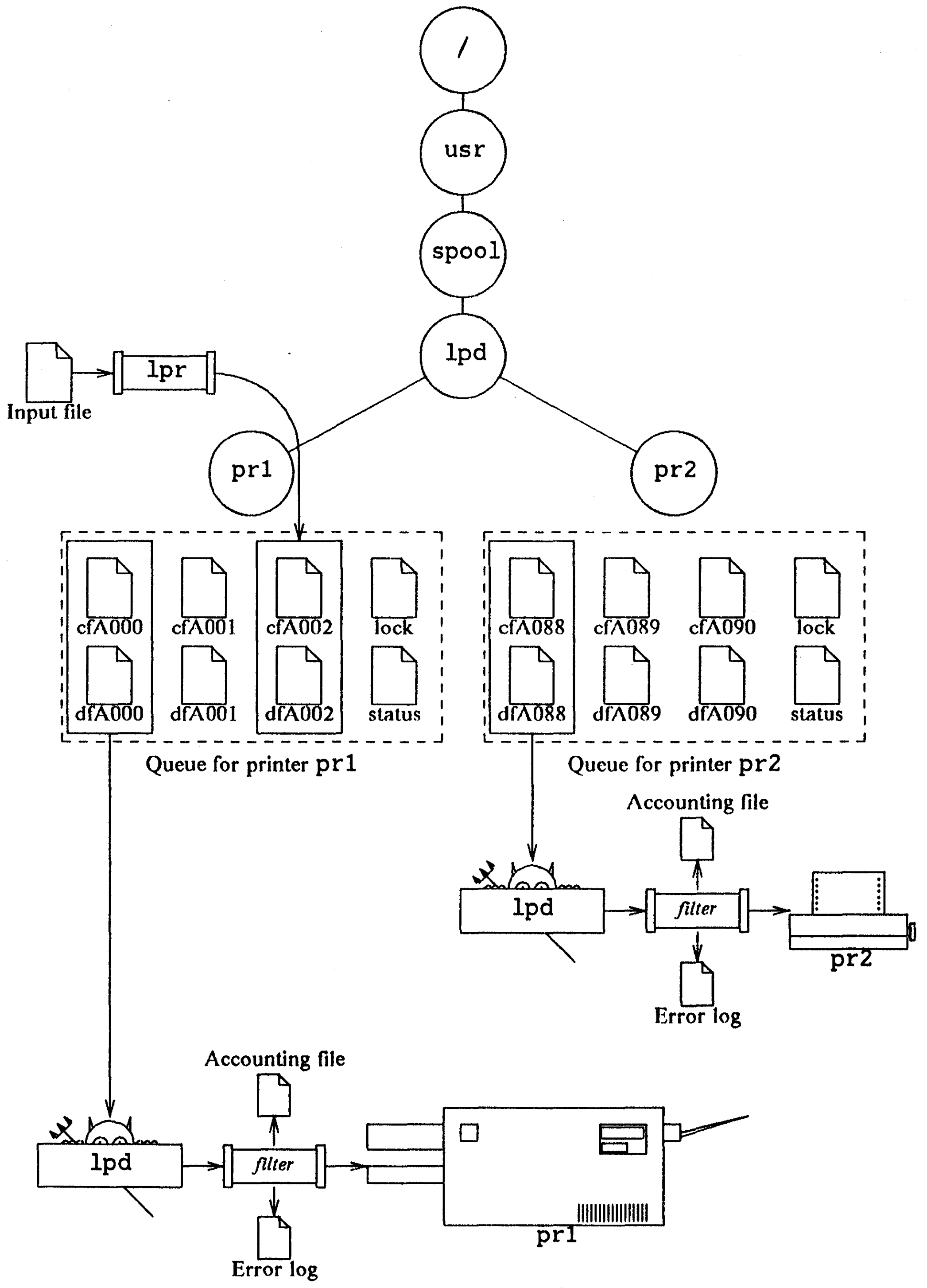


Figure 9-1: Overview of the Line Printer Spooling System

When `lpr` enters a print request into a spool directory, it sends a message to the corresponding `lpd` daemon informing it of the existence of the new request. If no daemon is actively servicing the indicated spool directory, the original `lpd` forks a copy of itself to process the request. The printing daemon first creates a lock file called `lock` in the spool directory to prevent multiple daemons from becoming active simultaneously and trying to process the same set of print requests. The lock file contains two lines; the first is the process identification number (pid) of the daemon and the second is the name of the control file of the job currently being processed. When the daemon finishes a print job, it removes the control and data files for the completed job from the spool directory and, if additional requests are pending, changes the second line of the lock file so that status inquiries by `lpq` and `lprm` will give accurate results. If no further requests are found, the daemon dies.

An `lpd` daemon servicing a particular printer processes jobs in a "first in, first out" pattern as dictated by the names of the control and data files. (As explained below, however, it is possible to rearrange the print queue by placing selected jobs at the top of the queue.) `Lpd` determines the type of the request then launches a printer interface program, or filter, to translate the information in the data file into a form appropriate for the intended printer. Printer filters are also responsible for maintaining accounting records and disposing of any error messages or other diagnostic output that may arise in the course of sending the file to the printer.

Once the line printer system is in operation, the administrator may query its status and control its operation using several utility programs. The program `lpq` gives information on queued requests, listing the files remaining to be printed, their owners, and their sizes. A more comprehensive administrative program is the utility `lpc`, which provides the ability to query printer status, enable or disable either a printer or its associated queue, restart a printer daemon, and rearrange the order of requests in a queue. It is important to note that spooling a print request and actually sending the spooled request to a printer are two separate operations that may be controlled independently of one another. It is `lpr` that places requests into a spool directory, while `lpd` arranges for their final processing. Thus, the line printer system administrator may stop `lpr`'s spooling to a particular printer without halting the printing of jobs already spooled. This is useful if you want to take a printer out of service but first want to let it complete any queued jobs. Similarly, the administrator may stop a device from printing without disabling spooling. This may be useful if you need to take the printer off-line temporarily to clear a paper jam, change ribbons or toner cartridges, or perform some other maintenance operation.

3. Installation

IBM/4.3 is distributed with the line printer programs and configuration files installed in their usual directories. The line printer system consists mainly of the following programs and files:

`/usr/ucb/lpq`	Spooling queue examination program
`/usr/ucb/lprm`	Queued job deletion program
`/usr/ucb/lpr`	Job queuing program
`/etc/printcap`	Printer capability data base
`/usr/lib/lpd`	Line printer daemon
`/etc/lpc`	Line printer control program
`/etc/hosts.lpd`	List of remote hosts allowed to use printer

Table 9-1: Main Components of the Printer Spooling System

It is recommended that you leave these programs and files in their preferred locations. If this is not possible, you should symbolically link the ones you install elsewhere to the locations listed above.

3.1. Disk Space Requirements

One of the important administrative tasks of the line printer system manager is to ensure that there is sufficient free space in the file system housing the spooling directories to hold all user-submitted print requests. If there is insufficient space, user jobs might not get spooled, or jobs that are spooled may be truncated at random points. It is to your distinct advantage to try to minimize the aggravation caused by recurring space problems by providing adequate space for the spooling system *before* it is enabled for use by system users.

It is impossible, however, to say categorically what a safe amount of space might be, since the amount of space needed to hold print jobs is a function of the number of users on your system, their volume of printing, the number and speed of the printers available, the service requirements of those printers, and several other factors. The best you can do is to estimate space requirements then monitor disk consumption for the first few days or weeks of operation to see whether your guess is a reasonable one. Anywhere from half a megabyte to several megabytes of space may be required to allow all user requests to be successfully spooled and printed. On a machine dedicated as a network print spooler for a large user community, as much as ten or twenty megabytes of space may be reasonable. For further discussion of space requirements and some advice on how to cope with limited resources, see the section "Space in the Spooling File System" below.

3.2. Access Permissions

The various print queues are generally maintained in the directory `/usr/spool/lpd`. The requests and status information for a particular printer are contained in a subdirectory whose name is the same as the printer. The spooling area should be writable only by the user `daemon` and by the group `daemon`. Further, the programs `lpr`, `lprm`, `lpd`, and `lpq` should be installed `setuid root` and `setgid daemon`. This scheme ensures that normal (non-privileged) users will be unable to remove print request other than their own (since they must use the utility program `lprm` to do so) or circumvent printer accounting (by inserting print requests into the queue directory without using `lpr`).

Once these programs are placed into the appropriate directories and given the correct permissions, three things still must be done to activate printing on a specific printer:

(1) Create a printer capability description for the printer (also called a "printcap" entry) and install it in the data base file `/etc/printcap`.

(2) Create a printer interface program (or a set of such programs) for your printer.

(3) Turn on the line printer daemon, `/usr/lib/lpd`, and enable the printer with `lpc`.

Most of the work involved in installing a new printer will occur in the first two operations. Some interface programs are quite trivial, while the interface programs for very sophisticated printers such as laser printers and digital typesetters are often quite complicated. For those printers, for example the IBM 3812 Pageprinter, you will probably receive a set of interface programs and printer capability descriptions as part of the software release accompanying the hardware. If not, your best course of action is to modify an existing printcap description and an existing printer interface program. The following two sections provide some advice on these issues.

3.3. Printer Capability Descriptions

The file `/etc/printcap` is the central storehouse of information about the printers available on your system. It is modeled after the `termcap` data base, which lists terminal capabilities, but it is much simpler than `termcap` and generally easier to maintain. This file must contain an entry for each printer that a user can select using a command like:

```
    % lpr -Pprinter ch.1
```

For a given printer, the corresponding `/etc/printcap` entry lists the spooling directory, accounting log files, lock files, and printer interface programs. If the printer is attached to the system through a serial port, the printcap entry will also list the baud rate to use when communicating with the printer and several other parameters necessary for the operating system to condition the line to the printer. A list of printer capabilities that may be specified in an `/etc/printcap` entry is included in the following table.

Name	Type	Default	Description
`af`	*str*	NULL	Accounting file
`br`	*num*	none	If `lp` is a serial line, set the baud rate
`cf`	*str*	NULL	`cifplot` data filter
`df`	*str*	NULL	`tex` data filter (DVI format)
`fc`	*num*	0	If `lp` is a serial line, clear flag bits
`ff`	*str*	`\f`	String to send for a form feed
`fo`	*bool*	false	Print a form feed when device is opened
`fs`	*num*	0	Like `fc`, but set bits
`gf`	*str*	NULL	Graph data filter (`plot`(3X) format)
`hl`	*bool*	false	Print the burst header page last
`ic`	*bool*	false	Driver supports (non standard) ioctl to indent printout
`if`	*str*	NULL	Text filter which does accounting
`lf`	*str*	`/dev/console`	Error logging file
`lo`	*str*	`lock`	Lock file
`lp`	*str*	`/dev/lp`	Device to open for output
`mx`	*num*	1000	Maximum file size (in `BUFSIZ` blocks) 0 = unlimited
`nd`	*str*	NULL	Next directory for list of queues (unimplemented)
`nf`	*str*	NULL	`ditroff` data filter (device independent `troff`)
`of`	*str*	NULL	Output filtering program
`pl`	*num*	66	Page length (in lines)
`pw`	*num*	132	Page width (in characters)
`px`	*num*	0	Page width in pixels (horizontal)
`py`	*num*	0	Page length in pixels (vertical)
`rf`	*str*	NULL	Filter for printing FORTRAN style text files
`rg`	*str*	NULL	Restricted group
`rm`	*str*	NULL	Machine for remote printer
`rp`	*str*	`lp`	Remote printer name
`rs`	*bool*	false	Restrict remote users to those with local accounts
`rw`	*bool*	false	Open the printer device for reading and writing
`sb`	*bool*	false	Short banner (one line only)
`sc`	*bool*	false	Suppress multiple copies
`sd`	*str*	`/usr/spool/lpd`	Spool directory
`sf`	*bool*	false	Suppress form feeds
`sh`	*bool*	false	Suppress printing of burst page header
`st`	*str*	`status`	Status file
`tc`	*str*	NULL	Use description of printer *str*
`tf`	*str*	NULL	`troff` data filter (CAT phototypesetter)
`tr`	*str*	NULL	Trailer string to print when queue empties
`vf`	*str*	NULL	Raster image filter
`xc`	*num*	0	If `lp` is a serial line, clear local mode bits
`xs`	*num*	0	Like `xc`, but set bits

Table 9-2: Printer Capabilities

One important difference between /etc/printcap and /etc/termcap is that a user cannot substitute some other printer capability entry for the ones contained in /etc/printcap. (Recall that /etc/termcap will not be searched if the user has the environment variable TERMCAP set.) The reason is simply that otherwise a user could circumvent the accounting and logging procedures specified in the printcap entry.

Like terminal capability descriptions, a particular printer capability description may not include all of the items listed in the preceding table. A very simple printcap entry might look like this:

```
lp|default line printer:\
        :af=/usr/adm/lp.acct:if=/usr/lib/lpf:
```

Any entries not specified in the printer capability description will be assigned the default value. If, as illustrated, the name of the printer is 'lp', the line printer system will treat it as the default line printer and send it all printing requests that do not include a printer specification -P*printer*. You can override this specification using the PRINTER environment variable.

3.4. Special Entries for the 3812 Pageprinter

In addition to the standard entries listed above, the IBM/4.3 system recognizes two additional printer capabilities, SS and PP, which are intended for use by the IBM 3812 Pageprinter. The PP entry specifies the port through which the Pageprinter is attached to the host machine. This port usually differs from that specified in the lp entry since the IBM/4.3 system uses a special line protocol, ap, to communicate with the 3812. If the Pageprinter is to be used on your system, the kernel configuration file must specify the asynchronous protocol ap pseudo-device. Refer to Chapter 3, "Reconfiguring the Operating System", for complete instructions on how to include that specification.

The SS printer capability identifies the log file for the 3812. This file maintains a list of all messages received from the printed itself, such as PAPER JAM or other diagnostic messages relating to the state of the printer. This file differs from the error message file specified by the lf entry, which collects messages from the interface filter. In any event, the use of lf is not recommended; instead, syslogd should be used to manage messages from the interface filters, as explained below.

3.5. Printers on Serial Lines

There are several printer capabilities that allow you to specify how the system is to communicate with a printer attached to your system through a serial port. For example, the lp entry lists the device to open for output. It can often be omitted if only one printer is available on your system, since in that case you will very likely have linked the special file for that port to the device /dev/lp (which is the default port to open). Further, the br entry indicates the baud rate at which the system is to communicate with the printer. Which rate you run at depends on the capabilities of the printer. Some impact printers print so slowly that there is no point in using any baud rate higher than 300. Most modern non-impact printers (such as laser printers and typesetters), on the other hand, are able to communicate with a host computer at 9600 baud with no problem, as long as flow control is enabled (so that the host computer will quit sending data when the printer's buffers are full).

There are four other printcap capabilities related to serial operation, fs, fc, xs, and xc. The first two are used to set or clear flag bits on the serial line driver that determine how the system interprets output to the printer and input from the printer. The latter two capabilities are used to set or clear local mode words.

4. Interface Programs

When `lpd` undertakes to service a print request found in the spooling directory, it passes the data to be printed to an *interface program* or *filter*. The function of the interface program is to handle whatever device-specific processing still remains to be done on the input file before it can be sent to the printer. The input file to the filter is the data to be printed (namely, the '`df`' file) and the output file is simply the printer itself, as depicted in the following general diagram:

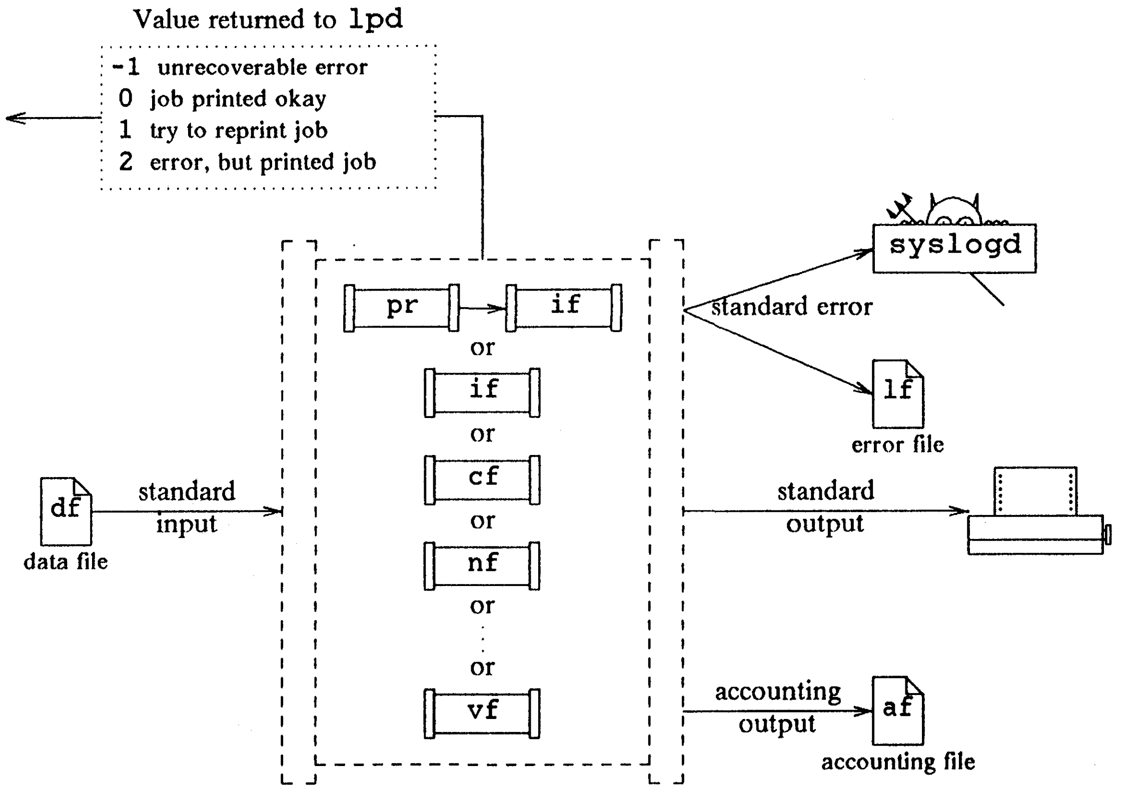


Figure 9-2: Position of the Printer Interface Program

The work that the interface filter needs to perform depends on the kind of input it is given, and there are several possibilities. For example, the data file may contain the output of any one of a number of common text-processing programs such as `nroff`, `troff`, or `tex`, or it may contain output of some other IBM/4.3 utility such as `plot` or `cifplot`. The data file may even consist of nothing more than the output of piping a normal ASCII text file through `pr`.

Because the data file may contain various types of input, a number of interface programs must be available to turn that data file into a form understandable by the destination printer. A moderately capable printer (i.e., one able to produce output for a number of different types of input data file) will have a description like this:

```
dt:sample line printer:\
     :lp=/dev/lp:af=/usr/adm/lp.acct:\
     :if=/usr/local/dt/dtif:of=/usr/local/dt/dtof:\
     :gf=/usr/local/dt/dtgf:nf=/usr/local/dt/dtnf:\
     :df=/usr/local/dt/dtdf:vf=/usr/local/dt/dtvf:\
     :tf=/usr/local/dt/dttf:
```

The `nf` printcap entry specifies the printer filter to be used if the data file contains output from `ditroff`, the device-independent `troff`. Similarly, the `tf` entry specifies the filter to use if the data file contains output from the original `troff` (intended for a CAT phototypesetter). Depending on the kind of input contained in the data file, `lpd` will spawn the appropriate filter. `lpd` determines which filter is appropriate by inspecting the control file. (Of course, the control file garners this information ultimately from `lpr`.)

Table 9-2 includes the filters available with IBM/4.3.

4.1. Filter Return Values

In addition to filtering the data file into a form suitable for sending to the printer, the interface program must also communicate the results of its actions back to the calling daemon so that `lpd` knows whether to proceed with further printing or to attempt some corrective action. The interface program, whether it is a compiled program or a shell script, must pass an exit value back to its parent process in accordance with the conventions illustrated in the diagram above. If the filtering and printing occurred as expected and no errors were discovered, a value of `0` must be returned. In that case, `lpd` knows that further jobs may be processed and sent to the printer. If for some reason the job should be reprinted, a value of `1` should be returned. If an error occurred but the job was printed anyway, a value of `2` should be returned. Finally, if some unrecoverable error happened, a value of `-1` should be returned.

4.2. Signals

The printer interface program may trap and interpret signals in any way desirable, or it may ignore them entirely, with one exception. The interrupt signal, `SIGINT`, must be trapped and used to perform whatever cleanup operations are necessary to keep the printer system running smoothly and to keep from cluttering the file system. The parent `lpd` daemon may send this signal to the interface program for a variety of reasons, the most obvious one being that a user requests termination of a currently printing job with the `lprm` command.

When the printer interface program receives the `SIGINT` signal, it must do whatever it can to clean up after itself and leave the printer in a state ready to receive further print jobs. If any lock files had been created to ensure exclusive use of some resources, they should be removed. Similarly, any temporary files created in the filtering process should be removed.

4.3. Arguments

The arguments passed to a filter vary, depending on the specific filter. The following arguments apply to most filters:

filter `-x`*width* `-y`*length* `-n` *login* `-h` *host acctg file*

The `-x` and `-y` arguments specify horizontal and vertical space, respectively, in pixels. (See the `px` and `py` entries in the `printcap` file.) The `-n` option specifies the login name of the job owner, and the `-h` option specifies the host name of the owner. The `acctg_file` argument specifies the name of the accounting file (from `printcap`).

There are two filters whose arguments differ:

- The `if` filter has the following arguments:

  filter [`-c`] `-w`*width* `-l`*length* `-i`*indent* `-n` *login* `-h` *host acctg file*

  You can use the `-c` flag, which is optional, to pass control characters which you do not want interpreted to the printer. (for example, when using the `-l` option of `lpr` to print a file). The values for `-w` and `-l` come from the `pw` and `pl` entries in the `printcap` file. The remaining arguments are the same as described above.

- The `of` filter is called with the following arguments:

  filter `-w`*width* `-l`*length*

The values for `-w` and `-l` come from the `pw` and `pl` entries in the `printcap` file.

5. Operation

This section discusses three aspects of operating the line printer system: starting it up, removing jobs from the printer queue, and using a remote printer.

5.1. Starting Up the Line Printer System

For the line printer system to operate normally, the line printer daemon `lpd` must be running. Usually the daemon is started up automatically at boot time by placing the following lines into the multi-user initialization file, `/etc/rc`:

```
if [ -f /usr/lib/lpd ]; then
        rm -f /dev/printer
        /usr/lib/lpd
        echo 'Printer daemon started' > /dev/console
fi
```

You should place these lines into that file if they are not already there. If you want to launch the line printer daemon without rebooting your system, simply give its full name as a command:

```
# /usr/lib/lpd
```

As indicated in the overview above, this will launch the parent `lpq` daemon, who will scan the queue directories for waiting print jobs and listen on various sockets for new print requests.

5.2. Removing Jobs from a Printer Queue

The `lprm` command is provided so that ordinary users can remove jobs from a spooling queue that they have previously submitted. This is accomplished by removing both the control and data files corresponding to the specified print requests. If necessary, `lprm` will first kill off a daemon servicing a queue and then restart it after the files are removed. If the files are destined for a printer on a remote system, `lprm` will first look into the local queue to see if the requested job still resides there. If not, it tries to remove the control and data files from the remote machine.

Normally, only those jobs owned by a particular user are candidates for removal. The diagnostic message:

```
Permission denied
```

will be issued if a user attempts to remove print requests that are not owned by that user. The superuser may remove any jobs from the print queue, however. A particular printer's queue may be entirely emptied by the command:

`# lprm -P`*printer* `-`

If the *printer* option is not specified, or the environment variable `PRINTER` is not defined, only the print requests for the default line printer will be removed.

5.3. Using a Remote Printer

It is extremely simple to configure the `printcap` file so that some or all print requests are routed over a local area network for printing on a printer attached to a remote host. Suppose, for example, that you want to configure the line printer daemon on the local host `tuna` to send all print jobs to the remote host `grunnion`. To do this, simply install a printcap entry like the following into the file `/etc/printcap` on `tuna`:

```
lp|default line printer:\
        :lp=:rm=grunnion:rp=pr3:sd=/usr/spool/lpd:mx#0:
```

By giving the printer the name '`lp`', you have established it as the default printer. This part of the printcap is optional, and you may give the remote printer whatever designation you like. The important part of this printcap is that the `lp` entry is empty and the `rm` entry contains the host name of the remote machine. Further, the `rp` entry lists the name of the printer on the remote system (in this example, `pr3`). If you want the print requests to be routed to the default printer on the remote system, you can omit the `rp` entry. Of course, the remote host specified by the `rm` entry must be a known name on your local area network (as stored in the file `/etc/hosts`). In addition, if there is an `/etc/hosts.lpd` file on `grunnion`, it must contain the name `tuna` so that users on `tuna` will be allowed access to the remote printer. If the printer capability description for `pr3` on `grunnion` contains the entry `rs`, only those users on `tuna` who also have accounts on `grunnion` will be allowed to print on `pr3`.

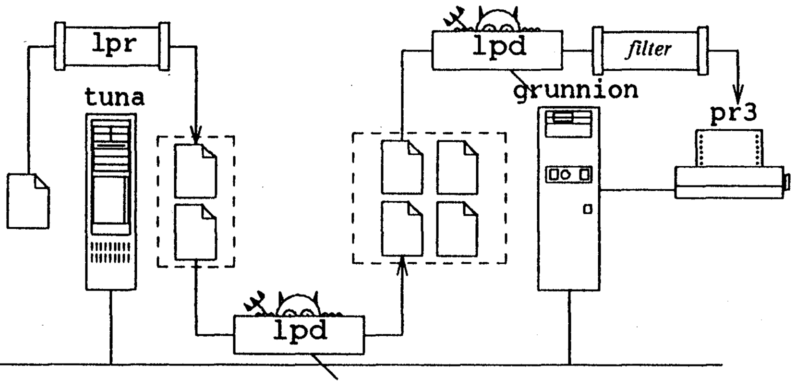


Figure 9-3: Spooling Print Requests to a Remote Printer

As you can see above, the file to be printed is stored in two different print queues, one on the local host and one on the remote host. This allows spooling for the remote printer to continue on the local host even if the remote host is down or network traffic is temporarily interrupted. Once the data and control files are transferred to the remote print queue, they are removed from the local queue. Similarly, no files will be transferred from the spooling directory on `tuna` to the spooling directory on `grunnion` unless enough disk space is available on `grunnion` to contain the files. The local printcap for the remote printer contains the entry `mx#0` so that the local `lpd` daemon will not enforce any size restrictions on data files being queued.

Note also that all data file filtering is done on the remote machine. This means that the data file will be filtered by the appropriate filter as specified in the `/etc/printcap` file on the remote machine. As a result, no filters are listed in the local printcap file.

6. Administering with Lpc

Once the line printer system is installed, configured, and launched, you may control its operation by using the `lpc` program. For example, you can prevent additional print jobs from being placed by `lpr` into the print queue of the default printer by executing the command:

```
# lpc disable lp
```

You may manipulate the printer and the print queue independently of each other. For example, you may deactivate a printer and its queue altogether (by disabling both of them). You may disable just the printer, for instance to clear a paper jam, while allowing the print queue still to accept print requests. Finally, you may disable just the print queue, so that existing jobs will be serviced even though no new jobs are accepted.

The following table lists the commands you can specify as arguments to the `lpc` command.

Command	Arguments	Description
`help`	*command*	Print a short description of the specified *command*. If no argument is given, print a list of all commands recognized by `lpc`. '`?`' is a synonym for '`help`'.
`abort`	*printer*	If there is an active spooling daemon (`lpd`) on the local host, kill it. Furthermore, prevent new invocations of the `lpd` daemon from being started by `lpr`.
`clean`	*printer*	Remove any temporary files, data files, or control files that cannot be printed from the queue of the specified local printer.
`disable`	*printer*	Turn the specified printer queue off. This command prevents new jobs from being entered into the queue by the `lpr` command.
`down`	*printer message*	Turn the specified printer queue off, disable printing, and put the specified message into the printer status file. This command is normally used to deactivate a printer and to inform users of the cause of the deactivation.
`enable`	*printer*	Enable spooling into the specified printer's queue, thereby allowing `lpr` to enter new jobs.
`exit`		Exit from the `lpc` command. This is useful only if you are running `lpc` interactively (see below). '`quit`' is a synonym for '`exit`'.
`restart`	*printer*	Attempt to launch a new line printer daemon, `lpd`. This is useful if the previous `lpd` has died unexpectedly, leaving unprocessed jobs in a queue. If you are the superuser, you may execute this command even if `lpd` is currently running; in that case, however, the current daemon will first be killed. This command is most useful for normal unprivileged users to attempt to restart the daemon overseeing the queue for printer *printer*.
`start`	*printer*	Enable printing and start a spooling daemon for the specified printer.
`status`	*printer*	Display the status of the specified printer, listing whether queuing and/or printing are enabled, the number of jobs in the queue, and whether a daemon process is active.
`stop`	*printer*	Stop the spooling daemon after the current job completes and then disable printing. Users may continue to spool requests using `lpr`.
`topq`	*printer jobnum*	Place the specified job numbers at the top of the printer queue.
`up`	*printer*	Enable everything and start a new printer daemon on the local host. This command undoes the effects of the `down` command.

Table 9-3: `lpc` Commands

Note that you may give the argument '`all`' anyplace that a particular printer is specified if you want the command to affect all printers listed in `/etc/printcap`.

6.1. Using Lpc Interactively

If you invoke the command `lpc` with no arguments, you will be prompted for commands interactively. For example:

```
# /etc/lpc
lpc> help
Commands may be abbreviated.  Commands are:

abort   enable  disable help    restart status  topq    ?
clean   exit    down    quit    start   stop    up
lpc>
```

It is generally more useful to give administrative commands from this interactive mode than from the command line since you will be able to execute a `status` immediately to see whether the requested change has actually occurred. To return to the shell, simply give the `exit` or `quit` command.

6.2. Restricting Printer Access

The default operation of the line printer spooling system is to accept requests for all printers specified in the `/etc/printcap` file from all users on the system. It is however possible to restrict printing on a particular printer to the members of a given group. This is accomplished by specifying the `rg` printer capability. For example, if the printcap description for a particular printer contains the entry:

```
:rg=doc:
```

then only users belonging to the group `doc` will be allowed to queue requests to that printer. Note that the specified printer may be either local or remote; the `rg` capability governs only the placement of print requests into the local print queues.

As indicated above, it is also possible to control remote access to local printers. A remote host must have its name listed in the file `/etc/hosts.lpd` for the local machine to allow the remote `lpd` daemon to send requests to the local printer. Remote users may also submit jobs directly on the local machine using `rlogin` or `rsh` commands if the name of their remote host is listed in the file `/etc/hosts.equiv`. It is possible, however, to restrict printing by remote users to those that actually have accounts on the local machine by including the boolean `rs` in the printcap entry.

7. The Format of Printer Control Files

As mentioned above, the `lpd` daemon servicing a particular print queue determines which output filter is appropriate for a given job by inspecting the associated control file. The control file contains information about the origin of the print job and the processing necessary to turn the data file into a form suitable for printing. It may also specify that certain non-printing actions are to occur, such as sending mail to the user who submitted the job or removing certain files upon completion of the actual printing.

A control file is a sequence of lines, each of which begins with a key letter that indicates what to do with the data on the remainder of the line. The data specify in part who submitted the print request, what machine it originated on, and what data file constitutes the print job. The command-line arguments passed to the interface filter are collected by `lpd` entirely from the entries in the control file. The output filter typically uses some of the information included in the control file to print a banner on the burst page. A sample control file might look like this:

```
    Htuna
    Pnat
    Jstdin
    Ctuna
    Lnat
    fdfA001tuna
    UdfA001tuna
    N
```

According to this sample control file, the job originated on the machine `tuna` (key letter H) and was submitted by the user `nat` (key letter P). The file to be printed was received via the standard input to `lpr` (key letter J). Since there is a line beginning with the key letter '`f`', the data file `dfA001tuna` is already formatted and should therefore be further processed with the `if` interface filter (or the `of` filter if accounting is not required). This control file also specifies that the data file should be removed (unlinked) after successful completion of the printing (key letter U). Finally, since there is an N key letter that has no data following it, the print request must have been received by `lpr` through its standard input. In other words, `lpr` was invoked in a pipeline.

Generally, there is no need to look at control files in any detail. They are automatically generated by `lpr` and removed by `lpd` upon completion of printing. If you do need to investigate the control files, you should consult the manual pages for `lpd`(8) for a complete list of the available key letters and their meanings.

8. Troubleshooting

This section discusses two aspects of handling problems with a printer system: establishing error message logging, and solving problems relating to space in the spooling file system.

8.1. Error Message Logging

Errors that arise in the operation of `lpd` and associated commands such as `lpr` and `lpc` are logged using the system-wide error-logging daemon `syslogd`. Exactly where the error message is placed depends on the severity of the error and the configuration of the message-handling daemon. Many sites prefer to log printer messages to a standard message file such as `/usr/adm/messages`. To do this, simply insert a line like the following into the `syslogd` configuration file, `/etc/syslog.conf`:

```
    *.err              /usr/adm/messages
```

You can instead route all line printer spooling system messages to a specific location by inserting a line like the following into `/etc/syslog.conf`:

```
    lpr.*              /usr/adm/lpd.errors
```

This configuration will cause all error messages from the line printer system to be logged in the file `/usr/adm/lpd.errors`, regardless of the severity of the problem. In this case, you will have a specific location in which to look for such error messages.

Recall that the `lf` printer capability specifies the name of an error log file. This is the location to which error messages are sent *when they originate from the line printer interface program.* When a message is sent to the standard error output of the interface program, it will be appended to the file specified by the `lf` entry (and if none is specified, it will be sent to `/dev/console`). The `lf` capability is a relic of pre-`syslogd` days and is not recommended as a general purpose message logging scheme. Instead, it is highly recommended that the interface program use the `syslogd` daemon to report errors encountered when filtering or printing a request. This can be accomplished either by invoking the `syslog()` function (if the interface filter is a compiled program) or by executing the `logger` command (if the interface filter is a shell script). For a more complete discussion of the message-logging daemon `syslogd` and the configuration of `/etc/syslog.conf`, see Chapter 15, "System Messages." For information on the messages that can be generated by the line printer system, see the "4.3BSD Line Printer Spooler Manual" (SMM:6).

8.2. Space in the Spooling File System

Occasionally, a data file will be too large to fit into the space available in the spooling directory of the specified printer. For instance, the intermediate output file of a `troff` command (or other document processing system) can easily grow to several megabytes for a document that is even slightly over a hundred pages long. When there is insufficient space to copy the entire data file into the spooling area, the `lpr` command will spool the request, but it will truncate the data file at some appropriate spot so that the truncated part will fit into the spooling area. Filtering and printing of the truncated data file will then proceed as usual.

Generally, however, this is not what your users want, so you may need to institute one of several possible remedies for spooling space problems. Of course, the most obvious solution (and also the best long-term solution) is to increase the amount of available spooling space by moving the file system which houses the spooling directory to a larger disk partition. In cases where a particular machine is operating as a print server for an entire network of machines, you will want to place the `/usr` file system in a relatively large partition (for example, the `c` partition of an unused disk). You might even want to install `/usr/spool` as a separate file system entirely. Then you can selectively decide not to back it up at all when you dump other file systems to secondary media.

If it is not possible to solve spooling space problems by increasing the amount of available space, you can instruct users to submit a large print request without actually copying the data file to the spooling directory. To do this, they can use the `-s` option to `lpr`, which requests that a symbolic link be made to the file. For example, the command:

```
% lpr -s -n book.tr
```

will symbolically link the file to be printed, `book.tr`, to the data file in the default spool directory, `/usr/spool/lpd/lp`. Since a symbolic link is used, you can link a data file to its image in the spooling directory even across file system boundaries.

If you observe that `lpr`-submitted requests are often causing the spooling file system to fill up, you may want to instruct users submitting large jobs to use symbolic links into the spool directory in this way rather than always creating a copy of the print job there. Often such a large print job is the output of a complicated pipeline of pre- and post-processors that is launched by `make`, so it is relatively painless to have the job always be symbolically linked by editing the `Makefile` to include the `-s` flag on `lpr`.

There is an easy way for the line printer system administrator to get `lpr` to enforce a limit on how much free space it will consume with its spooling activity. The file `minfree` in each spool directory, if it exists, specifies the minimum amount of space to leave free on the file system housing the spool directories. This feature is provided so that the line printer queue will not completely fill the disk. The file is in human-readable ASCII format, so it can be edited with your favorite text editor if you need to change the value.

If you are routing requests over a network, you must also make certain that enough space exists in the remote spooling areas to accommodate requests generated on a local machine. Recall that control and data files are forwarded to a remote spooling area only if there is sufficient space to hold them both. If there is not, the request will not be accepted on the remote machine (or else it may silently be truncated to fit). Unlike the case of large files on the local machine, it is not possible to symbolically link a large data file across a network to a remote spooling area. Once again, the best solution to a space shortage problem on a network print spooler is to enlarge the amount of space available for line printer spooling on the remote system. A short-term remedy is to remotely copy the large file to temporary space on the remote machine and then invoke `lpr` on the remote machine, specifying that a symbolic link be used. For example:

```
% rcp book.tr grunnion:/tmp/book.tr
% rsh grunnion ´lpr -s -r -n /tmp/book.tr´
```

The `-r` option instructs `lpr` on the remote system to remove the data file upon completion of the printing.

Finally, if a print job is too large to be copied into any file system on the remote print spooler, the best solution is to segment the job into smaller jobs. Exactly how you do this depends on the type of print job. For example, the following simple script will select a range of pages from a `ditroff` output file:

```
#! /bin/sh
# ditpgs: extract a range of pages from ditroff output
if [ $# != 2 -a $# != 3 ]; then
     echo "ditpgs: incorrect number of arguments!"
     exit 1
fi

BEG=$1
END=`expr $2 + 1`

sed -n -e "1,/^x init/p" \
       -e "/^x font/p"   \
       -e "/^p$BEG$/,/^p$END$/p" $3

echo "x stop"
```

Similar scripts can be devised for other types of print jobs. Since these page-selection filters are useful in their own right (aside from their value in solving space problems), it is worth taking a few minutes to develop them for the types of output common at your site.

9. Printer Accounting

The line printer system can easily be configured to maintain accounting information about all print jobs sent to it, and there is a utility, `pac`, that helps determine printer usage and monetary charges for each user account. Even in computing environments where there is no need to charges users for printer use, it may still be advisable to enable printer accounting so that the system administrator will be able to monitor printer activity, thereby providing useful information about consumption of paper, ribbons, or other necessary printer supplies.

To enable printer accounting, the printcap must include an `af` entry, which specifies the file into which the output filter should place per-job printer usage records. Typically this file is stored in the `/usr/adm` directory, like most other log files and accounting records. For example:

```
:af=/usr/adm/lp.acct:
```

In addition to the `af` entry, the printer capability description must include indications of printer interface programs other than that specified by the `of` entry. The reason for this is that the `of` filter is started by `lpd` only once and has all text files piped through it. Thus it is impossible to maintain records about individual print jobs. The other print filters are launched on a per-job basis and automatically do accounting if an `af` entry is included in the printcap. If you write your own printer interface program, you must make certain that it correctly appends records to the accounting file (whose name is passed to it as an argument).

Each record of a successful user print job occupies one line in the accounting file. The general format of the line is as follows:

number-of-pages *hostname*:*username*

(The white space here is the tab character.) If you are writing your own printer filter, you will need to include sections of code similar to the following.

```
char *name;
char *host;
char *acctfile;

if ( freopen(acctfile, "a", stdout) != NULL )
  {
        printf("%7.2f\t%s:%s\n",
                    (float)npages, host, name);
  }
```

To interpret the printer accounting records, you may use the utility program `pac`. See the manual page `pac`(8) for details on using that program.

10. Printer Management Tips

- When you attach a printer to a particular serial port, you must also make certain that the operating system does not try to accept logins on that port. If you do not suitably configure the system, the kernel will launch a `getty` process on that port and data originating from the printer may be interpreted by `getty` as a login attempt. To make sure that this doesn't happen, you need edit the file `/etc/ttys` and change the status indicator '`on`' to '`off`'. For example, if a printer is attached to your machine on serial port `/dev/tty00`, the corresponding line in `/etc/ttys` might look like this:

  ```
  tty00   "/etc/getty std.9600"   ibm3812   off secure
  ```

- Sometimes the `lpq` command will report that a daemon is active on a particular printer but in fact nothing is printing on it. In this case, the printer is said to be "hung". To remedy the situation, try executing the following two commands:

  ```
  # lpc abort printer
  # lpc start printer
  ```

- It is possible to print multiple copies of a single document by including the **-#***n* command line option on `lpr`. You should be aware however that this option causes `lpd` to send the same data file to the printer in series the specified number of times. Some printers are able to print multiple copies of a single page, and hence multiple copies of a document, much more efficiently than if the request is sent as separate jobs. You may therefore wish to modify the printer interface filters to intercept requests for multiple copies and translate them into the appropriate printer control statements.

  For example, the following script contains a fragment of PostScript[1] code that will instruct the printer to produce the requested number of copies:

  ```
  #! /bin/sh
  # pscopy: print multiple copies of a PostScript file
  echo "%!"
  ```

[1]PostScript is a registered trademark of Adobe Systems Incorporated.

```
echo "statusdict begin /#copies $1 def end"
cat -
# now reset the printer
echo "%!"
echo "statusdict begin /#copies 1 def end"
```

Note that the printer output will not be collated, so that manual reshuffling of the pages may be necessary. Even including the time it takes to collate the output by hand, however, the time savings can be dramatic for large and moderately-sized documents.

- To ensure that certain line printer system commands work properly across a local area network, you should place into the `/.rhosts` file on the remote print server the names of all machines from which that server will accept print jobs. For example, the utility `lprm` may not succeed in removing a job that is already queued on a remote machine unless the host on which the command is run is a "trusted" host.

CHAPTER 10

Understanding the Uucp Network

1. Introduction

The family of programs known collectively as "uucp" allows you to establish a network of machines that can exchange files, run commands remotely, forward mail, and transmit news articles. Some typical uucp activity is illustrated in the following diagram:

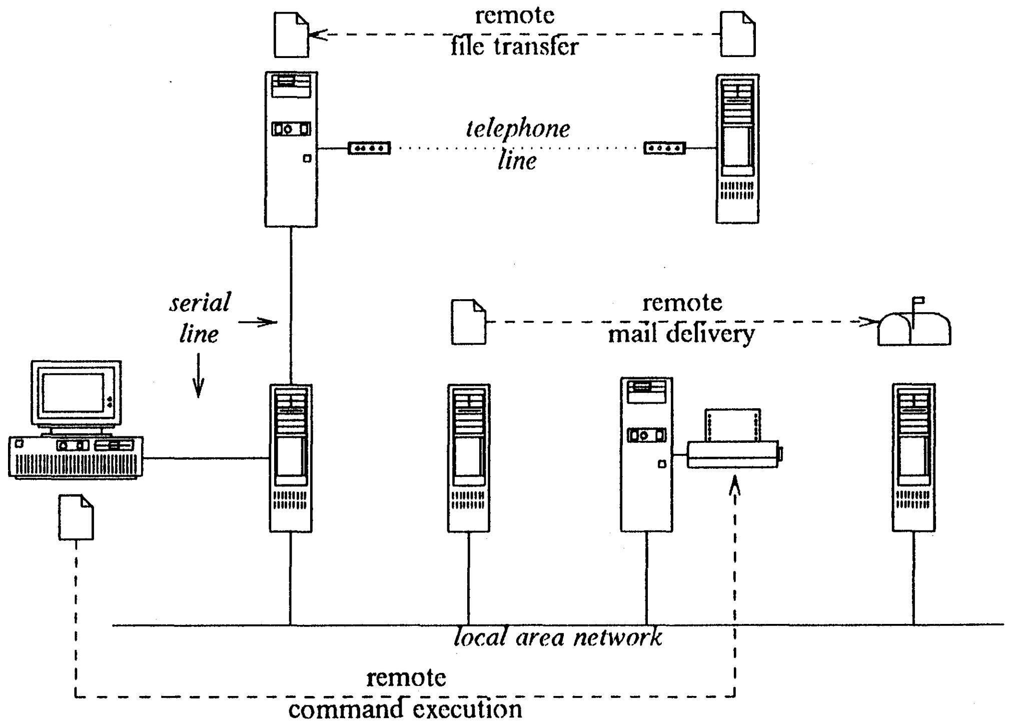


Figure 10-1: Some Typical uucp Network Activities

The uucp network is fundamentally a *batch* network, meaning that user requests to transfer files between systems or to execute commands remotely are translated into work and data files that are then queued to a spooling directory where they will found and interpreted by the `uucico` program. Although `uucico` is invoked immediately whenever the user-level programs `uucp` and `uux` are executed, the requests may not actually be processed at that time. For example, there may not currently be a free modem available to handle the outgoing telephone call, or the uucp system may have been configured to disallow calls to the remote system until the late hours of the night. Exactly when the queued requests are processed therefore depends on parameters specific to the remote system, over which a normal user may have no control. The processing of requests also depends upon how often the `uucico` program is run, and that is usually controlled, as you will see, by the `cron` daemon and its configuration file, `/usr/lib/crontab`.

uucp was designed as this kind of "store and forward" network largely because it was first intended to service file transfer over low-speed, non-dedicated telephone lines, where there is usually no permanent communications link between any two systems on the network. It made sense for each system to save up requests for the other system and then to try to establish a communications channel at some specified time. If no connection could be made, the requests simply would be saved until one could be established. Recently, uucp has been given the ability to carry out its activities over a local area network using the Internet protocols (TCP/IP).

There are two principal services provided by the uucp networking software, file transfer from one machine to another and remote command execution. File transfer is accomplished with the `uucp` command. For example, if a user `nat` wants to copy a file named '`ch.1`' in the current working directory on the local host to the home directory on remote host named '`grunnion`', that user might run the following command:

```
% uucp -C ch.1 grunnion!~nat/ch.1
```

The sequence of steps involved in servicing this request is depicted in the following diagram:

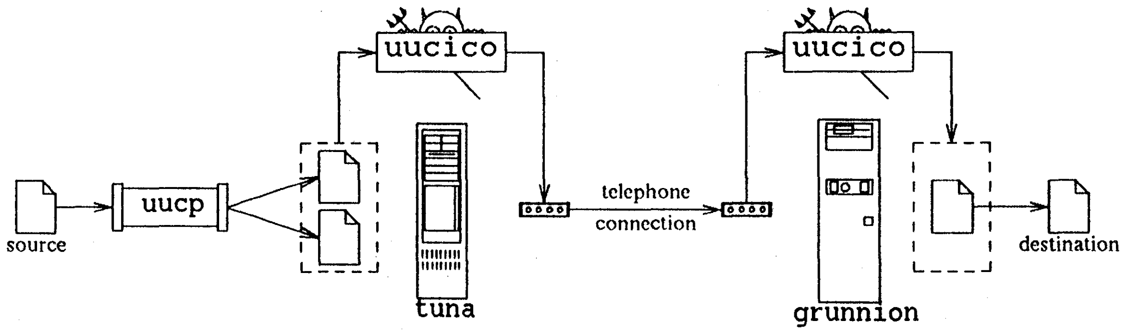


Figure 10-2: Remote Copy With `uucp`

When the command listed above is executed, `uucp` will place several files into a spool directory. One of the files, a data file, is simply the file to be transferred. It is copied into the spool directory since the `-C` command line option was specified. (The default action is not to spool a copy, but to use the specified source file. In this case a copy was deliberately spooled so that future changes to the original source file will not appear in the remote copy.) The other file placed into the spool directory is a work (or control) file containing instructions governing the file transfer. The work and data files remain spooled in this manner until a `uucico` daemon finds them there, at which time the daemon will interpret the work file and initiate a telephone connection to the appropriate remote system. When `uucico` succeeds in logging in to the remote system, the shell launched is in fact another invocation of the `uucico` daemon. The second daemon is said to be in *slave* mode, receiving instructions and data from the original `uucico` daemon on `tuna`, which is in *master* mode. If everything goes okay, the file will be transferred from the local host to the remote, placed into a temporary spooling directory on the remote host, and finally moved into the user's home directory on the remote host. The file transfer requested by the original `uucp` command is now complete.

Remote command execution is accomplished with the `uux` command. A user on the local system, `tuna`, can request that a command be executed on a remote machine by invoking the `uux` command, for instance as follows:

```
% cat ch.1 | uux - grunnion!lpr
```

This requests that the file `ch.1` in the current directory be piped through `uux`, which will send its standard input over to `grunnion` to be piped through the command `lpr`. The sequence of operations set in motion by this pipeline is quite similar to that involved in servicing a uucp request and is depicted in the following diagram:

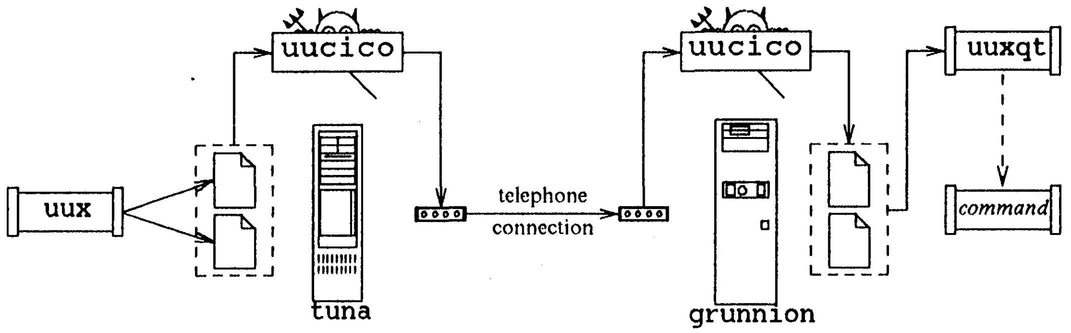


Figure 10-3: Remote Command Execution With uux

As before, several files are spooled to a uucp directory, where they are interpreted by the uucico daemon. One of the files is the work file, in this case the file ch.1. The other file spooled by uux is an execute file, which contains information about the origin of the request and about the processing the remote system needs to do in order to satisfy the request. The local uucico daemon begins a conversation with the uucico daemon on the remote system, at which time the execute and work files are transferred to the remote system. The execute file is interpreted by the uuxqt program on the remote system, which launches the command requested by the remote user. In this case, the work file is spooled to the line printer system.

Many other services can be built on top of these two basic services. A prime example of a software package that often rides piggyback on uucp is the USENET network. The USENET network software, netnews, oversees the storage of news articles on the local file systems and manages all user interaction with those articles (such as reading them, replying to them, etc.). The netnews programs also decide whether an article received by a machine (or originating on that machine) should be transmitted to some neighboring machine. The actual transmission of news articles from one machine to another, however, is handled entirely by the uucp networking software.

2. Overview of the Uucp System

The programs and files that make up the uucp system are found in three principle directories, as depicted in the following diagram. The programs in /usr/bin constitute an ordinary user's primary interface to the uucp system. It is by running one of these programs (usually uucp or uux) that a user sets in motion the elaborate chain of events that make up uucp network activity (for example, a file transfer from one machine to another or the remote execution of a command). With two exceptions, all of these programs are owned by the uucp administrator's account, uucp, and run setuid uucp and setgid daemon. These commands need to be installed setuid and setgid in order to be able to read the various configuration files in /usr/lib/uucp and to create files in the uucp spool directories. The uucp programs maintain data security by preventing normal users from reading or writing anything in its spool and administrative directories. The only way that a normal user can insert a file into one of the spool directories, and hence queue a file for transmission to a remote system, is by running a user-level command which will do that for him. The two programs that are *not* owned by the uucp account are uuencode and uudecode, which are two utilities that provide a means to encode and decode 8-bit binary data files for transmission across 7-bit data lines. These programs do not need to read or write anything located in the restricted uucp directories, so it is safest to have them owned by root.

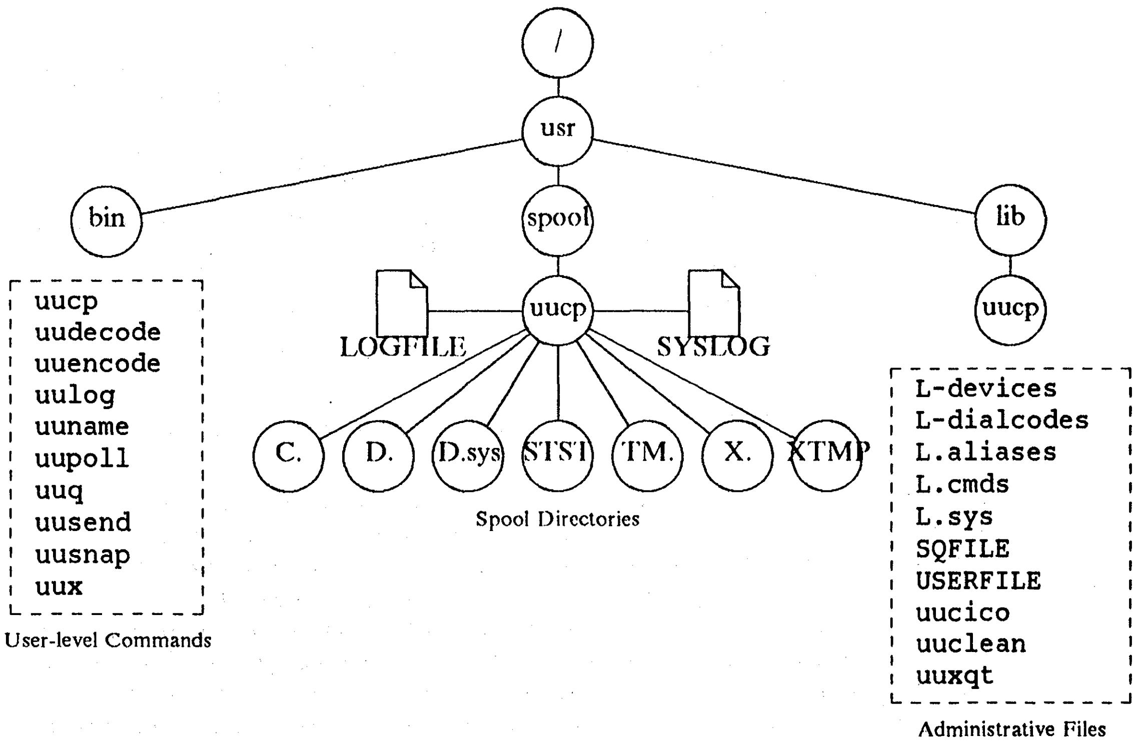


Figure 10-4: Location of uucp Files and Commands

2.1. Using Uucp Over A Local Area Network

As indicated in the introduction above, it is possible to route uucp traffic over a local area network. If your machine is connected to such a network, it is generally preferable to use it rather than direct serial lines or modems to send and receive uucp network traffic, largely because file and data transfers will occur more quickly and more reliably than using traditional serial connections. In addition, TCP/IP network communication can handle full 8-bit data and file transfers, thereby obviating the need to filter binary files through the utilities `uuencode` and `uudecode`. Finally, since two systems located on the same local area network are *ipso facto* connected by a permanent communications link, there is usually no need to wait until such a link can be established, as commonly happens when modems must be used to link systems. Note however that uucp still operates as a "store and forward" network, even when communicating with other systems over a local area network.

To use the uucp programs over a TCP/IP link, you must add an entry to the network services data base file, `/etc/services`. The appropriate entry looks like this:

```
    uucp    540/tcp    uucpd    # uucp on TCP/IP
```

This indicates that the `uucpd` daemon will be listening on port number 540 using the TCP protocol.

3. Uucp System Maintenance

The uucp system requires periodic maintenance to make sure that sites are being polled correctly, to remove old spool files, to rotate log files, and to clean up garbage left in uucp directories.

These and other actions are typically performed with the assistance of the `cron` daemon by including some entries into the configuration file, `/usr/lib/crontab`. Generally, `cron` is configured to execute one of several scripts which call the appropriate utilities to perform the desired actions. As a result, once you have suitably modified the `crontab` file and installed the scripts on your system, uucp maintenance will proceed fairly automatically. The `crontab` file is discussed in detail in Chapter 16.

The specific actions and their frequencies required for uucp system maintenance may vary among installations, but the following are some common tasks:

- Poll remote systems, until an answer is received, on an hourly basis. To do this, you use a program called `uupoll`, which resides in `/usr/bin`. `Uupoll` does this by placing a request for a null job in the queue for the remote system, then invoking `uucico`. For more information, see `uupoll`(8).
- Clean up garbage files in the spool (`/usr/spool/uucppublic`) on a daily basis. This is done using `uuclean`, which resides in `/usr/lib/uucp`. This program is typically started by the daemon each day to remove files that are more than three days old from the spool directories. (These are usually files relating to jobs that could not be completed.) You can, however, specify a time period other than three days, as well as a directory, one or more file prefixes, the level of debugging output you want produced, and that mail be sent to the owner upon a file's deletion.
- Remove old log files (found in `usr/spool/uucp` with names beginning with LOG) on a daily or weekly basis, depending on the number of files generated and available space.

This page intentionally left blank.

CHAPTER 11

Implementing Local Area Networks

1. Introduction

For uses such as sending, receiving, and forwarding electronic mail, or for distributing news articles in the USENET network, the uucp family of programs serves splendidly. It provides error-free transmission of data across phone lines and local area network cables between cooperating systems, and it allows the system administrator to determine exactly when, if ever, such transmission shall take place. For many important uses, however, the batch type of networking provided by the uucp system is inadequate. For example, uucp has no capability to manage interactive login sessions from a local machine to a remote host. While it is possible to use other IBM/4.3 system utilities (such as `cu` or `tip`) to login to remote systems over normal phone lines or directly-connected serial lines, it is generally quicker and more reliable to use a *local area network* to accomplish this. The IBM/4.3 system supports local area network access using both Ethernet and Token-Ring hardware.

Typically a local area network consists of numerous machines linked to a single hardware device, as depicted in the following illustration.

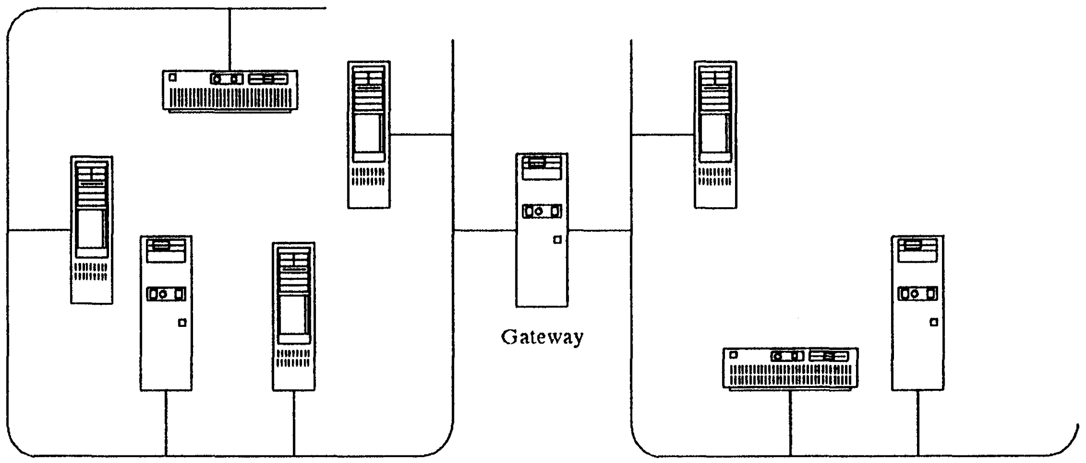


Figure 11-1: Local Area Network Hardware Configuration

Because there is an abundance of information on local area networks, this chapter does not discuss them in detail. For information, you should see the following articles:

- Section 5 of "Installing and Operating Academic Operating System 4.3" in Volume II of the IBM/4.3 documentation
- Section 15 of "4.3BSD Network Implementation Notes" in the *UNIX System Manager's Manual*
- Section 11 of "Name Server Operations Guide" in the *UNIX System Manager's Manual*

- Section 8 of "Timed Installation and Operation Guide" in the *UNIX System Manager's Manual*

CHAPTER 12

Managing USENET

1. Introduction

USENET is an electronic network that links UNIX-based systems (including IBM/4.3 systems) around the world. It is used for exchanging information, posting questions, distributing public-domain programs, and a variety of other activities; it functions as a sort of electronic bulletin board containing a large number of news groups. Unlike some bulletin board systems you may be familiar with, however, USENET does not operate on a single host machine and there is no central administrative authority that collects and distributes news messages. Rather, USENET messages are transmitted from the machine they originate on to neighboring machines, and from there onto more and more distant machines until messages have been distributed to all interested machines located in the messages' distribution area. Most machines subscribe only to those news groups that are of interest to the users of the machine, largely because subscribing to all news groups can consume large amounts of disk storage space. Some machines, though, transmit all active news groups, whether or not they are of interest to the users of that machine; such sites are known as "backbone" machines.

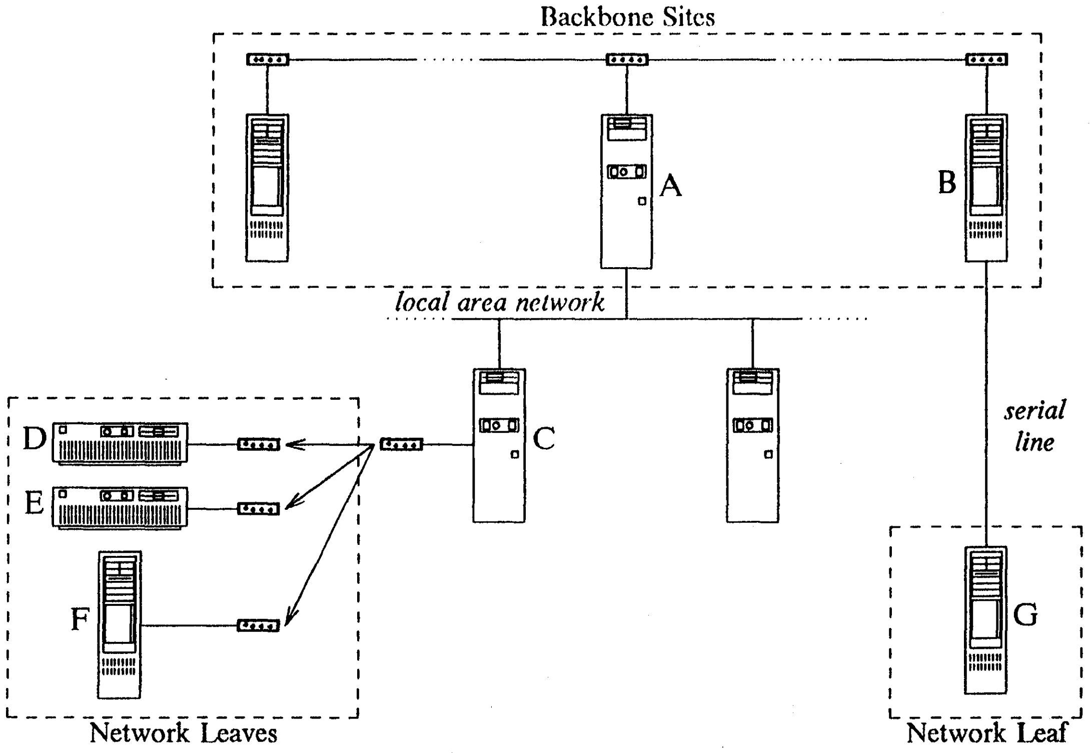


Figure 12-1: A Typical Segment of the USENET Network

In this diagram, machines A and B are backbone sites; they exchange all news articles that come to them from other backbone sites and from machines they serve.[1] Machine C is not a backbone site, but it still subscribes to all active news groups because it is a news source, or "feed", for machines D, E, and F. This kind of USENET host is generally called a "secondary" site. Since machines D, E, F, and G do not serve as a feed for any other machines, they are sometimes referred to as USENET "leaves".

As you may have guessed from the figure above, USENET is a logical network, in the sense that it rides piggyback on existing networking software such as uucp and the Internet protocols to move articles from site to site. Accordingly, USENET messages may be transmitted over phone lines, through direct serial lines, through Ethernet cables, across Token-Ring connections, or through whatever physical hardware is used to connect one machine to its news feed. The actual underlying hardware and software protocols used in the transmission of articles are invisible to the USENET users, since their interactions with USENET messages are always accomplished by a news reading and posting program (of which there are several).

From an administrative point of view, USENET installation and maintenance is reasonably straightforward, once you have a working network connection with a site that agrees to act as a news feed. (For information on installing USENET, see "USENET Version B Installation" in the *UNIX System Manager's Manual.*) USENET is designed so that most administrative functions can be performed automatically, either by receipt and processing of certain control messages sent across the network itself, or by periodic invocation of various commands by the `cron` daemon. The primary tasks involved in acting as the USENET administrator for your site include making sure that news articles and log files do not consume too much disk space and keeping configuration files up to date. The USENET administrator will also be called upon to add or remove news groups and to monitor log files used by the news software. Finally, you will also have to ensure that your local user community makes intelligent and polite use of the facilities provided by the USENET network.

2. Overview of USENET Operation

USENET articles are posted, passed from site to site, and ultimately read by using a set of programs collectively called the `netnews` programs. These programs are in the public domain and may be obtained from most USENET sites. If they are not already installed on your system, you will probably obtain the source code from your news feed; see the USENET installation article referenced above for more information.

To illustrate the overall operation of USENET, let us follow the path of a news article from its original posting to its reading on some other system. First and foremost, a news article is *posted* by some user at some USENET site. There are several ways to post an article, either by using a news-posting program (such as `postnews` or the newer `Pnews`) or by sending the article by electronic mail to an account on a remote system that receives news articles. If the article is posted using the `mail` program, it will automatically pass through the `recnews` program. In either case, the article will then be handed over to the `inews` program, which determines how the article should be sent from the local machine to its USENET neighbors.

There are two primary methods for getting news articles from one machine to another. If the originating machine is connected to a network that allows remote command execution, then the article can simply be sent as the standard input to the `rnews` program running on the receiving machine. For example, if the two machines are connected by the uucp network, then `inews`

[1]There are other requirements as well that a machine must fulfill in order to be considered a backbone site. A backbone site must exchange every non-local newsgroup that it receives with at least two other backbone sites (or with the main feed for a particular geographic area), have the disk capacity to handle the volume of net traffic, and run a recent version of the `netnews` software. In addition, a backbone site must agree to be advertised as a backbone site.

will run a command like the following:

```
uux - -r -z remote!rnews
```

On the other hand, news articles can be sent through an inter-machine mail link. In this case, the mail will be sent to an account named `rnews` on the remote system, which will probably call the program `uurec` to process the incoming mail, strip off mail headers, and pass the news article to `rnews`. The different paths for originating and transmitting news articles are illustrated in the following figure.

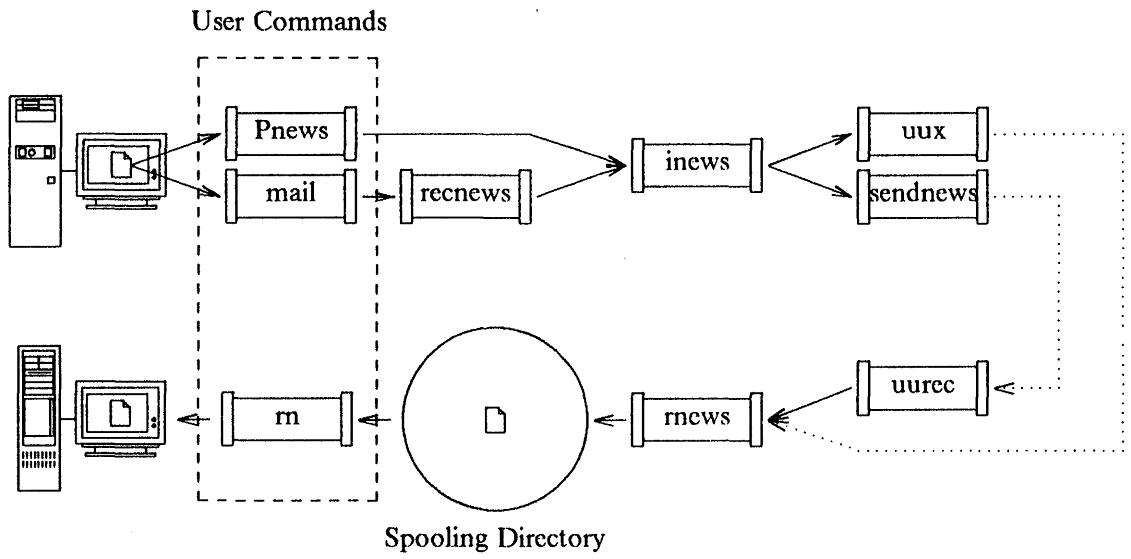


Figure 12-2: Overview of News Posting and Reading

An article passed to `rnews` is checked against a history file listing the articles already received on the local system. If the article has not yet been seen, it will be forwarded to USENET neighbors that the remote system feeds. The `rnews` program will also place the news article into an appropriate spot in the file system hierarchy, where they are accessed by a news-reading program such as `rn` or `readnews`. Typically the news articles are placed into subdirectories of the `/usr/spool/news` directory, as indicated in the following figure:

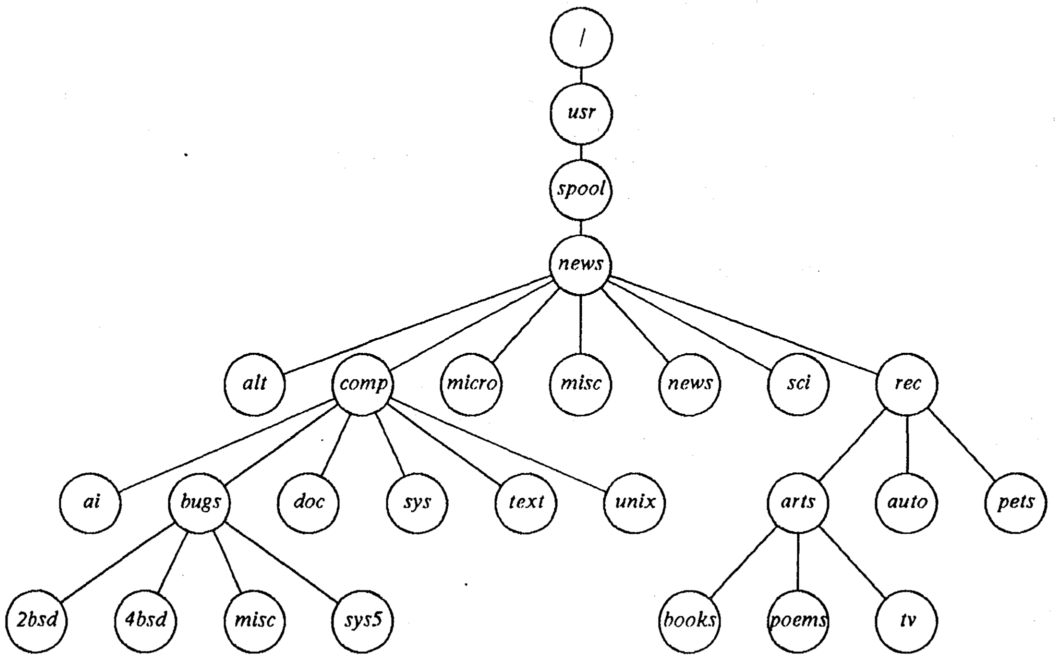


Figure 12-3: Part of the News Directory Hierarchy

3. Maintaining the News System

Maintaining the new system is largely a job of ensuring that files no longer needed in the system are discarded. Specifically this applies to history and log files, which will grow with use and can become a problem if not discarded when appropriate, and news groups that are no longer active. As administrator for the news system, you should do the following:

- Use the `expire` program (which is discussed later in this chapter) to delete lines from the history file relating to articles that have been deleted. The history file contains information on the articles you've received from other sites; USENET checks it when attempting to deliver a new article. If the article has already been received at your site, USENET will not duplicate it. You may want to manually check the history file every few months to make sure that `expire` is removing the appropriate lines. Be sure, however, that you don't completely discard the history file, in case a site attempts to send you an article you recently received.
- Use the `trimlib` script to keep the log file from becoming too large. You can install `trimlib` in `/usr/lib/news` and add an entry to your `crontab` file (which is discussed in Chapter 16) to automatically invoke this script once a week.
- Make sure to remove inactive newgroups. To do this, run the shell script `rmgroup` (which resides in `/usr/lib/news`) with the name of the newsgroup you want to remove as the argument.

4. Creating a New Newsgroup

Occasionally, someone at your site may suggest that a new newsgroup be created in order to serve a need that is currently not served. Since there is no central USENET administrative authority, it is possible for the USENET administrator at any site to create a newsgroup and then inform the net of the existence of the new group. Other sites are then free to subscribe to the new group or not to subscribe, as they wish. It may also be necessary to create newsgroups intended for local distribution only. The steps required to establish a new newsgroup in each of these two cases are listed below.

4.1. Creating USENET Newsgroups

Generally, it is *not* a good idea to add new groups to the existing USENET groups without first determining that such a newsgroup is really needed. It is possible that the discussion you wish to carry on fits nicely into an existing newsgroup, or that it would at least be tolerated by an existing newsgroup. Even if there is no current forum for your topic, it is also possible that there is insufficient interest in that topic to warrant the creation of a new group. It is preferable to avoid a proliferation of newsgroups and you are likely to receive some unfavorable reactions if you create a new USENET group without first following these steps:

(1) First and foremost, determine if a new newsgroup is really needed. Scrutinize lists of existing groups to see if one of them will accommodate the topics you wish to discuss. If the traffic in a particular related newsgroup is not too heavy, chances are that the added discussion will be welcomed there and there will be no need to create a new group. For example, the group `comp.lang.postscript` is devoted to discussing the PostScript page description language, but it is also used to trade actual PostScript programs, thereby obviating the need for a separate `comp.lang.postscript.sources` group.

(2) Select an appropriate name for the new group, and decide if you want it to be a moderated newsgroup or not. The name should be as short and pithy as possible, consistent with the requirement that it should try to exhibit its relation to existing groups.

(3) Post an article to the group `news.groups` describing your proposed new group. Request that comments, both pro and con, be mailed to you rather than posted directly to the network. Also cross-post your article to related newsgroups, but be sure to set the 'Followup-to' field so that posted responses go only to the `news.groups` group.

(4) Wait a few weeks and then review the comments that you receive by mail and on the network. Pay special attention to any objections that are expressed. At this stage it may be necessary to refine or modify the name and direction of the new group. If you do so, start the whole process over again, notifying the `news.group` subscribers of the modifications.

(5) Collect and categorize the responses you have received by electronic mail. There is no magic number of votes needed to justify the creation of a new group, but currently it is recommended that the positive responses must outnumber the negative responses by a margin of at least 100. If you cannot find at least that many people in the projected distribution area who would actively read and contribute articles, then the group is probably not needed. If you do get a significant response, however, then you should summarize your totals and post an article to `news.group` containing the summary. Also, this article should include the names and addresses of those expression an opinion.

(6) Send mail to `backbone@rutgers.edu` summarizing your results and asking that the new group be created. You can of course issue the appropriate control message yourself, but many sites will ignore it unless you are a recognized backbone site. If the new group is to be moderated, include all the relevant information about who is going to do the moderation.

If you follow these steps precisely and manage to elicit the appropriate amount of user response, then the new group probably will be created by the administrators at Rutgers.

4.2. Creating Local Newsgroups

Local newsgroups may be distinguished from those having a larger distribution by the fact that names of local newsgroups contain no prefix (and hence no periods). If a group is to be maintained entirely locally, so that messages are never forwarded to any neighboring machines, then the steps outlined in the previous section may be skipped and the USENET administrator may simply create the group. For example, to create the local newsgroup called 'general', give the command:

```
# inews -C general
```

The appropriate spooling directory will be created the first time an article is posted to the new group.

5. Expiring Old Articles

By default, a news article expires two weeks after it has been received on the host system, whether anybody has bothered to read the article or not. You can configure the cron daemon to remove expired articles automatically by inserting the following line into the configuration file, /usr/lib/crontab.

```
15    23    *    *    *    root    /usr/lib/news/expire
```

At 11:15 p.m. of each day, the program /usr/lib/news/expire will scan the news directories, searching for files that have reached their two-week limit. If any such files are found, hey will be removed from the system.

If your system is short on disk space, you can decide to expire articles sooner than the two week default, one one of several ways. First, you can change the constant DFLTEXP in the netnews software source code and then recompile the code. As distributed, DFLTEXP has a value of 1,209,600 seconds (or two weeks), which you may alter to suit local preferences.

A slightly simpler way to modify the two week default for expiring articles is to provide an argument to the expire command that you put into your crontab file. The -e option, followed by a number, will cause articles older than that number of days to be expired. For example, the crontab entry:

```
15    23    *    *    *    root    /usr/lib/news/expire -e 7
```

will expire articles after 7 days. In theory, this should reduce the amount of disk space consumed by USENET articles by approximately one-half.

6. Control Messages

Much of the day-to-day administration of the network news is handled automatically by the news system itself. For example, suppose that a user appeals to the network to help solve some problem, but then manages to solve it locally. In this case, there is very little use in having the message read by the entire news-reading community in the distribution area of the message, or in having the message further distributed. Instead, the original poster can request that all systems that have already received the message cancel it, so that it is not read or replied to by future news readers. To do this, the user may send out what is called a "control message".

A control message is simply a news message whose header contains a line beginning with the keyword 'Control'. The remaining portion of the control line is the message to be acted upon. For example, here is the relevant line from a typical control message:

```
Control: cancel <1987Sep16.144435.26473@nat.ucbvax.berkeley.edu>
```

Control messages are intended primarily for communication among USENET systems and not for human users or administrators of such systems. When the netnews system (typically rnews) receives and recognizes a control message, it will act upon it immediately, unless instructed differently. It is possible to have the news software queue the control messages for manual processing by the local USENET administrator, but this is not generally recommended since it invariably leads to delays in processing control messages.

The important part of a control message is just the news header itself, which contain the control line. The body of the message (i.e., the part that follows the header) is usually ignored, although it can be used to explain the reasons for the control message. Only the `checkgroups` control message contains a message body that is important to the receiving system.

For compatibility with earlier news systems, messages having a newsgroup '`all.all.ctl`' are also interpreted as control messages. In addition, if such a message header does not contain a line beginning with the keyword "Control", the Subject line is used instead. Also, if the first four characters on a "Subject" line are "cmsg", the remainder of that line is interpreted as a control message.

There are currently eight different control messages, each specified by a keyword command like `cancel` above. The keywords, the necessary arguments, and their actions are:

Message	Arguments	Description
`cancel`	*message-id*	If the message with the specified identification number *message-id* is present of the local system, then it is cancelled. If that article is not present, then the control message will *not* be sent on to neighboring sites. Only the author of a message or the local USENET administrator may issue such a control message.
`ihave`	*message-id*	The sending host has the specified message *message-id* and is prepared to forward it to the host receiving the control message, if requested. Normally, all articles in the appropriate newsgroups are send to a host, which then consults a history file to see if it has already been received; if so, the newly-received article is thrown away. The `ihave` control messages allows a site to determine whether an article should in fact be send to another site or not.
`sendme`	*message-id*	The host issuing this control message wants the specified message *message-id* sent to it. This message is typically used to reply to an `ihave` control message.
`newgroup`	*name*	The specified new group *name*is added to the active file and mail will be sent to the local USENET administrator indicating that this was done. A further argument '`moderated`' may be present, indicating that the new group will be moderated.
`rmgroup`	*name*	The specified group *name* is removed from the local system's active file. If the MANUALLY compile flag was not specified at software installation time, then the articles belonging to this group, the group directory, and the appropriate line in the active file will be removed.
`sendsys`		The `sys` file, containing a list of the local system's neighbors and newsgroups sent to them, is to be sent by return mail to the originator of the control message. This message is used to keep USENET maps up-to-date and to determine which sites are receiving network news.
`version`		The name and version number of the news software is to be mailed back to the originator of this control message.
`checkgroups`		The message containing this control line is a list of all active newsgroups, together with a short description of each group. The body of the message is piped through the program `checkgroups`, which will update the local newsgroups file, add any missing newsgroups, and mail a message to the USENET administrator concerning old groups which should be removed.

Table 12-1: USENET Control Messages

Any unrecognized message keywords will cause an error message to be mailed to the local USENET administrator, so part of your job as system administrator will be to inspect this mail and act accordingly. As time passes, additional keywords may be defined, in which case the source code (in particular, the file `control.c`) will need to be modified to recognize new keywords and to perform the appropriate actions when they are received in control messages.

There are three ways to send control messages:

- To post network-wide control messages, use `net.msg.ctl`.
- To send a restricted broadcast of a control message, use `btl.msg.ctl`.
- To send a control message to a particular system, use `to.`*systemname*`.ctl`.

7. Batch Processing of Articles

If your news feed communicates with you via the uucp network and you subscribe to a reasonably large number of newsgroups, then you will probably want USENET articles to be sent to you in a *batched* and *compressed* form. In this form, many articles are combined into a single file which is then compressed in order to reduce the size of the transmission. When such a transmission arrives at your site, it is uncompressed and then unbatched, and the individual articles are placed into the proper locations in the news directories.

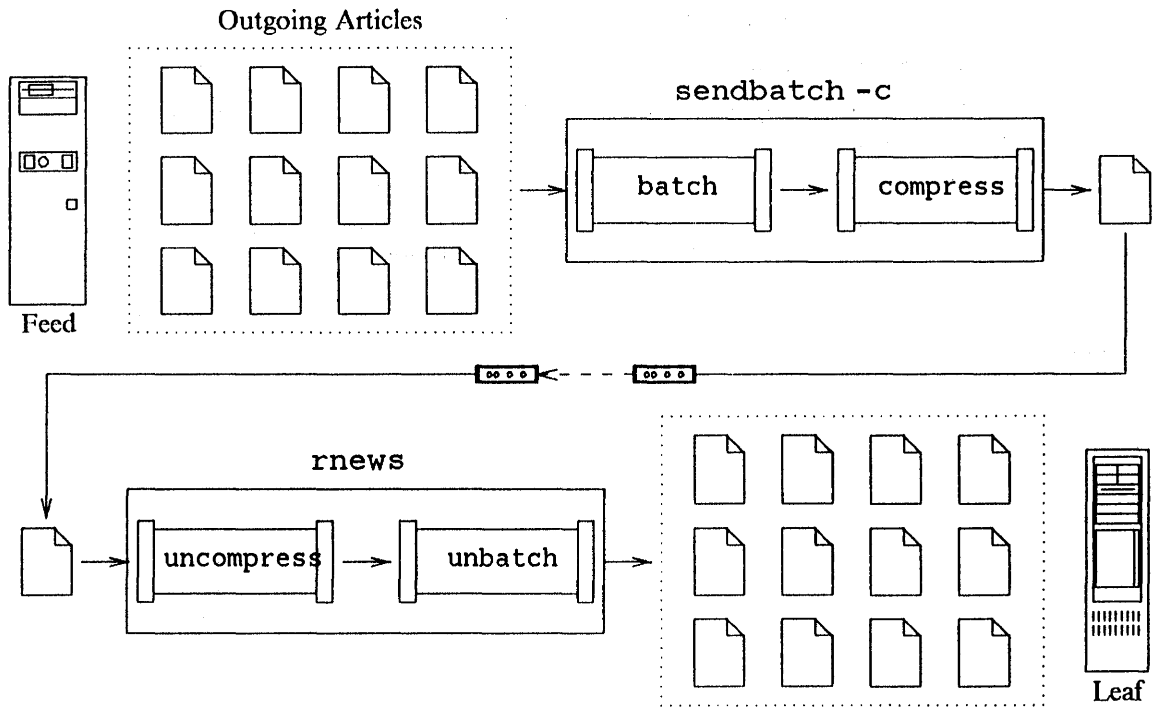


Figure 12-4: Batching and Compressing News Transmissions

The alternative to batching is to have uucp execute one command for each separate news article, thereby increasing the amount of work required to get the news articles to your site. Similarly, compressing the batched articles can reduce the total file size by as much as 50%, effectively halving the transmission time and expense.

8. User Education

USENET survives solely by the good graces of those who provide the hardware and administer the network software at each site. Participation in the USENET network is completely voluntary, and at any time, any site on the network is free to discontinue transmitting USENET articles, if the costs should become too great for that site to bear or if it proves to be a nuisance to continue USENET services. It is important therefore that your local users community be made aware of certain rules of etiquette governing USENET use. For example, a site that floods the net with numerous sizable, vitriolic, and largely useless postings, or with obviously self-serving advertisements, is abusing the courtesy of other USENET sites and may risk losing its news feed. In order to help avert such a situation, the system administrator must educate the local user community to whatever degree possible. On-line USENET documentation should be made available and users should be encouraged to read them. In addition to the on-line documentation, users should be directed to the following two articles reprinted in the *User's Manual Supplementary Documents*, "How To Read the Network News" (USD-9) and "How To Use USENET Effectively" (USD-10).

9. Tips on Managing USENET

- If you find that USENET articles have consumed more disk space than you would like, you can manually expire articles in certain specified newsgroups by invoking the `expire` command with the `-n` argument, which will cause the immediate expiration of all newsgroups listed. For example, the command:

  ```
  # /usr/lib/news/expire -n rec.*
  ```

  will expire all articles in the recreation newsgroups.
- If the transmission and processing of news articles puts a significant load on your machine, then arrange to have these tasks done at night.
- Never expire articles while unbatching of incoming mail is occurring.
- It is highly recommended that you subscribe to the newsgroup `news.admin`. This group is devoted to a discussion of `netnews` administration.

CHAPTER 13

Accounting

1. Introduction

In many computing environments, it is essential to keep an accurate record of who is logging into the system and what they are doing while connected to it. Typically, such information is used to determine account charges and billing information for the users of a system. This information can also be used in a variety of other ways, however. For example, in the event of a system break-in by an unauthorized user, a complete record of logins may provide some clues as to the identity of the culprit and the method used to gain access to the system. It may also be useful to know which commands are being used most often on a particular system so that they can be given priority in the software maintenance schedule. Also, by maintaining a record of who is using a system, the system administrator can help balance the distribution of the limited resources available among the entire user community. By monitoring login times, the system administrator can determine the optimal time to perform large system maintenance tasks such as file system backups.

The IBM/4.3 system can be set up to provide two types of accounting information: user connect time accounting and system resource accounting. This chapter describes, for each of these two different accounting systems, how it works, how to start it, how to stop it, and how to summarize accounting records.

2. User Login Accounting

The simplest type of accounting provided by the IBM/4.3 system is called user login accounting or user connect time accounting. User login accounting provides the system administrator with information about which users are logging in to the system, when they are logging in, and how long they remain logged in. It maintains records on who is currently using the system and on who has used the system in the recent past.

The system uses two main files to store user login accounting information: `/etc/utmp` and `/usr/adm/wtmp`. Neither one of these files is in a human-readable form, so you will never alter or view their contents directly. Rather, various system daemons and utilities are used to update and summarize the information contained in these files, as described in the following sections.

2.1. Starting Up Login Accounting

Each time a user logs into the system, the `login` program attempts to write an entry into the file `/usr/adm/wtmp`. An entry is also added to this file, if it exists, by the `init` command each time a user logs out, thereby maintaining a complete record of how long the user was logged in to the system. If the file `/usr/adm/wtmp` does not exist, however, then no user connect time accounting is done. In that case, the system administrator will have no way of knowing who has logged into the system in the recent past or how long they remained connected to the system. If the file does exist, however, then user connect time accounting will be done automatically. This means that there is no special command that must be run in order to turn on connect time accounting if the file `/usr/adm/wtmp` already exists. If you want to enable connect time accounting but that file does not exist, simply create it 0-length, as follows:

```
# touch /usr/adm/wtmp
```

The file /usr/adm/wtmp is a binary data file. In addition to user logins and logouts, it also maintains data on system reboots and date changes, and if left undisturbed, it will grow without limit. Consequently, any needed information should be extracted from it periodically and the file should be truncated to 0 bytes. You may perform this truncation by executing the command:

```
# cp /dev/null /usr/adm/wtmp
```

After collecting any desired user accounting information (as described below) and before re-initializing the accounting file in this manner, you may wish to copy it onto a backup medium such as streaming tape or floppy diskette. (Consult the chapter on performing backups for information on doing this.) You may also wish to rotate the file /usr/adm/wtmp, as described in complete detail in Chapter 16.

2.2. Listing User Login Sessions

A record of who has logged on the IBM/4.3 system in the recent past may be obtained with the last command. The last command will look into the file /usr/adm/wtmp and extract information about a user and a teletype, or about a group of users and teletypes. For example, the last command with no arguments will print a record of all logins and logouts, in reverse order. The beginning of a typical output might look like this:

```
% last
smith   ttyp0          ibmpa        Fri Mar 26 13:46   still logged in
smith   ttyp0          ibmpa        Fri Mar 26 12:22 - 13:34  (01:12)
smith   ttyp0          ibmpa        Fri Mar 26 12:20 - 12:21  (00:01)
judy    ttyaed                      Thu Mar 25 09:59 - 17:07  (07:08)
reboot  ~                           Thu Mar 25 09:54
smith   ttyp0          ibmpa        Mon Mar 22 15:37 - 15:42  (00:05)
judy    ttyaed                      Mon Mar 22 12:17 - 15:00  (02:43)
jr      ttyaed                      Mon Mar 22 12:10 - 12:16  (00:06)
judy    ttyaed                      Mon Mar 22 10:09 - 12:10  (02:00)
judy    ttyap16                     Mon Mar 22 10:08 - crash (2+23:45)
judy    ttyap16                     Mon Mar 22 10:07 - 10:08  (00:00)
judy    ttyaed                      Mon Mar 22 10:05 - 10:09  (00:04)
smith   ttyp0          ibmpa        Mon Mar 22 10:00 - 12:40  (02:39)
jr      ttyp0          bullhead     Mon Mar 22 09:27 - 09:27  (00:00)
judy    ttyaed                      Sun Mar 21 15:24 - 17:40  (02:15)
```

The first column lists the user name given at login time; the second column lists the name of the teletype port through which the login occurred. If the login session occurred across a network, then the third column will contain the name of the remote host from which the connection was made; otherwise the third column is blank. Finally, the remaining columns list the login and logout times, along with an indication of the total elapsed time for that login session.

The last command can be instructed to give login data about a particular user, or a particular teletype port, instead of the default complete listing. For instance, to obtain information about all login sessions by the user judy, a user would type:

```
% last judy
```

and to obtain login information about all logins that occurred through the system console, a user would type:

```
% last console
```

These two types of options may be combined, so that the command:

```
% last judy console
```

will list all logins by judy on the system console. There is a special string recognized by the last command, 'reboot', which requests that only reboots of the system be listed. So the command:

```
    % last reboot
```

will list, in reverse order, all system reboots recorded in the file `/usr/adm/wtmp`.

2.3. Summarizing Connect Times

Although the `last` program provides a complete listing of all recent user logins, it does not provide any totals from among the data listed. To obtain a listing of total connect times, you may invoke the program `/etc/ac`. This program produces a printout of total connect times for each user who has logged in during the life of the current accounting file, `/usr/adm/wtmp`. For example, if invoked with no arguments, a total of all connect times by all users is printed:

```
    # /etc/ac
         total    429.75
```

If, on the other hand, you wish to see how much connect time a particular user has consumed, you may invoke the command with the `-p` option. For instance:

```
    # /etc/ac -p
         gordon   312.05
         jr        52.26
         monroe    64.60
         susan      0.16
         ping       0.21
         yokela     0.05
         kevin      0.57
         carl       0.07
         mar        0.01
         root       0.00
         total    429.97
```

A further option, `-d`, requests that only a daily total be printed; this restricts the listing to connect time within each midnight-to-midnight period:

```
    # /etc/ac -p -d
         gordon     7.69
    Nov  1     total     7.69
         gordon     8.15
         jr         6.52
    Nov  2     total    14.67
         gordon     8.17
         jr        20.63
         monroe     5.25
    Nov  3     total    34.05
         gordon    12.67
         monroe     5.01
         susan      0.01

         [lines omitted]

         gordon     7.35
    Nov 27     total     7.35
         gordon     2.67
         monroe     2.35
    Nov 28     total     5.02
```

As you can see, this kind of connect time summary is likely to be quite lengthy on a system with even a moderate amount of use.

2.4. Listing Current System Users

At any time, you may obtain a list of current users of the system by executing the `who` command. You will obtain some output similar to this:

```
# who
gordon   ttyaed  Mar 28 10:51
monroe   ttyp0   Mar 28 11:03 (ibmpa)
```

The `who` command obtains this information by consulting the file `/etc/utmp`, which is a binary data file that contains information about all users currently logged in. As indicated by the printout, `/etc/utmp` contains four pieces of information about each current user: the user's login name (in the form of the user identification number), the teletype through which the user is logged in, the time that the user logged in, and the user's remote host, if that user is logged in across a network (indicated above in parentheses).

Unlike the file `/usr/adm/wtmp`, the file `/etc/utmp` will not grow without boundaries, since each time a user logs out (or hangs up the telephone, if logged in through a modem), the corresponding entry in `/etc/utmp` is removed by the `init` process. Therefore, there is no need to monitor the size of `/etc/utmp` or to back it up onto a secondary storage medium. It is, however, advisable to truncate this file at boot time, in order to guard against a possibly inaccurate file left over by a system crash. You may wish to include the following line in your local system start-up file, `/etc/rc.local`:

```
cp /dev/null /etc/utmp
```

If this line is placed there, the system will automatically trim the list of current system users at boot time.

3. System Accounting

System process accounting involves keeping track of exactly who is doing what on the system. The kernel implements process accounting by maintaining internal statistics about each process as it runs and then appending a record summarizing those statistics to a system accounting file. If process accounting is enabled, the following items will be monitored and reported:

- Name of each command or process run on the system.
- User time expired during the running of the process.
- System time expired during the running of the process.
- Total elapsed time for the process.
- Time of day at which process was initiated.
- Uid of person running process.
- Gid of person running process.
- Average amount of memory consumed by process.
- Number of disk I/O blocks used by process.
- Controlling terminal line.

In addition, the accounting record contains a flag indicating whether the process was killed by a signal, whether it was run by the super-user, whether it dumped core, and several other useful pieces of information about the process.

The accounting file maintained by the process accounting system, usually `/usr/adm/acct`, therefore provides a nearly-complete picture of who is doing what on the system, how long it takes, when they did it, and whether the command exited normally or not. Since it is designed to maintain such information about every process in the system, the system accounting file will inevitably grow quite large. It is the job of the system administrator to monitor the accounting file periodically, extract and summarize the necessary information, and then truncate the file to a reasonable size. The following sections provide detailed instructions on starting, stopping, and maintaining the process accounting system.

3.1. Starting Up Process Accounting

Like the user connect time accounting system discussed above, the process accounting system is entirely optional and does not have to be running in order for the IBM/4.3 system to provide services to local and network users. Unlike the login accounting system, however, which starts up automatically if the file `/usr/adm/wtmp` exists, the system administrator must explicitly tell the operating system to begin collecting process accounting information. This is done by invoking the command `/etc/accton` with some file name provided as an argument. Essentially, `/etc/accton` just passes the file name to the system call `acct()` which notifies the kernel to begin placing accounting records into the named file. Once accounting is enabled, the kernel will append a record to the accounting file as each process terminates. The record lists, among other items, the name of the process, the time at which the process was launched, the user and group identification numbers of the user who initiated the process, the average memory usage, and accounting user and system time.

On the IBM/4.3 system, process accounting records are usually kept in the file `/usr/adm/acct`. If your system is running in multi-user mode and you want to turn on process accounting, you can simply type the following command to the shell:

```
# /etc/accton /usr/adm/acct
```

Usually, however, the following lines are placed into the system start-up script, `/etc/rc`, in order to initiate system accounting automatically when the system enters multi-user mode:

```
/etc/accton /usr/adm/acct
      echo -n ´ accounting´  > /dev/console
```

You should make sure that these lines do not precede the commands that mount the file system holding the accounting file. Otherwise, the kernel will not be able to locate the accounting file and the `accton` command will fail.

3.2. Stopping Process Accounting

You may at times wish to turn off the process accounting system, so that records are no longer appended to the accounting file at the completion of each process. You will need to turn off system accounting, for example, if you want to unmount the file system containing the accounting file. If you do not stop the accounting before you attempt to unmount the file system, you will get the message:

```
/dev/hd0g: Device busy
```

To turn off process accounting, you must invoke the `accton` command with no arguments, as follows:

```
# /etc/accton
```

Once this command is executed, the kernel will cease appending accounting records to the system accounting file.

3.3. Listing User Commands

One utility that reads and interprets the system accounting file is the `lastcomm` program. This program gives information on commands that have already terminated and for which records have been entered into the accounting file. A typical segment of `lastcomm` output might look like this:

```
# lastcomm
sh          S      root      __         0.12 secs Wed Apr 13 13:45
atrun              root      __         0.08 secs Wed Apr 13 13:45
sh          S      root      __         0.09 secs Wed Apr 13 13:30
atrun              root      __         0.08 secs Wed Apr 13 13:30
df                 smith     tty00      0.25 secs Wed Apr 13 13:26
who                smith     tty00      0.06 secs Wed Apr 13 13:26
mail        X      smith     tty00      0.45 secs Wed Apr 13 13:25
rlogin             smith     tty00      2.42 secs Wed Apr 13 07:58
rlogin       F     smith     tty00      8.81 secs Wed Apr 13 07:58
sendmail     F     root      __         0.05 secs Wed Apr 13 13:20
```

The first column lists the command name. The second column may contain a flag indicating whether it was run by the superuser (in the example, 'S'), whether it ran following a `fork` but not an `exec` (in the example, 'F'), or whether is was terminated by a signal (in the example, 'X'). The remaining columns are fairly self-explanatory; there are the user name, controlling terminal, elapsed time, and time of day when launched.

As with the `last` command, you can supply certain arguments to the `lastcomm` command to restrict the records output to a subset of the complete list. For example, the command:

```
# lastcomm smith
```

will list the records for the user `smith`. Similarly, the command:

```
# lastcomm console
```

will list records for all processes launched from the system console. This can be useful, for example, if you know that some unauthorized user was able to gain access to the console. `lastcomm` will give you a complete list of the commands run by that user and perhaps allow you to isolate any damage that may have been done.

3.4. Maintaining Process Accounting Files

As indicated above, the main process accounting file `/usr/adm/acct` will grow quite large as processes are launched then terminate. It is useful to reduce the amount of disk space occupied by accounting records by summarizing the records in `/usr/adm/acct` and then truncating that file. The summaries are kept in two other files, `/usr/adm/savacct` and `/usr/adm/usracct`, which contain, respectively, a summary of the raw accounting data and a per-user summary of all accounting data. These summaries, especially the per-user summaries in `/usr/adm/usracct`, can be used to determine account charges.

The IBM/4.3 system provides the general administrative command, `/etc/sa`, to report on, maintain, and clean up the system accounting files. One of the most common uses of the `sa` command is to merge the raw accounting records contained in `/usr/adm/acct` with those records already summarized in `/usr/adm/savacct`. When invoked with the `-s` option, `sa` will summarize the records in the raw file and in the summary file and then create a new summary file containing all the previous accounting information. Note that `sa` will also output the new merged statistics, so if you merely want to merge the two files, you should redirect the output of `sa`, as indicated:

```
# /etc/sa -s > /dev/null
```

After this command has completed, the raw accounting file will contain just a single entry (that corresponding to the `sa` command itself).

4. Accounting Tips

- There is no system utility that indicates whether or not process accounting is enabled. (Part of the reason for this is that process accounting is handled directly by the kernel and not by some ancillary daemon or process.) You can determine this easily, however, by running the `lastcomm` command and comparing the date of the most recent command logged with the current system date. For example:

  ```
  # date ; lastcomm | head -1
  Wed Apr 13 13:02:40 PDT 1988
  date             root     console     0.05 secs Wed Apr 13 13:02
  ```

  The output indicates clearly that process accounting is currently enabled.
- You should note that the accounting system summarizing program provided with the IBM/4.3 system, `/etc/sa`, does not differentiate logins or system resource consumption by *group*. There is no way, short of writing your own shell scripts or programs, to obtain usage totals for a particular group. This reflects the decision to associate accounting programs with the user identification number (uid) and not the group identification number (gid). It is also probably the basis for the use of the word 'account' to describe the collection of directories, files, and other objects that altogether allow a user to log in to the system.
- Remember that the kernel writes process accounting records into the accounting file (usually `/usr/adm/acct`) when the process *terminates*. One consequence of this scheme is that no accounting records are kept for processes which never terminate, or for processes which are running when the system crashes. It may therefore be possible for very clever users to circumvent the process accounting system by arranging to crash the system at some appropriate time.
- In the IBM/4.3 system, the accounting file is typically `/usr/adm/acct`. As you have seen, it is possible to save process accounting records in some other location, but this practice is not recommended since several utilities (for example, `lastcomm`) expect records to be kept in `/usr/adm/acct`.

This page intentionally left blank.

CHAPTER 14

Administering Quotas

1. Introduction

Disk space for file storage is one of the most easily endangered resources of any computer system. Without some method of limiting individual space consumption, a file system can quickly be filled by indiscriminate accumulation of user files or by runaway processes that create large output files. The IBM/4.3 system includes a subsystem designed to allow the system administrator to set up *quotas* limiting the consumption of disk space by an individual user. When the quota system is operating, the administrator can place limits on the total amount of disk space that a user's files may occupy, on the total number of files that a user may own, or both. Moreover, the administrator can apply these two kinds of quotas on a per-user basis, allocating some users more space than others, or assigning identical quotas to all users.

For each of the two types of quotas available (total disk space and number of files), the administrator must set both a "soft" and a "hard" limit. The hard limit is the absolute maximum amount of disk space or number of i-nodes that a user can own. The quota system will not allow these numbers to be exceeded under any circumstances, so once they are reached, any attempts to consume more disk space or create new files will fail. The first time the hard limit is reached, a message will be sent to the user's terminal screen. The user must reduce space consumption (or the total number of files) in order to bring the usage under the hard quota, since no further resources will be allocated to the user. Only one such message will be printed.

The soft limit is the total number of blocks (measured in 1K) or i-nodes that the user is not expected to exceed. The soft limit acts as a cautionary boundary that space consumption is approaching the hard limit. If the user exceeds this number, a warning will be issued. If the user logs out without reducing below the soft limit the amount of disk space occupied or the total number of files (whichever limit was surpassed), the user will be warned once again at the next login session and the total number of remaining warnings is reduced by one. In all, three warnings are given before the user is considered to have exceeded the hard limit. No more resources will be allocated to that user; in particular, that user will be unable to log in until the disk usage is reduced below the quota.

You should note that the disk space quota subsystem is entirely optional. In the default system configuration, the quota system is *not* installed and users will be able to create as many files as they like, of whatever size they like, wherever in the file system they have write permission, until the available disk storage is depleted. In order for the size and/or number of user files to be governed by the quota subsystem, the system administrator must install the subsystem into the kernel and configure user quotas. This chapter describes the installation, configuration, and operation of the disk quota system.

2. Installation

The quota system is quite simple to use and monitor once it has been installed on your IBM/4.3 system. Installation itself is not difficult, but it does involve reconfiguring the operating system and selecting quotas for individual users, as described in the following sections.

2.1. Reconfiguring the Operating System

Before the system administrator can set quotas on system users, several steps are necessary to install and configure the quota system. First and foremost, the operating system (or "kernel") must be reconfigured to include routines used by the disk quota subsystem. This task was used as an example in Chapter 3 to illustrate how to reconfigure the operating system, and you should refer to that discussion for a complete step-by-step walk-through of the process. The remainder of this section provides a summary of the steps necessary to install the disk quota subsystem into the kernel. Before doing any kernel reconfiguration, however, you should make sure that this has not already been done for you. Run the command:

```
# quota
```

If the quota subsystem has already been installed, you will see the following message:

```
Disc quotas for root (uid 0):       none.
```

(The alternate spelling '`disc`' for '`disk`' in this and other messages from quota-related commands reflects the fact that the disk quota subsystem was developed by researchers in Australia.) If, on the other hand, the quota system has *not* been installed, you will see the following message:

```
There are no quotas on this system
```

You should proceed with the rest of this section only if you see the latter message.

To begin, you must include the following line in the system configuration file (probably a copy of `/sys/conf/GENERIC`, which we shall call 'GEN_QUO'):

```
options         QUOTA
```

This line instructs the system configuration process to include sections of code relevant to the operation of the disk quota system. If this line is not put into the configuration file, then the necessary code will not be included and you will be unable to establish disk quotas on your system.

When you have finished inserting this line into the system configuration file, you will need to create a target directory to hold several files that will soon be created. Then run the `config` program:

```
# mkdir ../GEN_QUO
# config GEN_QUO
```

If everything goes smoothly, the `config` program will create several files in the target directory. You can complete the system regeneration process by issuing the following commands:

```
# cd ../GEN_QUO
# make depend
# make vmunix
```

Finally, you will need to install the newly-created system image into the root directory and reboot the system. For example,

```
# cp /vmunix /vmunix.old
# cp vmunix /vmunix
# sync
# sync
# reboot
```

If you encounter any difficulties in this phase of the quota installation procedure, or if you are unfamiliar with the process of generating a new kernel, please refer to Chapter 3 and to the document "Building IBM/4.3 Systems with Config", in Volume 2 of the Technical Computing Systems documentation.

2.2. Setting Up Disk Quotas

The next step involved in installing the disk quota system is deciding which of the available file systems need to have their space regulated by user quotas. Generally, it is sufficient to place disk-space quotas only on those file systems that store users' home directories. If other file systems are in part dedicated to holding user files, then they too are good candidates for inclusion under the quota system. Possibly the /usr file system qualifies under this category, depending on your local usage patterns. For various reasons, however, it is recommended that spooling directories and temporary directories (such as /tmp) not be placed under the quota system.

Once you have determined which file systems shall be governed by quotas, the file /etc/fstab must be modified to indicate this information. Each file system listed in /etc/fstab that is to have quota-checking enabled must be indicated by a file type entry of the form 'rq'. For example, suppose that you wish to establish quotas on the /usr file system. The original entry for this file system in /etc/fstab looks like this:

```
/dev/hd0g:/usr:rw:1:2
```

To enable quotas on this file system, modify the entry so it looks like this:

```
/dev/hd0g:/usr:rq:1:2
```

Next, create a null file called quotas in the root directory of each file system on which a space limitation is to occur. For example:

```
# cd /usr
# touch quotas
```

The file quotas is used to maintain a record of disk quota limits and usage for the file system in whose root directory it is located.

Finally, the system administrator must decide how much disk space each user shall be allocated for each file system and then inform the system of these decisions. These space decisions will no doubt need to be based on local parameters such as the number of persons using the system, how important and space-intensive their projects are, how much free disk space is available, and other factors too numerous to mention here. Once these decisions have been made, they are implemented by invoking the edquota utility.

To illustrate the use of the edquota command, suppose that the user whose login name is guest is to be limited to 100 kilobytes of disk space and 100 files. Invoke the command:

```
# edquota guest
```

The edquota command creates a temporary file and calls up an editor on that file. The editor will be either vi or whatever editor is the value of the EDITOR environment variable. When you finish and write your changes, edquota will update the quota file quotas in the root directory of the specified file system. (You cannot edit the file quotas directly since it is stored in a binary format.)

For example, since no quotas have yet been placed on the user guest, the editor file will look like this:

```
fs /usr blocks (soft = 0, hard = 0) inodes (soft = 0, hard = 0)
```

Note that the quotas are specified in unit called "blocks". The edquota command interprets a "block" as 1024 bytes, regardless of the fact that the IBM/4.3 file system uses blocks that are at least 4096 bytes long (and may be much larger). The use of the word 'block' here is therefore unfortunate and is likely to cause confusion if you are not careful. When using edquota and the related utilities, always remember that what is called a block is just a kilobyte. The same warning applies to the du command.

Now change the four zeros to the appropriate value; for this example, you should place a hard limit of 100 kilobytes and 100 i-nodes (or files) and a soft limit of 10 percent less than the hard limit. Change the line so that it looks like this:

```
fs /usr blocks (soft = 90, hard = 100) inodes (soft = 90, hard = 100)
```

Then write the file and quit; you are finished establishing some sample disk quotas.

2.3. Checking Quotas for Consistency

After establishing some quotas on a file system but before actually starting up the quota-checking system, the system administrator should run the `quotacheck` utility in order to make sure that the quotas listed in the file `quotas` make sense for the file system it resides in. For example, having established a quota in the `/usr` file system for the user `guest`, you might give the command:

```
# quotacheck /usr
```

The `quotacheck` command examines a file system, builds a table of current disk usage, and compares this table against the quotas listed in the file `quotas` in the root directory of the specified file system. You may want to think of `quotacheck` as the quota system analogue of `fsck`. The `quotacheck` utility ensures, for instance, that the number of blocks actually owned by some user corresponds with the number listed as owned in the `quotas` file. If there is any inconsistency between the two figures, the `quotas` file is updated to reflect the actual usage. If, however, `quotacheck` finds no inconsistencies, then it exits silently (unless invoked with the `-v` option, which causes it to indicate the calculated quotas for each user on a particular file system).

Normally, `quotacheck` is run at boot time by placing the line:

```
/etc/quotacheck -a
```

into the file `/etc/rc.local`. The `-a` option requests that all file systems specified in `/etc/fstab` as having quotas in force be checked.

2.4. Starting Up the Quota System

The system administrator informs the operating system that quota-checking is to be turned on by invoking the `quotaon` command. The quota system can be made operational for a particular file system by providing the name of the file system as an argument. For example, the command:

```
# /etc/quotaon /usr
```

will enable quota-checking for the `/usr` file system. If no file systems are named as arguments but the `-a` option is present, then `quotaon` will read the file `/etc/fstab` and enable disk quotas on all file systems that have type `rq`, as explained above. In either case, the option `-v` may be specified in order to produce a "verbose" command output in which a message is printed for each file system where quotas are successfully turned on.

Typically, the line:

```
/etc/quotaon -a
```

is inserted into the system start-up file `/etc/rc.local` so that quotas are turned on automatically at boot time. This command should be run *after* the `quotacheck` utility.

3. Operating the Quota System

Operating the quota system includes the following tasks, discussed in this section: listing quotas, turning off quota checking, summarizing disk quotas, and compacting data.

3.1. Listing Quotas

A user may check the quotas that have been set by the system administrator by running the `quota` command. For example, if the user `guest` logs in and runs the command, that person may see something like this:

```
% quota   .

Disc quotas for guest (uid 98):
   Filesys current  quota   limit  #warns   files   quota   limit  #warns
     /usr       25     90     100              15      90     100
```

For each of the two types of disk space limited by the quota system (total disk space and total number of files), four numbers are reported: the current usage, the soft quota, the hard quota, and the number of remaining login warnings. In the columns headings, the soft quota is labeled '`quota`' and the hard quota is labeled '`limit`'. As you can see, the user `guest` has only 15 files which consume about 25 kilobytes of space, well below the soft limit set by the system administrator. As a result, no warnings have been issued yet.

The super-user may check on other users' disk quotas by supplying an optional argument to the command. For example, if the super-user runs the following command:

```
# quota guest
```

then the disk usage and established limits on the user `guest` will be printed. The output will be identical to that given above.

3.2. Turning Off Quota Checking

If the quota-checking facilities should need to be turned off for some reason, the `quotaoff` utility may be run. For example, to disable the quota checking system that is in force on the `/usr` file system, give the command:

```
# quotaoff /usr
```

The specified file system must be listed in the file `/etc/fstab` and must be mounted at the time the quota-checking is turned off. Upon successful completion of this command, the quota system will be disabled. Users will be able to exceed any quotas (either hard or soft) that would previously have applied to them.

Note that running `umount` on an idle file system will automatically cause the quotas on that file system to be turned off. Remounting the file system, however, will not cause the quota system to be reactivated.

3.3. Summarizing Disk Quotas

As noted above, the quota usage for any particular user may be listed by running the `quota` command. A more complete summary of the space usage and quota limits for a particular file system may be obtained with the `repquota` command. For example, a summary of disk usage and quotas in force on the `/usr` file system might look like this:

```
# requota /usr
                        Block limits                File limits
User            used    soft    hard  warn    used  soft  hard  warn
root     --    35632       0       0     0       1     0     0     0
daemon   --      141       0       0     0       1     0     0     0
notes    --     1110       0       0     0       1     0     0     0
uucp     --      745       0       0     0       1     0     0     0
guest    --       25      90     100     3      15    90   100     3
nobody   --       56       0       0     0       1     0     0     0
```

The column headings listed should be self-explanatory. The second column indicates whether either one or both of the block limit and the file limit have been exceeded. For example, if the user `guest` has 95 kilobytes of file storage, then the appropriate line might look like this:

```
    guest     +-     95     90     100     3     45     90     100     3
```

The `+-` indicates that the block limit has been exceeded, but not the limit on the number of files. Similarly, if both soft limits have been exceeded, the line from `repquota` might look like this:

```
    guest     ++     95     90     100     3     95     90     100     3
```

The `++` indicates that both soft limits have been exceeded. Notice that the user has not logged in since the quotas were exceeded, as indicated by the fact that there are still three warnings remaining. When the user `guest` next logs in, the following warning will be printed:

```
    Warning: too much disc space on /usr, 2 warnings left.
```

If the user ignores this warning and attempts to create additional files without first removing some, the kernel will issue the following message as soon as the hard limit is reached:

```
    DISC LIMIT REACHED (/usr) - WRITE FAILED
```

The only recourse for this user is to remove some files until the amount of disk space consumed is once again below the soft limit. If this warning appears when the user is in the middle of editing a document and does not want to lose extensive changes made to it, it is possible to write the file onto a file system where no quotas are in force. For example, from within the `vi` editor, `guest` could type:

```
    :w /tmp/ch.02
```

to write a copy of the file being edited (called '`ch.02`') into the temporary file space. It must be moved from there by the time the system is next rebooted, however, or the file may be lost forever.

4. Compacting Data

Users with chronic disk space problems (as indicated by continually approaching or exceeding their quotas) may be able to free valuable disk space by taking advantage of various data compaction utilities available on the IBM/4.3 system. These utilities allow the user to replace a file by a *compacted* or *compressed* version of that file. When the original file is a plain ASCII file (such as source code or straight text), the compacted file will typically be about half as large as the original. So, by carefully compacting files that are not used very often (but which could not be conveniently moved to a secondary storage medium such as magnetic tape), a user can maintain many more files and directories than the disk space quota would otherwise allow.

The recommended way to compact files is by using the `compress` command. For example, to reduce the amount of disk space occupied by the file `report.feb`, the user would issue the command:

```
    # compress report.feb
```

This command will create a new file called `report.feb.Z` in the current working directory and remove the original. (The suffix '`.Z`' indicates simply that this is a compacted version of a file.) The compressed file may later be uncompacted and restored to its original state with the `uncompress` command. For instance, to recover the file `report.feb`, just type the command:

```
    # uncompress report.feb.Z
```

Users should know that a compressed file cannot be edited using the normal IBM/4.3 system editors, nor can it be given as input to commands in the same way that the original file could. There is, however, a utility called `zcat` which operates exactly like the `cat` command, except that it outputs an uncompressed version of the file (without actually recreating the original file). By running `zcat` as the first command in a pipeline, you can send the original version of the file as input to other commands, even though only a compacted version of the file is available on disk. For instance, if the file `report.feb` contains `troff` input text, then the command:

```
    # zcat report.feb.Z | troff -me | lpr
```

will produce a printed copy of the report. Refer to `compress`(1) for complete details on these utilities. See also the manual entries for `pack`(1) and `compact`(1) for related data compaction utilities.

This page intentionally left blank.

CHAPTER 15

Handling System Messages

1. Introduction

It is an unfortunate fact of life that errors and other problems occasionally occur during the operation of any computer system. A problem may be a fairly minor one (such as the inability to deliver a piece of electronic mail because the addressee is unknown to the system), or it may be an extremely serious one (such as the inability to create a file because a file system is out of space). Yet no matter what the reason, when such exceptional conditions arise it is essential that a notification of the condition be sent to the system administrator or to other users who may be affected by it. This allows corrective or preventive action to be taken, and will help you keep the computer and its peripheral devices running smoothly. In addition, by collecting and logging diagnostic messages that arise from problem conditions in the system, you will be able to maintain an on-line record of error messages which you can monitor and summarize as needed.

The IBM/4.3 system includes a very powerful and flexible set of tools for monitoring the diagnostic messages that result from problems in the operating system, in peripheral devices, in daemons controlling certain machine resources, or in the execution of user-invoked commands. These tools can be configured to automatically route diagnostic messages to an appropriate place, whether to a log file, to the console screen, to the terminals of some or all currently logged-in users, or even to another machine on your local area network. This chapter describes the configuration and operation of these IBM/4.3 system message logging facilities. You should note that an earlier diagnostic message reporting and logging system, `/etc/dmesg`, has been rendered obsolete by the commands and daemons described here.

2. The System Message Log Daemon

All system messages are monitored by the daemon `/etc/syslogd`, which is usually launched when the operating system enters multi-user mode. This daemon reads from the device `/dev/log` (to read messages generated by local commands), from the Internet domain socket specified in `/etc/services` (to read messages generated by network activity), and from the special device `/dev/klog` (to read messages generated by the kernel). When a message arrives from one of these three sources, `syslogd` responds by taking an action indicated by an entry in the configuration file `/etc/syslog.conf`. Most commonly, the message will either be logged in a file or displayed on the system console screen (or both). Other possible actions include displaying the message on other terminal screens or sending the message to a remote computer on a network. The logical position of the `syslogd` is depicted in the following figure:

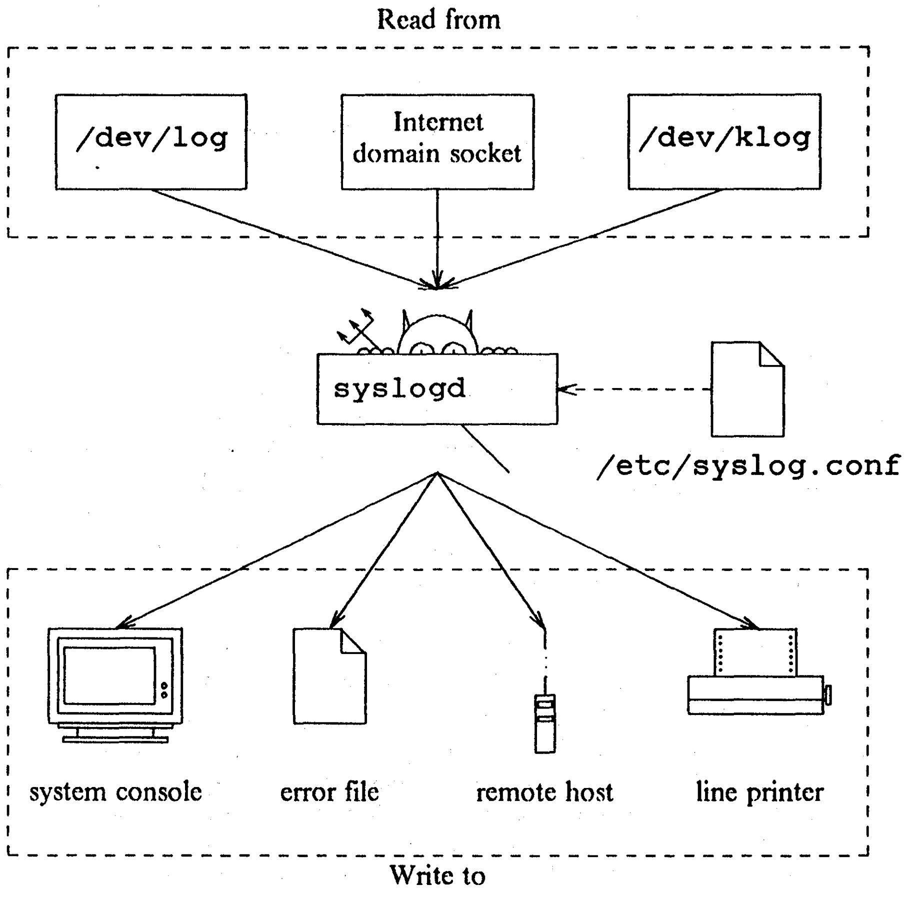


Figure 15-1: Where `syslogd` Reads From and Writes To

The `syslogd` daemon performs a `sync` operation each time the system accesses it, so that logged information is immediately written out to the disk. Each *message* managed by the system message logging facility is just one line. It may begin with a priority code enclosed in angle brackets, indicating the level of severity of the reported condition.

3. The Format of /etc/syslog.conf

The disposition of all system diagnostic messages received by the `syslogd` daemon is controlled by the entries in the file `/etc/syslog.conf`. This section describes these entries, the actions that result from them, and the use of comments and blank lines. It also provides a sample `syslog.conf` file.

3.1. Selectors and Actions

Each entry in `syslog.conf` occupies a single line, which is taken to consist of a *selector* and an *action* (which are separated from one another by one or more tab characters). Thus the general format of a line in `/etc/syslog.conf` looks like this:

selector *action*

Format 15-1: `/etc/syslog.conf`

A sample entry from `/etc/syslog.conf` might look like this:

```
    kern.err   /dev/console
```

This line indicates that whenever an error message is received from the kernel, that message is written onto the system console terminal, `/dev/console`.

A *selector* actually incorporates two pieces of information, a *facility* and a *level*. The facility indicator is an abbreviation of where the message came from, and the level indicator specifies the severity of the message. As illustrated, the facility indicator is separated from the level indicator by a period.

The currently available *facility* values are listed in the following table:

Facility	Description
`kern`	The facility associated with messages generated by the kernel (such as a `panic` or `double panic`). Such messages cannot be generated by any user-level process.
`user`	The facility associated with messages generated by ordinary user processes. This is the default facility specifier and is used if no facility is specified.
`mail`	The facility associated with messages generated by the electronic mail system.
`daemon`	The facility associated with messages generated by system-level daemon processes, such as `ftpd`, `routed`, and `syslogd` itself.
`auth`	The facility associated with messages generated by the permissions authorization system, including commands such as `login`, `su`, and `getty`.
`lpr`	The facility associated with messages generated by the line printer spooling system, such as `lpr`, `lpc`, `lpd` and similar commands.
`local0`	A facility reserved for local use. In all, there are eight local facilities (`local0` through `local7`) that may be interpreted according to local needs.
`mark`	A special-purpose facility designed to allow the system message logging daemon to periodically "mark" a log file or console screen. If enabled, a `mark` facility sends a message every 20 minutes. The interval may be altered by providing an argument to the `syslogd` at the time it is launched. For example, the command line: `/etc/syslogd -m10` will change the mark interval to 10 minutes.

Table 15-1: Error Message Facility Indicators

The available values for the priority *level* are listed in the following table:

Level	Description
`emerg`	The priority of an emergency or panic condition. Usually, a message with this priority is broadcast to all users and logged to the system error file. Such a condition may very likely result in a system crash.
`alert`	The priority of a serious condition that should be corrected immediately, if possible.
`crit`	The priority of a critical condition, such as hard input/output errors on a disk drive.
`err`	The priority of errors that do not fit into any of the first three priorities.
`warning`	The priority of warning messages.
`notice`	The priority of conditions that are not error conditions but which should probably be handled specially by the system administrator.
`info`	The priority of informational messages.
`debug`	The priority of debugging messages. These messages are normally of use only while debugging a program.
`none`	No priority. This priority indicator is provided in order to turn off facilities that would otherwise be selected by the asterisk wildcard character. For example, the selector: `*.notice;mail.none` will select all facilities at priority `notice`, except for any messages originating in the mail system.

Table 15-2: Error Message Level Indicators

Note that these levels are listed in decreasing priority. For example, the level `err` is of greater priority than the level `info`. Each available facility may have any of these priority levels associated with it, except for the `mark` facility, which always signals a message of priority `info`.

3.2. Actions

When a message of a certain level is received from a certain facility, the `syslogd` daemon takes whatever action is listed on the appropriate line of the configuration file, `/etc/syslog.conf`. For example, consider the following line in the configuration file:

```
lpd.debug          /usr/adm/lpd-errs
```

This instructs `syslogd` to write whatever messages it receives from the line printer system that are of severity `debug` (or greater) into the log file, `/usr/adm/lpd-errs`. As you probably surmise, if an action is simply a file name, then the corresponding message is written onto the end of that file. Nothing prevents the file from being a terminal device such as the system console. So the following type of configuration line is quite possible:

```
kern.emerg         /dev/console
```

This instructs `syslogd` to write whatever messages it receives from the kernel that are of severity `emerg` on the system console.

In addition to writing the received message onto the end of a file, `syslogd` can be configured to send the message to one of three other places. A full list of message destinations is given in the following table:

Specification	Description
file	The message is appended to the file specified by the name *file*. The file must be specified using an absolute path name (i.e., a name that begins with a leading slash, '/').
@host	The message is forwarded across a local area network to the remote system whose name is *host*. The disposition of the message on the remote system depends of course on the configuration of `syslogd` on that host.
user	The message is written on the terminal screen of the named *user*, if that user is currently logged in. Multiple users may be specified by listing their names separated by commas.
*	The message is written on the terminal screens of all users who are currently logged into the system.

Table 15-3: Destinations for System Messages

3.3. Comments and Blank Lines

As you have no doubt already guessed by looking at the sample `/etc/syslog.conf` listed above, any line in this file that begins with the pound sign '#' is interpreted as a comment and is ignored. Similarly, all blank lines are ignored.

3.4. Sample Syslogd Entries

It is difficult to give any detailed advice concerning the disposition of diagnostic messages arising from the operation of an IBM/4.3 system, since your interest in and reactions to certain messages will be highly site-specific. Nonetheless, it will be useful to look at a sample `/etc/syslog.conf`:

```
# a sample /etc/syslog.conf

*.err                   /dev/console
kern.debug              /dev/console
auth.notice             /dev/console

kern.debug              /usr/adm/messages
daemon,auth.notice      /usr/adm/messages
*.err                   /usr/adm/messages
mail.crit          /usr/adm/messages

lpr.debug          /usr/adm/lpd-errs

mail.debug              /usr/spool/mqueue/syslog

*.alert                 operator
kern.err           operator
daemon.err              operator

*.alert                 root
```

```
    *.emerg              *

    auth.crit       @gator
```

The very first line indicates that all messages of severity `err` or greater are to be written on the system console terminal, regardless of their origin. The second and third lines also cause some other messages of lesser severity to be written there, but only when they arise from the kernel at level `debug` or greater or from the authorization system at level `notice` or greater. This disposition of messages from the authorization system can help you to help track failed login attempts and failed `su` attempts.

Since it is also useful to save messages of great severity and messages indicating possible security violations (such as failed `su` attempts), the next four lines send those messages also to the system-wide message file, which is usually `/usr/adm/messages`. Notice that the `syslogd` daemon can manage multiple message log files, as indicated by the next two entries which send all printer-related messages to the file `/usr/adm/lpd-errs` and all mail-related messages to the file `/usr/spool/mqueue/syslog`.

All the remaining lines in this sample configuration file, except the last one, illustrate how to send messages to a particular user or group of users. For example, all messages of severity `alert` (or greater), as well as all kernel and daemon messages of severity `err` (or greater) are sent to the `operator` account, if anyone is currently logged in using that name. The `operator` account is intended to be used primarily for file system backups and restores, and it is important that that person should receive notification of any abnormal activity that may affect those backups or restores.

The final line listed above shows how to route error messages over a network to a remote machine. Any authorization-related messages of severity `crit` or greater will be sent to the `syslogd` on the remote host named '`gator`'. This allows a central authority to supervise possible security violations on an entire network of workstations and servers. Similar `syslogd` configuration lines may be useful, for example, if your site maintains a central print server and you prefer all printing system messages to be sent to that server.

4. Starting Up Syslogd

The daemon `syslogd` is normally started up at system boot time. You can launch it automatically whenever you enter multi-user operation by including the following line in the file `/etc/rc` or the file `/etc/rc.local`:

```
    /etc/syslogd
```

As a rule, the message logging daemon should be started up *after* the file systems are checked and mounted, but *before* any other daemons are started up. Consequently, the line given above is usually placed into the file `/etc/rc.local`, which is run before editor files are preserved and other local and network daemons are started up.

It is possible to modify the default behavior of the message logging system by supplying a command line argument when `syslogd` is first launched. For example, you may specify that `syslogd` configure itself according to instructions located in some file other than `/etc/syslog.conf` by invoking it with the `-f` argument.

A special debugging flag, `-d`, can be provided to turn on `syslogd` debugging. This is most useful for checking the syntax of a revised configuration file. As the following command output illustrates, the first thing `syslogd -d` does is to parse the configuration file:

```
    # /etc/syslogd -d
    off & running....
    init
    cfline(*.err          /dev/console)
```

```
cfline(kern.debug     /dev/console)
cfline(auth.notice    /dev/console)
```
[lines omitted]
```
cfline(auth.crit      @gator)
```

If there are errors in the configuration file, they will be listed in the diagnostic output. For example, suppose that you forget to specify a severity level on some line, so that the first several lines in `/etc/syslog.conf` look like this:

```
*.err                /dev/console
kern                 /dev/console
```

Then, when you run `syslogd` with debugging turned on, you will see diagnostics like these:

```
# /etc/syslogd -d
off & running....
init
cfline(*.err          /dev/console)
cfline(kern           /dev/console)
syslogd: unknown priority name "": no such file or directory
```

(In fact you will see this message repeated several times, owing to the fact that the first line was successfully parsed and acted upon.) Such messages are an indication that you need to edit the configuration file and correct the mistake.

5. Stopping Syslogd

Once started, the system message daemon `syslogd` will continue in operation unless stopped manually by the system administrator (or, in rare circumstances, by abnormal system activity). To halt system message logging manually, you should send a terminate signal to the daemon using the `kill` command. Recall that the process id number of `syslogd` is automatically saved in the file `/etc/syslog.pid` each time the error daemon is launched (usually from within `/etc/rc`). You can therefore be assured of killing the appropriate process by executing the command line:

```
# kill `cat /etc/syslog.pid`
```

Sometimes, the file `/etc/syslog.pid` cannot be created when `syslogd` is launched; in such cases, you will need to find the process id number of the `syslogd` daemon by using the `ps` command:

```
# ps -ax | grep syslogd
```

You should get two lines of output; for example:

```
  45 ?  S      0:02 /etc/syslogd
1977 p0 S      0:00 grep syslogd
```

Select the process id number of the daemon itself and execute the `kill` command using it as an argument. In this case, the pid is 45, so you would type:

```
# kill 45
```

6. Sending Messages from the Command Line

The command `logger` allows the system administrator to enter messages into the system log files manually from the command line. For example, you may run the command:

```
# logger System time and date reset
```

to record that the system time and date were just reset. The message specified on the command line will be entered into the appropriate log file exactly as if it had originated as a message from a program.

If no file is specified, the default system message file (`/usr/adm/messages`) will be used. You can send the message to another file by providing the `-f` command line argument. For example, the command:

```
# logger -f /usr/spool/adm/syslog Games turned off.
```

will append the message to the file specified.

The `logger` command also assumes a default facility and level for the message provided, namely '`user.notice`'. If you want to send the message to a different selector, you may specify it on the command line. For example, the command:

```
# logger -p mail.info Truncated /usr/spool/mail/smith
```

will perform whatever action `syslogd` is configured to perform for informational messages from the `mail` system. See the manual page `logger`(1) for a complete summary of the available options for the `logger` command.

7. Checking Messages

The system administrator should monitor the error log files regularly in order to determine whether the system is operating as expected, and if not, what corrective action to take. To see the end of a log file, execute the command:

```
tail -r /usr/adm/messages
```

The `-r` flag will cause the output to be sorted in reverse order, so that the first line seen is actually the last line that was appended to the file. You should check the configuration file `/etc/syslog.conf` to see what other log files are in use by the system message logging daemon.

8. Message Handling Tips

- If you plan to make major revisions to the message daemon configuration file, `/etc/syslog.conf`, or if you are inexperienced with `syslogd` and its configuration file, then you should invoke `syslogd` with the debugging flag (`-d`) in order to ensure that the configuration file is correctly set up. To be extra safe, you should first make a copy of the original, edit the copy, and then run `syslogd` with the debugging flag on the revised copy:

  ```
  # cp /etc/syslog.conf /tmp/syslog.conf
  # vi /tmp/syslog.conf
      [make changes to the new copy]
  # /etc/syslogd -d -f/tmp/syslog.conf
  ```

  Note that no space is allowed after the `-f` flag and before the file name. If `syslogd` successfully parses the new file, then you can overwrite the original copy with the revised one and then start up `syslogd`:

  ```
  # mv /tmp/syslog.conf /etc/syslog.conf
  # /etc/syslogd
  ```

  Since there should be at most one instance of `syslogd` running on your system at one time, you should perform these tests while no `syslogd` is running.

- You can edit the error message daemon configuration file, `/etc/syslog.conf`, even while the system is operating in multi-user mode and the message daemon `syslogd` is running. To force `syslogd` to reread its configuration file, send it a hangup signal, as follows:

```
# kill -HUP `cat /etc/syslog.pid`
```

Thus, you can change the disposition of system messages "on the fly", without needing to reboot the system or bring it down to single-user state.

This page intentionally left blank.

CHAPTER 16

Executing Periodic Commands

1. Introduction

One of a system administrator's major concerns is to reduce as much as possible the number of actions that must be performed manually to keep the system running smoothly. To help achieve this goal, IBM/4.3 provides a simple and flexible way to run a command or shell script automatically at specified times. The `cron` daemon and its configuration file `crontab` are the tools you will use to establish periodic command execution. One of the main uses of these tools is to perform a variety of system administration tasks that need to be accomplished at regular intervals and that require minimal operator assistance. For example, `cron` can be used to help maintain an uncluttered file system by removing files that have not been accessed within a certain amount of time; this is especially useful for finding and removing temporary files, outdated log files, and the like. The `cron` daemon is also useful for starting various communications processes such as `uucp`. This chapter explains how to set up the `cron` system to handle such tasks automatically and efficiently. For information on establishing uucp connections using `cron`, see the section on administration in "Installation and Operation of UUCP" in the *UNIX System Manager's Manual* (Section 9).

2. How Cron Operates

`Cron` is a clock daemon that is usually started up at multi-user initialization time by placing the following line in the file `/etc/rc`:

```
/etc/cron
```

If this line is not present in your `/etc/rc` file or in `/etc/rc.local`, the `cron` daemon can be launched by typing that line to the shell. The basic operation of `cron` is to read the file `/usr/lib/crontab` (and, if it exists, the file `/usr/lib/crontab.local`) once every minute, on the minute, to see if there are commands contained there that are scheduled to be run. If there are such commands, `cron` will run them. The logical position of `cron` is illustrated in the following diagram:

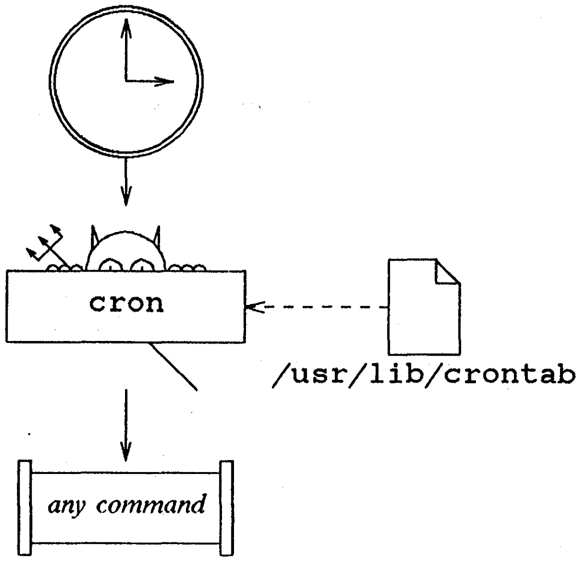


Figure 16-1: The Position of `cron`

If the specified command does not itself redirects its output, `cron` will place any output resulting from the execution of the command into the file `/usr/adm/cronlog`. `cron` will also rescan the `crontab` file(s) each time they are altered. In addition, unless terminated manually by the system administrator or by abnormal system activity, `cron` never exits. There should therefore be at most one instance of the `cron` daemon running on a system at any given time.

3. The Format of Crontab

The format of the files `/usr/lib/crontab` and `/usr/lib/crontab.local` is straightforward. The `crontab` files consist of lines, each of which has seven fields (which are separated from one another by any number of spaces and/or tabs). The format is as follows:

minute	*hour*	*day-of-month*	*month*	*day-of-week*	*user*	*command*
1	2	3	4	5	6	7
		When			Who	What

Format 16-1: `/usr/lib/crontab`

As indicated, the seven fields may be thought of as grouped into three parts, a time specification (indicating *when* the specified action is to occur), a user specification (indicating *who* is to run the specified action), and a command specification (indicating *what* action is to occur). The appropriate values for each of these parts are discussed in the following three subsections.

3.1. Time Specification

The first five fields jointly indicate the *time* when the command specified in the last field will be run. Each of these five fields may be filled with a number from the appropriate legal range for the field, a dash-separated range of values, a comma-separated list of values, or an asterisk (*) indicating that all legal values are selected. The legal ranges are as follows:

minute	00 to 59
hour	00 to 23
day-of-month	00 to 31
month	01 to 12
day-of-week	01 to 07 (where 01 denotes Monday)

Table 16-1: Legal `crontab` Time Specifications

If the specified value is a dash-separated range, then `cron` will run the appropriate command for each value in the range. So, for example, a line whose *day-of-week* field was

```
1-5
```

would be selected for each working day, i.e. Monday through Friday. If the specified value is a comma-separated list, then `cron` will run the command for each value in the list. A line whose *day-of-week* field was

```
1,3,5
```

would be run every Monday, Wednesday, and Friday, at the time selected by the first two fields. You should notice, however, that this will happen only if the *day-of-month* field is also selected in the `crontab` entry (perhaps because it contains an asterisk). The `cron` daemon will run a command listed in the `crontab` file just in case *all* five time fields are satisfied.

3.2. User Specification

The format of `crontab` has recently been altered to include an indication of *who* the command is to be run by. More precisely, this field contains the login name of the user whose user identification number is to be adopted as the effective uid of the specified process. For many commands the *user* field will contain the name '`root`', since `root` privileges are needed to read or write certain files or to run certain commands. But some of the commands that are typically run by `cron` do not need `root`'s absolute privileges and it is dangerous to run them under the `root` uid. For such commands, the system administrator can specify a login name with lesser privileges.

For example, processes that are run periodically for the uucp system are usually run with the `uucp` account listed as the user. This allows them to access (and possibly remove) files and directories in `uucp` spool directories. Also, some commands that need minimal access privileges have the account `nobody` in the *user* field. The `nobody` account was added to the system precisely to provide a less dangerous uid for `crontab` entries that do not require greater privileges.

3.3. Command Specification

The final field in a `crontab` entry is the specification of what action is to be taken at the time specified in the first five fields. The *command* field of a `crontab` entry may contain any valid command line. In particular, it may contain input/output redirection characters, pipes, and multiple commands separated by semicolons.

3.4. Sample Crontab Entries

The following is a sample `crontab` entry:

```
00   *  *  *  *  root   date > /dev/console
```

This specifies that the date command is to be run every hour of every day of every week in the month, at the beginning of each hour (i.e., minute 00). The output here is redirected to the system console, so if there is no other activity on the console, a list of times will accumulate on the console screen.

The sixth field in a crontab entry specifies the user whose uid and permissions the command specified in the seventh field should inherit. In this example, the date command will be run with the uid and permissions of the superuser. (This allows the command to actually write on the system console, which may not be possible for other users.)

Here is another typical crontab entry:

```
00,15,30,45 *   *   *   *   root   /usr/lib/atrun
```

When cron reads a crontab file containing this line, it will execute the atrun command every 15 minutes. This command will be run every hour of every day of every month, since the second, third, fourth, and fifth time fields each contain an asterisk. The atrun command is invoked in order to process user-specified commands or scripts that were submitted with the at command. Remember that even though a user may submit a command to be run at some later time with the at command, that command will not actually be run unless the atrun command is executed periodically. This task is typically assigned to cron. Notice also that the granularity of the at command is dependent upon the corresponding crontab entry. For instance, a command submitted to run at exactly 12:20 a.m. would not be run until 12:30 a.m., since the crontab entry listed above has a granularity of 15 minutes. To achieve finer control over times specified with the at command, you could use a crontab entry like this:

```
00,05,10,15,20,25,30,35,40,45,50,55  *   *   *   *   root   /usr/lib/atrun
```

This entry will execute the atrun command every five minutes, so that a command submitted to run at 12:20 a.m. will now be executed at exactly 12:20 a.m. (or as close to it as possible, since it takes cron a few seconds to launch the command). The system administrator must balance the greater precision offered to users of at with the increased system load incurred by running atrun more often. On most systems, a granularity of 15 minutes is perfectly acceptable.

Finally, the following crontab entry will send a holiday greeting to the terminal screen of each user who is logged in at the appropriate time:

```
00  00  01  01  *  root   echo "Happy New Year" | /bin/wall
```

Here, cron has been instructed to write to each logged-in user at midnight (12:00 a.m.) on January 1st of each year. Notice that the fifth field, the *day-of-week* field, contains an asterisk, so that the command will be executed no matter which day of the week New Years Day falls on.

It is possible to send a multi-line message to all users by inserting a percent sign, '%', at the appropriate spot in the message. For example, you might expand the previous greeting, as follows:

```
00  00  01  01  *  root   echo "Happy New Year%To All" | /bin/wall
```

The percent sign will be translated by cron into a newline character before the string is given as input to wall, so that the wall command receives a two-line message as input. You can embed as many percent signs as you like in order to create a message of the desired number of lines. Currently it is not possible to have the percent sign appear in the resulting message.

Although you can specify multi-line input to a command using the percent character, it is not possible to embed multi-line commands into the *command* field of the crontab file. If a multi-line command is desired, or if multiple commands must be run to accomplish some administrative task, the relevant lines should be put into a shell script and cron should be instructed to execute that shell script. This situation is illustrated below.

3.5. Comments

In the configuration files read by `cron` (i.e., `/usr/lib/crontab` and `/usr/lib/crontab.local`), any line that begins with a pound sign (#) is treated as a comment and is ignored by `cron` when it reads that file. You may also isolate a group of related `crontab` entries by surrounding them and any applicable comment lines with vertical white space. Blank lines can be used for this purpose.

4. Cleaning /tmp and /usr/tmp

It is fairly typical for the system directories that hold temporary files, `/tmp` and `/usr/tmp`, to be cleared of most files and subdirectories at system multi-user initialization time. (Check the start-up file `/etc/rc` to see whether this happens on your own system.) Since, however, your IBM/4.3 system may continue running for days, weeks, or even months at a time, it is good practice to clear out these directories periodically. This is partly because these directories can easily become filled with scratch files or buffer files from processes that die unexpectedly or that are killed by the user. Another reason is that users sometimes copy files into the temporary directories and then forget to remove them when finished. Keeping these directories from accumulating too much junk is a perfect task for `cron`.

One way to manage the temporary file space with `cron` is to insert into `crontab` an entry like this one:

```
00  02  *  *  *   root   find /tmp -atime +3 -exec rm -f {} \;
```

At 2:00 a.m. on each day in the month, `cron` will search the directory `/tmp` for files that have not been accessed in the last three days. If it finds any such files, it will remove them.

While this command removes all stale files below the `/tmp` directory, it does nothing to any subdirectories (except empty them of all files). We would also like `cron` to remove these empty directories, if there are any. To accomplish this, we could add another entry to `crontab`:

```
10  02  *  *  *   root   find /tmp ! -name . ! -name lost+found -type d \
                                -mtime +1 -exec rmdir {} \;
```

This command will look for all directories under `/tmp` that have not been accessed in a day and that are not named either '`.`' or '`lost+found`'; then, if it finds any, it will remove them. (Note that we instructed `cron` to run this command ten minutes after running the previous command. This was so that we could be sure that the previous command was completed before the second began.)

A much better solution to the general problem of managing temporary file space is to collect all of these related procedures into a shell script and to have `cron` execute that script, instead of putting each such command into `crontab`. For example, the command field of these two previous `crontab` entries, as well as related commands to be run on a daily basis (such as similar commands for the directory `/usr/tmp`), might be put into a shell script called `daily`. If such a script is stored in the directory `/usr/adm`, then the following single `crontab` entry will suffice:

```
00  02  *  *  *   root   /bin/sh /usr/adm/daily 2>&1 | mail root
```

In this example, any output and error messages resulting from the running of the script will be mailed to the superuser. This strategy of collecting similar commands into shell scripts minimizes the number of entries in `crontab` and makes it much easier to read and maintain.

5. Removing Other Old Files

There are two tools available to you to help in maintaining files that can become cumbersome. Both can be invoked by `cron`:

- The `uuclean` program purges files it knows are no longer needed from the uucp spool directory and is typically run by `cron` on a daily basis. See Section 5 in Chapter 17 for more information.

- The `trimlib` script helps in keeping USENET log files to a reasonable size and is typically run by `cron` on a weekly basis.

6. Turning off Games during Prime Time

Computer games are no doubt sometimes a relaxing pastime, but most installations frown upon games-playing during the prime working hours. You can put the abilities of `cron` to good use by having it run the following simple script sometime in the morning:

```
#! /bin/sh
# turn_off
chmod go-x /usr/games/*
logger Games turned off
```

And games may be turned back on with the following script:

```
#! /bin/sh
# turn_on
chmod go+x /usr/games/*
logger Games turned on
```

To activate this scripts at the appropriate time, you can put the following three lines into the file `/usr/lib/crontab.local`:

```
# turn games off/on at appropriate times
00 08 * * 1-5     root  /usr/games/turn_off
00 18 * * 1-5     root  /usr/games/turn_on
```

Then the games will be turned off at 8 a.m. each weekday morning and back on again at 6 p.m.

7. Calendar

The `calendar`(1) program is a reminder service for IBM/4.3 system users. It works by consulting the file `calendar` in the current directory and printing lines that contain today's date or tomorrow's date. By including the command `calendar -` in a `crontab` file entry, you can specify that `calendar` check each user's calendar file, then notify users of the results through `mail`(1). See `calendar`(1) for more information.

8. Accounting

You can include entries in your `crontab` file that request accounting information concerning connect time and process resource. IBM/4.3 stores information related to connect time accounting in `/usr/adm/wtmp`; you can use the `ac`(8) program to summarize this information. IBM/4.3 stores information regarding process time accounting in `/usr/adm/acct`, once this file is enabled by `accton`(8). You can use the `sa`(8) program to analyze and summarize this information.

You might want to implement procedures based on information provided by these commands for accounting tasks such as charging for computing time. A convenient and efficient way to do this is to place these commands in your `crontab` file, so that they are executed every day at the time you specify.

9. Creating User-Specific Crontab Entries

Unlike the versions of `cron` supplied with newer releases of System V (2.0 or later), the IBM/4.3 `cron` does not provide ordinary users with the ability to create and edit their own personal `crontab` files (thus allowing them to perform periodic actions automatically in the same way that the system administrator can). Nonetheless, it is simple to include entries in the system-wide crontab or crontab.local to provide this service. For example, if the user nat needs to execute a certain set of commands periodically, you can place an entry like the following into one of the cron configuration files, probably crontab.local:

```
00  03  *  *  *  nat   /bin/csh -f -c ~nat/cron.rc
```

In this case, cron will execute whatever commands exist in the file cron.rc in nat's home directory. No doubt the time and periodicity listed in the crontab entry must be chosen in consultation with the user. In the example listed, the script will be run each day at 3:00 a.m. It is then up to the user nat to maintain the script of C shell commands in the file cron.rc.

Before actually inserting such lines into the cron configuration files, however, you should determine whether in fact the user needs to use cron. (After all, imagine trying to maintain crontab entries for the dozens or even hundreds of different users in your computing environment!) It is quite possible that the users's needs can be accomplished more simply, without using cron directly. For example, suppose that a user needs to maintain a list of the files that exist in the home directory at a specified time each day. You *could* provide this service by inserting the following command into the crontab.local file:

```
00  03  *  *  *  nat   ls -l ~nat > ~nat/LIST
```

Alternatively, you could insert the previous line that runs the script cron.rc in nat's home directory, leaving it up to that user to insert the proper commands into it. But instead of using cron, the user can use the at command, in the following clever way. First, the user must create an at script that contains the desired commands, as well as a command that reinvokes at at the same time the following day. Here is an example of what this user might put into the at script:

```
# at.rc
# imitate cron actions with an at script
ls -l ~ > LIST
sleep 60
at 0300 at.rc
```

To start the whole process, the user should give this command:

```
% at 0300 at.rc
```

Then, at 3:00 in the morning, the home directory will be listed, the script will sleep for a minute (just to be safe), and then at will be reinvoked for the following morning with the same script. In this way, a recursive application of at can provide exactly the functionality that cron would provide, at much less cost to the system administrator. If ordinary users request that you create crontab entries for them, you should recommend that they try to accomplish the same effects in this somewhat simpler way.

This page intentionally left blank.

CHAPTER 17

Controlling Log Files

1. Introduction

There are a number of files located at various spots in the IBM/4.3 file system that are used to maintain a record of system activities such as user logins, command executions, news transmissions, system errors, failed `root` login attempts, and similar events of concern to the system administrator. Collectively these files are called "log files" because they contain an ongoing log of system activities. Log files can be extremely useful in helping track down system problems or maintain a secure system, but they may grow without bound unless they are periodically truncated. One good example of this is the file `/usr/adm/wtmp`, which collects user connect-time information. As long as users continue to log in and out of your system, this file will grow larger and larger. The processes that append records to `/usr/adm/wtmp` (`init` and `login`) do not check to see if the size of the file has exceeded some limit. Accordingly, it is the duty of the system administrator to collect the information needed from these files, summarize it, and then truncate the files so that they can continue to grow and perform their logging functions. This chapter explains some of the techniques commonly used to administer the many log files in the IBM/4.3 system.

Some log files are inspected, summarized, and truncated by special system utilities dedicated to that task, so that all the system administrator needs to do is make sure that those utilities get run at the appropriate intervals. On the other hand, most of the log files have no special utility programs associated with them. Accordingly, the main part of this chapter is devoted to developing and explaining several useful shell scripts that will help you maintain system log files. In either case, whether you use special utilities or the scripts presented here, the scheduling of the log file processing is usually handled by the clock daemon, `cron`, as illustrated below.

Exactly what you do with the information that is stored in the system's log files is highly site-dependent. If your system is well-administered and you experience few if any abnormal occurrences, then you may never need to inspect those files and you can pretty much forget about them once you have instituted a suitable log file rotation scheme. More probably, however, you will want to collect, summarize, and use the information in the log files in some important way. In an environment where users must pay for computing services, for instance, you might use the connect-time information stored in `/usr/adm/wtmp` to determine part of the account charges for the user community. Or if you have experienced a system break-in, the process accounting log files may help you determine when and how the break-in occurred.

2. Log File Rotation: Version 1

The simplest way to make certain that log files do not get too large is just to truncate them periodically. This may be accomplished using the `cron` daemon, by inserting a line into the file `/usr/lib/crontab`. For example, to truncate the log file containing a record of user logins, you might put in the following line:

```
    30 02  *  *  *   root   cp /dev/null /usr/adm/wtmp
```

At 2:30 a.m. of each morning, the file will be truncated to 0 bytes by copying `/dev/null` onto it. Since it is run each day, this command ensures that `/usr/adm/wtmp` never gets too large.

This scheme, however, has several clear disadvantages. Foremost among them is the fact that potentially useful information is simply thrown away rather than archived, summarized, or otherwise maintained on the system. If you want to use the connect-time information to charge account fees, you will need to collect the appropriate information before it is removed from the system.

As you can see, merely truncating a log file in this way is rarely the correct way to manage the information contained in it. Accordingly, most sites prefer to institute a procedure that combines *rotation* with *truncation.* Under this arrangement, the information in a current log file is first saved in an intermediate file. There are usually several such intermediate files, only the oldest of which is actually thrown away. This procedure can be illustrated by the following simple shell script:

```
#! /bin/sh
# wtmp_swap: rotate connect-time accounting files
echo "Rotating wtmp files"
cd /usr/adm
mv wtmp.2 wtmp.3
mv wtmp.1 wtmp.2
mv wtmp.0 wtmp.1
mv wtmp   wtmp.0
cp /dev/null  wtmp
chmod 644 wtmp
```

As you can see, existing intermediate files are rotated before the current `wtmp` file is placed into the first intermediate file; then `wtmp` is recreated, with a size of zero bytes. The sequence of actions can be illustrated as follows:

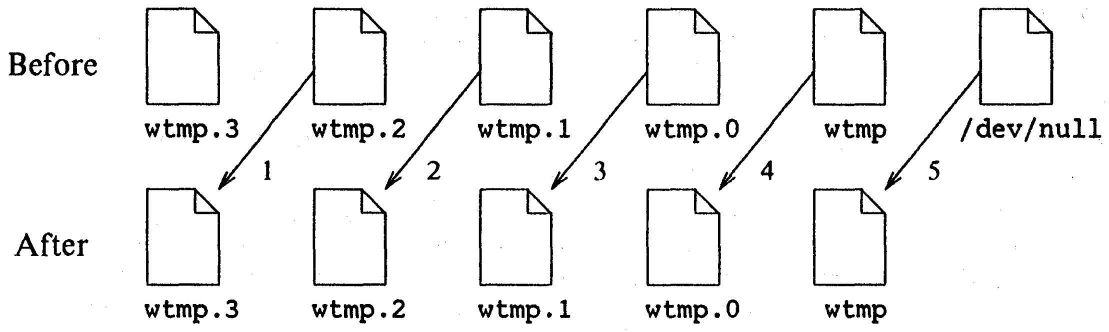


Figure 17-1: Log File Rotation Sequence

By rotating the log file `/usr/adm/wtmp` in this way, the system administrator can ensure that a reasonable amount of connect-time information is kept available on the system, while removing the danger that that file will eventually grow too large for the available storage space.

This log file rotation and truncation script is designed to be run periodically by `cron`. So you will want to put a line like the following into your `crontab` file:

```
00 02  *   *   *   root   /usr/adm/wtmp_swap > /dev/console
```

Notice that the output from the script is redirected to the system console, so that the system administrator will know that the script has run.

3. Log File Rotation: Version 2

The log file rotation and truncation script presented in the previous section provides the basic functionality needed to maintain system log files at an acceptable size while not throwing away too much information at any one time. The approach illustrated by the script `wtmp_swap`, however, has several disadvantages that can easily be alleviated. You will notice that that script as currently written rotates and truncates only a single file, `/usr/adm/wtmp`. To apply that method to other log files on the system, you would need to create a similar script for each log file, probably by copying that script and making suitable changes. If you later wanted to add some feature to the log file processing routine (for example, backing up the oldest intermediate file onto a floppy disk or streaming tape instead of just removing it), you would need to find, edit, and modify every copy of the script. Since the various copies may be scattered about in different directories, that is not a trivial undertaking.

It is a good idea therefore to encapsulate the rotation and truncation sequence into a script that can be applied to several different log files. It is also good practice to add some diagnostic output to the script, so the system administrator will know if it terminates prematurely. With these desires in mind, then, consider the following improved version of `wtmp_swap`:

```
#! /bin/sh
# log_swap: rotate a log file through intermediary files

ERROR='eval echo >&2' # send error message to stderr
USAGE='Usage: $0 logfile number'
trap '$ERROR "$0: exiting prematurely"; exit 1' 1 2 3 15

if [ $# != 2 ]; then
     $ERROR $USAGE
     exit 1
fi

LOG=$1
NUM=$2

i=`expr $NUM - 1`
while [ $i -gt 0 ]
do
     j=`expr $i - 1`
     if [ -f $LOG.$j ]; then
          mv $LOG.$j $LOG.$i
     fi
i=`expr $i - 1`
done

if [ -f $LOG ]; then
     cp $LOG $LOG.0
     cp /dev/null $LOG
fi
```

This script requires two arguments, namely the log file to be rotated and the number of intermediary files to use. If it is not invoked with exactly two arguments, it prints a diagnostic message and then exits; otherwise it proceeds precisely as did the previous script. To activate log file rotation, you must once again add some lines to the `cron` configuration file. For example:

```
00 02  *  *  *   root   /usr/local/log_swap /usr/adm/wtmp 6
01 02  *  *  *   root   /usr/local/log_swap /usr/lib/news/log 5
```

```
02 02  *  *  *   root    /usr/local/log_swap /usr/adm/messages 6
10 03  1  *  *           /usr/local/log_swap /usr/spool/uucp/LOGFILE 5
```

4. Uucp

During normal operation, uucp generates a number of small files, which it places in the directories beneath `/usr/spool/uucp`. If left unchecked, these files can cause space problems; uucp provides the following tools to aid in preventing this:

- The `uuclean`(8) program purges files it knows are no longer needed from the uucp spool directory. It works by looking in the spool directory for files with the specified prefix and deleting all files that are older than the specified number of hours. Typically `uuclean` is run from `cron` on a daily basis. It has the following arguments:

  `/usr/lib/uucp/uuclean` [`-m`] [`-n`*hours*] [`-p`*pre*] [`-d`*subdirectory*] [`-x`*num*]

  The `-p` options specifies the prefix `uuclean` should look for when searching for files; You can specify up to ten prefixes. The `-n` option specifies the age of files that should be deleted, in hours, provided they have the specified prefix. The default age is 72 hours. The `-m` specifies that `uuclean` is to send mail to the owner of the file upon the file's deletion, and the `-d` option specifies that `uuclean` should work in only the named subdirectory. The `-x` option specifies the level of debugging output you want.
- The `uulog` program scans session log files and helps you to maintain them by printing a summary log of uucp and uux transactions. Uucp gathers information from uucp and uux log files (which reside in `/usr/spool/uucp` under names beginning with LOG), then creates a file called LOGFILE in `/usr/spool/uucp/LOGFILE` that you can examine directly or through `uulog`, which resides in `/usr/bin`. When using `uulog`, you can specify that it print information about work done for a specific user, or about the system that requested the work. `uulog` then prints the time, date, and status for each request.

USENET also creates files that must be monitored. There are two groups of files that should be maintained:

- History files can be cleaned up with the `expire` program. This program deletes lines from the history file that relate to articles that have been removed.
- The log file can be maintained with the `/misc/trimlib` script, which you can install in `LIB/trimlib`. This script is typically invoked weekly by `cron`.

5. Other Files that May Grow without Bound

Aside from the log files discussed above, it is possible for other files in the system to grow without bound for one reason or another. For example, if a particular user never reads any electronic mail, then that user's system mailbox will get larger and larger as new mail keeps arriving. If the quota subsystem is not installed and running, or if quotas have not been established on the `/usr` file system, then the mailbox may eventually get very large. Similarly, it is simple to write a program or shell script that creates an increasingly large output file. It is even possible to cause a text-processing program like `troff` to go into an endless loop, thereby creating a file that grows until a user's disk space quota is exceeded or until the remaining space in the file system is consumed.

These types of large files are not log files, so the techniques presented above will not apply directly to such large files. Instead, it may be useful to run the following script periodically (perhaps once a week) to search for very large files and notify the system administrator:

```
#! /bin/sh
```

```
# find_big

BIG=`expr 1000000 / 512`
find / -size $BIG -type f -exec ls -l {}\; |\
              /usr/ucb/mail -s "large files" root
```

In this script, a file is considered to be too big when it is larger than a million bytes (expressed in 512-byte "blocks"). You may of course want to alter that constant to suit your local practices. Many applications such as image-processing and large data bases will routinely create and maintain files much larger than that.

The following `crontab` entry will activate the script once a week at 2:30 in the morning:

```
30 2 * * 0 root   /usr/local/find_big
```

This page intentionally left blank.

CHAPTER 18

Implementing Security

1. Introduction

Ensuring the integrity of the files and data stored on the IBM/4.3 system (and hence the smooth operation of the system) is a task that involves much more than regular disk backups and file system checking. A system administrator must maintain a high level of security in order to prevent malicious or careless users from damaging or destroying important system programs and files, or from looking at sensitive user-generated files. This chapter discusses some of the security issues that should be addressed by each IBM/4.3 system administrator. It pinpoints the main areas in which such systems may be insecure and suggests preventive action to avoid break-ins or sabotage.

Careless or naive users are generally not much of a security threat to a well-administered system, although in rare circumstances such a user can crash the system by running too many processes at once, thereby exhausting the system's swap space. Typically, the main threat to a system is generated not by careless users, but by malicious ones (hereafter referred to as the "bad guys") who generally know quite a bit about the operation of the system and how to circumvent many of its protection features. It is more difficult, indeed sometimes impossible, to prevent a bad guy from gaining access to your system, especially if you have publicly-available terminals or dial-in modems attached to the system. By following the security measures discussed in this chapter, however, you can protect the system's most obvious weak points and attempt to minimize such break-ins.

How actively the system administrator must pursue security loopholes and monitor user activity depends on the value of the data stored on the system and the importance of having a working system at all times. The level of security that must be maintained, however, is always a trade-off between the cost of protection (such as additional hardware, inconvenience to users, or time invested by a system administrator) and the value of the data being protected. The measures discussed in this chapter are designed to ensure a reasonable level of security and should be followed by *all* administrators of the IBM/4.3 system; additional security measures may be necessary to accommodate your local needs.

2. Overview of IBM/4.3 Security Mechanisms

The IBM/4.3 system provides several built-in security mechanisms and it is consequently relatively difficult to break into the system if it is properly administered. In previous chapters, you have become acquainted with a number of the system's features that help ensure the security of the system and its files. These are:

- *Login name and password prompting.* Before being granted access to the system, a person must supply a valid login name and password. This mechanism is designed to prevent unauthorized persons from using system resources. The best way to keep bad guys from damaging important system files or from inspecting the data on your system is to keep them from gaining access to the system, and the password mechanism tries to do precisely that.
- *File access permissions.* In order to be able to read and/or modify a file, or move into a directory, a user must have the proper access permissions. This allows a user to prevent other users (or other users not in the same group) from looking at, copying, or otherwise

manipulating personal files. In addition, the superuser can help assure the integrity of important system files by properly setting their protection bits.

- *Disk quotas.* A system can be severely disabled if it runs out of free file space, as can happen quite easily if users are allowed to create arbitrarily large files or large numbers of files. If the disk quota subsystem is installed and running, however, users will be unable to exceed their hard quotas and consume excessive amounts of disk space, even though free space remains in the file system.
- *System accounts with non-superuser privileges.* The IBM/4.3 system provides non-superuser accounts for use in performing several important administrative functions. For example, the `operator` account has read permission on all file systems, so that a user logged into that account may perform file system backups. This allows backups to be performed at non-peak hours by relatively inexperienced personnel without providing them full `root` privileges.
- *Process accounting.* If process accounting is enabled, then the system administrator will be able to determine who in the system is doing what. This allows the administrator to balance the use of machine resources among competing users and to help troubleshoot in the case of a lapse in security.
- *Setuid programs.* The mechanism of `setuid` programs (where a program can run with the effective user identification number of the owner of the program) allows normal users to perform functions that would otherwise require the assistance of a system administrator, without according those users the privileges given to system administrators.

There are several additional features of the IBM/4.3 system that contribute to its secure operation but which have not been discussed previously:

- *Data encryption.* If a user decides that the normal file access protections are insufficient to ensure that certain sensitive files will not be read by anyone else (even by the superuser), the user can encrypt those files. Encryption changes the file into a form that is unreadable to anyone who cannot first decrypt the file by providing the encryption key. In effect, the data encryption mechanisms are an extension of the password security mechanisms to individual files.
- *Mail encryption.* There is a mail encryption facility that allows users to encrypt messages sent through the electronic mail system. By using this facility, users can exchange potentially sensitive information without fear that a clear text version of it may be intercepted by other users or by the superuser.
- *Superuser password protections.* Several system commands are designed to restrict access to superuser privileges, even by users who for some reason may know the superuser password. For example, the `su` command allows only those users who are members of the group `wheel` to become `root`. Anyone else will be denied permission to assume the `root` uid, even though they may be able to supply the correct `root` password. As described below, it is also possible to prevent logins using the `root` login name over selected terminal lines. In a situation where very tight security is required, `root` logins can be disabled on all terminals except the system console. As a result, superuser privileges can be denied to anyone unable to access the console, even though the superuser password has become compromised.
- *Password aging.* It is a good idea for users to change their passwords often. Although in the IBM/4.3 system there is no automatic way to force a user to change the account password periodically, it is relatively simple to implement such a scheme using shell scripts. One possibility is presented below.

Not surprisingly, many of the principal security loopholes of an IBM/4.3 system arise from misuse of these mechanisms, either because they were incorrectly implemented or because they were not implemented at all. The large part of this chapter discusses the correct implementation and maintenance of these security mechanisms, as well as some consequences that may arise from incorrect administration.

3. Physical Security

The first level of security, and no doubt the most obvious, is to ensure that the physical hardware not be exposed to damage, unauthorized use, or theft. This may involve placing the equipment in a secure area that is accessible only by key or magnetic card; it may also involve physically securing the equipment with locks and cables. Every effort must be made to keep unauthorized persons away from the equipment. This is particularly important with the system console, since there is usually a superuser login session on it. Even when there is no superuser login on the console, a bad guy can usually obtain `root` privileges simply by powering down a machine and rebooting it in single-user mode, if the bad guy has physical access to the machine. If your workstation has a key to disable the keyboard, you should use it whenever you step away from the machine.

In addition to maintaining the physical security of the computing equipment, the administrator must maintain adequate physical security for backup tapes and disks. A bad guy who has access to current dump tapes can circumvent the password security and file access protections provided by the IBM/4.3 system simply by reading those tapes onto another system on which he has superuser privileges. Therefore, the system administrator should lock up all secondary media (streaming tapes and diskettes, for example) when they are not in use. It is also *highly* recommended that at least one complete backup copy of all file systems be stored off-site, so that the data and files can be resurrected on another system in the event that fire, flood, or other natural disaster destroys the machine and physical surroundings.

Finally, the system administrator should remind users at publicly-available terminals to log out whenever they leave the terminal unattended for any length of time. As you shall soon see, it is quite easy for a bad guy to install some short but effective scripts that can be used to undermine a system's security, if the bad guy can gain access to the system. It is imperative therefore that users either log out or disable use of the terminal if some errand takes them away from the terminal. The `lock` command is provided with the IBM/4.3 system to allow users to lock the terminal in order to be able to step away for a few minutes. Consult the manual page `lock`(1) for complete details on this program.

4. Password Security

A password is a user's main line of defense against unwanted persons reading, copying, or even removing files and directories owned by the user. It is essential that each user account have a password, and that the password be as secure as possible (generally, at least 6 to 8 characters long, with a mixture of letters, numbers, and special characters). Remember that the encrypted form of each user's password can be inspected by anyone on the system, since the file `/etc/passwd` is publicly-readable. While it is fairly difficult (though not impossible) to decrypt an encrypted password, it is relatively simple, using the programming tools of the IBM/4.3 system, to encrypt a string and then see whether the result matches any existing encrypted passwords. Short passwords can easily be broken by testing all possible combinations of letters and special characters. Even long passwords can sometimes be uncovered if the user selects a word that exists in a publicly accessible list such as an on-line dictionary or telephone directory.

Password security is particularly important to the superuser account, and the system administrator must take special care to guard the superuser password against accidently disclosure. As discussed in earlier chapters, the system superuser has practically unlimited powers when logged in to the IBM/4.3 system. A superuser can remove files and directories, change permissions on programs, and render disks and other peripherals unusable. It is therefore particularly important to keep unauthorized persons from knowing the `root` password or from otherwise assuming superuser privileges. The superuser password should never be given to another user just to allow that user to remove a lock file, or start a daemon, or any other administrative task, no matter how simple.

4.1. User Password Security

Most system break-ins begin when a bad guy gains access to a system as a normal user, not as the superuser. It is important, therefore, that users protect their passwords by selecting suitably random ones and by not revealing their passwords to other users. Unfortunately, it is all too simple for a user on a system to finesse a password out of unsuspecting users, especially users at publicly-available terminals, by writing a script to trap passwords. :p.Imagine that a malicious user logs in at a public terminal, executes the script, and then leaves the area. When another user attempts to log in, the script mails the user name and password to the bad guy, and then ends. Control returns to the shell, and the unsuspecting user might not notice anything unusual.

There is no real defense against this type of trick except to encourage user to change passwords as often as possible so that the stolen password will soon become outdated. Users should also monitor the time of last login (displayed by the system each time the user logs in) to ensure that someone else has not guessed or stolen their password. If such a security breach occurs, the user should notify the system administrator and change the account password immediately.

4.2. Password Aging

In order to help prevent unauthorized break-ins, it is very useful to have users periodically change their passwords. The IBM/4.3 system does not incorporate password aging mechanisms found on some earlier UNIX-based systems, largely because those mechanisms were difficult to use and maintain. It is, however, relatively easy to institute password aging by having `cron` periodically run a script that checks users' current passwords against a saved version of previous passwords. The following script has been found to work quite well:

```
#! /bin/sh
# agepasswd: Check age of passwords
SYSADM=root
PWFILE=/etc/passwd

cd /etc/passwd.age.data
awk -F: '{printf "%s %s0, $1, $2}' $PWFILE |
     while read user password junk
     do
        if (test "$password" != "XXX" -a "$user" != "who")
        then
          if (test -f $user)
          then
               if (test "$password" != "`cat $user`")
               then
                         echo "$password" > $user
               fi
          else
               echo "$password" > $user
          fi
        else
          rm -f $user
        fi
        if (test -f $user)
        then
          if (test "`cat $user`" = "")
          then
               echo "
Your password is invalid.  Please change it." |\
               /usr/ucb/mail -n $user
               echo "
User $user has an invalid password."  | /usr/ucb/mail -n $SYSADM
          fi
        fi
     done
```

For each line in the password file, this section of the script checks to see if there is a saved password in the directory `/etc/passwd.age.data`. If there is and it differs from the current password, then the saved password is updated; if there is no saved password, then the current password is saved. Finally, if the password is null, a note is mailed to the user and to the system administrator, since such accounts are most definitely security holes.

```
userlist=`ls`
for user in $userlist
do
     userline=`grep "\^$user" $PWFILE`
     if (test "$userline" = "")
     then
          rm -f $user
          echo "
User $user rm'd from passwd monitor." | /usr/ucb/mail -n $SYSADM
     fi
done
```

This section of the script checks to see whether there are saved passwords for accounts that no longer exist. If so, the saved password is removed and the system administrator is notified. The actual password aging mechanism is contained in the remaining section of the script:

```
userlist=`ls`
if (test "$userlist" != "")
then
     almost_old=`find $userlist -mtime 50 -print`
     if (test "$almost_old" != "")
     then
          for user in $almost_old
          do
               echo "
Your password has not been changed in the last 50 days.
You must change your password at least every 60 days.
Please change it." |\
               /usr/ucb/mail -n $user
          done
     fi
     too_old=`find $userlist -mtime +60 -print`
     if (test "$too_old" != "")
     then
          echo "
These users have not changed passwords
in the last 60 days: $too_old" |\
               /usr/ucb/mail -n  $SYSADM
          for user in $too_old
```

```
            do
                    echo "
    Your password has not been changed in the last 60 days.
    Please change it." |\
                    /usr/ucb/mail -n $user
            done
        fi
    fi
```

First, the remaining saved passwords are checked to see which are older than 50 days. If there are any such passwords, the user and the system administrator are notified. Then, the script searches for passwords older than 60 days and notifies the concerned persons if any are found.

To start up password aging, simply install the entire script as `/etc/passwd.age`, create the directory `/etc/passwd.age.data`, and then insert the following line into one of the `cron` configuration files, probably `/usr/lib/crontab.local`:

```
    35  23  *  *  *  root  /etc/agepasswd
```

In the late hours of the night, the `cron` daemon will run the aging script.

4.3. Password File Security

The file `/etc/passwd` maintains all account-related information, and it is essential for logging in. If it is destroyed or otherwise made unreadable by the system, no one (not even the superuser) will be allowed to log in to the system. The system administrator should therefore have a *current* copy of the password file available on a backup medium. It is also recommended that a copy of the password file be kept available on-line, so that the copy can be used to restore an original that has become corrupted.

More importantly, the protection modes of the password file must be set correctly. Generally, the password file should not be writable by any user on the system except the superuser, though for various reasons it must be readable by everyone. Thus:

```
    # ls -l /etc/passwd
    -rw-r--r--  1 root          8438 Mar 26 01:56 /etc/passwd
```

You should also ensure that the directory `/etc` is not writable by anyone other than the superuser. Recall that anyone with write permission on a directory can rename files within the directory, or even remove them, *regardless of the permission settings on the files.*

To protect against security violations on the passwd file (and others), make sure that the directory permissions on `/etc` are set up correctly. If you are in doubt, run the following command line:

```
    # chmod 755 /etc
```

This will ensure that the `/etc/` directory has the correct access permissions.

5. File Security

It is essential that system files and directories be given the proper permission, *before* the system is made available to normal users. The system administrator is responsible for making sure that all programs and files installed in publicly-readable directories (such as `/bin` and `/usr/bin`) have the appropriate permissions and ownerships, and you should not assume that the system as distributed conforms to the requirements of a secure system. For the most part, a program that will be executable by anyone on the system should be of ownership and group `bin` and should have protection mode 755. This establishes global read and execute permissions on the program, but restricts write access to the owner of the program. There are exceptions to this rule, especially when a program is designed to be run by a daemon process and not directly by a user. For example, some groups of commands such as the line printer spooling system and the uucp system require special ownership and execution modes. Consult the appropriate chapter in this guide for details on the required configuration.

6. Security for Setuid Programs

A bad guy is rarely content to break into a system just once. If a bad guy succeeds in becoming root and acquiring superuser privileges, he is likely to leave behind certain "burglar tools" that will make it easy to become root in the future. Of special concern are files that run setuid root.

Recall that a program can be configured to run with the file and directory permissions *of the owner of the file* (not the person invoking the program) by setting the setuid bit of the file on. A program that runs setuid root, then, allows the user to acquire superuser permissions as long as the program is executing. Normally, this is only a problem for programs that allow the user to give commands interactively or to spawn subshells.

Suppose that a bad guy succeeds in acquiring root privileges once. The bad guy knows that you will probably uncover that loophole soon enough, so his first course of action is to create another way to achieve superuser status. There are some simple ways to do this. Further, any bad guy intelligent enough to acquire superuser privileges will most certainly make it difficult to detect his necessary scripts and files by giving them an innocuous-looking name. You can trip up such a bad guy by maintaining a list of the setuid programs on your system and by periodically comparing the list to the actual setuid programs on your system. The following simple script provides a framework that you can modify as necessary:

```
#! /bin/sh
# check_setuid: check setuid root files against a saved list
SYSADM=root
cd /etc
find / -user root -perm -4000 -o -perm -2000 -print > setuid.tmp
if [ -f setuid.log ]
then
        diff setuid.tmp setuid.log | \
                /usr/ucb/mail -n -s "setuid root problems" $SYSADM
else
        mv setuid.tmp setuid.log
fi
rm -f setuid.tmp
```

When installing this script (presumably to be run periodically by cron), make sure that the file /etc/setuid.log is not readable or writable by anyone other than root.

It should be noted that the IBM/4.3 system automatically provides a very important setuid security mechanism not found in some earlier versions of the UNIX operating system. Previously, system administrators had to make sure that every program configured to run setuid root was not writable by anyone but root alone. The reason for this was that a file retained its protection modes even if another file was moved onto it (for example, using cp or mv). A publicly-writable setuid root program could therefore have had a shell program copied onto it, resulting in a setuid root interactive shell. In the IBM/4.3 system this is not possible, since writing to a file clears the setuid bit if the person doing the writing is not the superuser.

Such safety mechanisms notwithstanding, it is difficult to overstate the extent to which setuid and setgid programs can pose a significant security risk on the IBM/4.3 system. You should make certain that existing setuid and setgid programs are correctly installed. Just as importantly, you should be extremely careful in admitting new setuid/setgid programs onto the systems you administer. It is almost always possible to achieve the desired effects without resorting to the setuid or setgid mechanisms. If you must however install some such program, try to install it with non-superuser ownership, thereby attempting to minimize the potential damage if a clever user should succeed in breaking the security of the program.

7. Security for Device Special Files

The directory /dev contains a number of special device files that are used by the operating system to manage communications with terminals (usually having a device file with a name like /dev/tty1), networks, and storage media such as floppy and hard disks. There are also special files corresponding to the internal memory of the system itself (usually called /dev/mem and /dev/kmem). It is imperative that these special files have their permissions set correctly in order to prevent unauthorized access to their contents.

The special files /dev/mem and /dev/kmem should be readable and writable only by the super-user (or by the operating system itself). They should be owned by root and belong to the group kmem. For example:

```
# ls -lg /dev/{kmem,mem}
crw-r-----  1 root     kmem        3,   1 Aug 12 12:10 /dev/kmem
crw-r-----  1 root     kmem        3,   0 Aug 12 12:11 /dev/mem
```

The disks of the system must also be protected by correctly setting the protection mechanisms. All the special files in the /dev directory that correspond to disks should be owned by root, should be in the group operator, and should have permissions mode 600. For example, the device files for hard disk 0 should look something like this:

```
brw-r-----  1 root     operator  1,   0 Jan  9 22:18 /dev/hd0a
brw-r-----  1 root     operator  1,   1 Jan  9 22:18 /dev/hd0b
brw-r-----  1 root     operator  1,   2 Jan  9 22:18 /dev/hd0c
brw-r-----  1 root     operator  1,   3 Jan  9 22:18 /dev/hd0d
brw-r-----  1 root     operator  1,   4 Jan  9 22:18 /dev/hd0e
brw-r-----  1 root     operator  1,   5 Jan  9 22:18 /dev/hd0f
brw-r-----  1 root     operator  1,   6 Jan  9 22:18 /dev/hd0g
brw-r-----  1 root     operator  1,   7 Jan  9 22:18 /dev/hd0h
```

If these device files were readable or writable by other users, the security of files and directories stored on them would be endangered, since a bad guy could easily write a simple program to extract or modify the data stored therein.

8. File Encryption

The files and data stored on a computer system are very often of a relatively sensitive nature, such as company product plans or employee salaries. The security protection offered by the system of login names and passwords, together with the system of file and directory permissions, may not be adequate for such projects. Although these protections can ensure that most users will be denied access to certain files, the superuser can always look into any files that reside on the file system. In cases where even the superuser must be kept from looking at certain files because of their highly confidential nature, the user may encrypt, or encode, those files. This process translates the files into a form that is meaningless to anyone who cannot decrypt the files.

On the IBM/4.3 system, this encoding and decoding of files and data is accomplished by means of the crypt command. To use this command to encode a file, you must first select a *password* or *key* that crypt uses to transform the input file. As with login passwords, the crypt key should be at least six alphanumeric characters; if the password is shorter than that, the encryption is much easier to break.

Suppose that the file salaries is to be encrypted, and that the key you have chosen for the encryption is 'mykey6'. Then the command:

```
% crypt mykey6 < salaries > sal.encr
```

will encrypt the original file salaries (also called the "clear text") and place the results into the file sal.encr. To decode the encrypted version of the file, use the command:

```
% crypt mykey6 < sal.encr
```

The clear text will be send to the standard output, in this case the terminal screen. You may of course also redirect the standard output; for example:

```
% crypt mykey6 < sal.encr > sal.clear
```

After this command is run, the two files `salaries` and `sal.clear` will be identical.

You should be aware of two potential security loopholes in using the `crypt` command in this way. First, you will notice that the encryption key was specified on the command line, so it might be visible to anyone who happened to run the `ps` command at the right moment. To circumvent this problem, you may omit giving the encryption key on the command line. If you omit it, however, then `crypt` will prompt you for it interactively. For example:

```
% crypt < salaries > sal.encr
Enter key:
```

As with normal user password checking, `crypt` will not echo the key on the terminal screen when you enter it.

The second security loophole possible when using `crypt` concerns the original clear text files on the system. The user must remember to remove these files once they have been encrypted, or else all the good security work will have been in vain. The best solution to this problem is never to leave the clear text files on the system. This may be accomplished by creating and editing sensitive files using the encryption option of the text editor `vi`. To edit (or create) an encrypted file, give the command:

```
% vi -x sal.encr
```

Here the `-x` option instructs the editor to read (or create) the encrypted file `sal.encr`. You will be prompted for the encryption key before anything else happens. If you supply the correct key, then the file will be opened in its clear text form and you may edit it at will. When you save and quit the file, it will be saved in an encrypted form, using the same key you earlier specified. The line-oriented editors `ed` and `ex` also allow you to edit and save encrypted files.

9. Mail Encryption

If you want to send sensitive files or messages through the electronic mail system, there is a set of commands that attempt to implement a secure communications channel. In order to use this facility, the intended recipient of the mail must select a password or encryption key and use it to register with the encrypted mail system. That user must then supply the key when the mail is received. You will not, however, be prompted for the key when you send the mail.

To illustrate, suppose that you and another user wish to exchange some private mail. First, that user must register with the secret mail system by issuing the command `enroll`. The sequence will look like this:

```
% enroll
Gimme key:
%
```

As with passwords, the encryption key is not echoed on the terminal screen as it is typed. The user is now enrolled in the secret mail system and may be sent messages. This is done with the command `xsend`, which operates very much like `mail`. For example:

```
% xsend monroe
I agree entirely with your assessment of Smith.  I have
instructed the system manager to remove his account
immediately.  If the security leaks do not stop, then
perhaps we should also notify Curmudgeon.

Mark
.
```

The intended recipient will be notified by normal electronic mail that some secret mail has arrived. To read the secret mail, that user would type:

```
% xget
Key:
```

Note that the system has responded by requesting the secret key or password. If the same key is provided as the one given at enrollment time, the mail will be printed in an unencrypted form.

Unlike normal electronic mail, not even the superuser can read mail sent with this secret mail system, since both the message and the key are stored in a binary, encrypted form (in the directory `/usr/spool/secretmail`).

10. "Trojan Horses"

A "Trojan horse" is a command or a shell script that is masquerading as some familiar command. A bad guy can use Trojan horses to accomplish all sorts of mischief in the IBM/4.3 system, so you had better learn how to recognize and deal with them. Several scripts discussed earlier in this chapter are examples of Trojan horses.

There are two lessons to learn from these examples. The first lesson concerns the writability of directories used to hold publicly-executable commands. If anyone on the system can insert files into program directories, then the possibility is always open that bad guys will insert various Trojan horses there. So, the first line of defense is to make sure that program directories like `/usr/bin`, `/bin`, and others are not writable by normal users. There is also a corollary to this lesson: if you know (or suspect) that some unauthorized person has succeeded in acquiring superuser privileges, then you should scrutinize public program directories to make sure no Trojan horses were planted there. Especially keep an eye on the commands in `/usr/new` (if it exists on your system), since it is quite common for commands located there to have the same name as commands located elsewhere.

A second and more important lesson is this: you should make sure that search paths, especially for the superuser account, are set up so that public program directories are searched *before* any other directories. In particular, the current directory (indicated in the search path as '`.`') should be placed after other important directories in the search path. You may accomplish this by including a line like the following one in your `.cshrc` file:

```
set path = ( /etc /usr/ucb /bin /usr/bin . /usr/new )
```

Here the current directory will always be searched only after the four main program directories are searched.

To appreciate the full value of this last point, suppose that you, the system administrator, have the following search path:

```
# echo $PATH
.:/etc:/usr/ucb:/bin:/usr/bin:/usr/local:/usr/new
```

Now imagine that a user complains to you that all the subdirectories under his home directory are suddenly missing. Most probably, you will move into that home directory and run the command `ls` to see what's there. If you do so, however, you are likely to fall prey to a Trojan horse in the form of a bogus `ls` command in that user's home directory. Imagine that there is an executable script there containing the following commands:

```
#! /bin/csh
cp /bin/sh /tmp/RV33421
chmod ugo+s /tmp/RV33421
/bin/ls $argv[*]
```

As you can see, you have just helped the bad guy in his quest to assume superuser powers by making an innocuous-looking copy of an interactive shell program and then making it `setuid root`. Only afterwards was the real `ls` run. If you don't notice the file `ls` in the current directory, you would be none the wiser that *you* have just created a tremendous breach of security (namely the creation of a publicly-executable `setuid root` shell)! To repeat, make certain that the current directory is listed *last* in your search path, or at least after the main directories holding executable programs. To be extra safe, remove the '`.`' entry and all world-writable directories from the superuser search path.

11. Modem Security

Having a modem attached to your system that is configured to accept logins is always a potential security hazard. This is largely (though not exclusively) because you have virtually no control over who may call your system and attempt to log in. The first level of protection against unauthorized logins over the modem is to try to keep the phone number of the system as private as possible. If you are not operating a public time-sharing system, do not broadcast or publish modem numbers. If necessary, you may want to change modem phone numbers periodically. Remember that obscurity entails a certain amount of security.

The second level of defense against modem break-in is to make sure that every account on the system has a password. If a dial-in bad guy can find a user name for which there is no password, he will be able to log on as that user, and he may be able to attain superuser status by one of the means discussed above. You can detect accounts without passwords by running the following command:

```
# grep '[^:]*::' /etc/passwd
```

If there is any output from this command, it should be just for accounts running under restricted shells, like a `who` account that runs `/bin/who` as its login shell and then exits. If there are any other accounts without passwords, you should either deactivate the accounts or add passwords to them. Note that the password aging script given earlier in this chapter will automatically perform this check each time it is run. Incidentally, the `who` account itself may constitute somewhat of a security risk, since someone who discovers your modem number will be able to glean useful information about valid user names on your system, thereby making it slightly easier to break into the system. If you are highly concerned about modem security, remove the `who` account.

You can prevent someone from logging in as `root` across a dial-up line by suitably modifying the file `/etc/ttys`, as described in an earlier chapter. Recall that if the fourth field of a line in that file contains the annotation '`secure`', then the superuser can log in on the corresponding terminal line. If there is no such annotation for any dial-up lines, then no one will be able to log in as `root` on them. Determine whether or not you need to have remote superuser access to your machine; if you do not, then remove the argument '`secure`' from all dial-up line entries.

The final defense against modem break-in is to make sure that all users log out when they are done working over the modem. It is generally not sufficient for a user simply to turn off the modem, for this may leave a shell on the host machine still talking to the modem. The next person to dial up the modem will not have to log in, and will have access to all that user's files and directories.

12. Printer Security

To ensure adequate security for the line printer spooling system, it is essential that the owner and group of the various programs that comprise the system be set correctly and that the spooling areas be given the correct permissions. This prevents users from removing or modifying spooled output that does not belong to them, or from circumventing the printer accounting mechanisms. Unless your site has made significant modifications to the line printer system, the following steps should be taken:

- Make sure that the spooling areas are writable only by the user `daemon` and the group `daemon`.
- Make sure that the user program `lpr` runs `setuid root` and `setgid daemon`. Also make sure that the programs that manipulate the spooling queues (`lpd`, `lpq`, and `lprm`), run `setuid root` and `setgid daemon`.
- Make sure that the file `/etc/hosts.lpd` is not writable by normal system users.
- Make sure that the printer data base files `/etc/printcap` cannot be written by ordinary users. Typically this file is owned by `root`, belongs to group `staff`, and has permissions mode 644.
- Make sure that the printer accounting files (as specified in the printcap entry for each printer) cannot be read or written by ordinary users.

As you have seen, it can be dangerous having `setuid root` programs doing so much work on your system. Although the various programs in the line printer spooling system have been carefully written in an attempt to avoid the most common `setuid` security problems, you might think it preferable to remove the `setuid root` configuration altogether. In a networked environment where remote spooling is allowed, however, that is simply not possible. But if the IBM/4.3 system you are administering is *not* connected to a local area network, then you can relax the owner and group membership of the programs in the spooling system. Previous versions of the line printer spooling system had `lpd` running `setuid daemon` and `setgid spooling`, while `lpq` and `lprm` ran `setgid spooling`.

Finally, you must also ensure the physical security of the printer and its output. It does very little good to make certain that the line printer system is secure from tampering if a passerby can pick up whatever output happens to be near the printer. And it does a user little good to set up restrictive access permissions on files and directories if the physical hardcopy can be inspected by untrusted persons. Exactly how you maintain the security of the printer and its output will depend heavily on the use of the printer and the volume of output. Determine what measures are appropriate for your installation and then implement them.

13. Uucp Security

Software packages like uucp that allow file transfer to and from your system and unattended command execution on your system can pose significant security risks if the software is improperly installed or configured. Unless you actively restrict the uucp system along the lines suggested below, then any outside user who can log into your system will be able to execute any commands and copy any files available to the `uucp` login. Before activating the uucp system for intermachine communication, you should ensure that file ownerships are set correctly, that uucp-related login accounts are set up securely, and that certain configuration files have the correct access permissions.

13.1. File Security

Various control files used by the uucp system contain highly sensitive information (such as the names of systems you communicate with, and login account names and passwords) that should be kept secret from normal IBM/4.3 system users. In particular, the files `L.sys`, `USERFILE`, and SQFILE should be owned by the user uucp and should be readable and writable only by that user.

13.2. Password Security

The uucp system requires an account for the uucp system administrator. Usually the account name is 'uucp'. As indicated above, this account owns most of the files and programs that make up the uucp package. It is recommended that you create a separate login account for each remote system that is to communicate with your local system using the uucp package. For example, if you maintain uucp links to three systems named 'tic', 'tac', and 'toe', then the following lines (or suitable variants) should be added to the password file, /etc/passwd:

```
uutic::6001:1:uucp from tic:/usr/spool/uucppublic:/usr/lib/uucp/uucico
uutac::6001:2:uucp from tac:/usr/spool/uucppublic:/usr/lib/uucp/uucico
uutoe::6001:3:uucp from toe:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

You will then need to initialize passwords for these three accounts and convey that information to the uucp administrator at each remote site.

13.3. Command Execution Security

The file /usr/lib/uucp/L.cmds contains a list of commands that a remote uucp user may execute on your system via the uux command. When uux receives a request to launch uuxqt to process some execute file, it first checks to see that the command specified in the execute file is contained in the file L.cmds. By adding commands to this file or removing them from it, you can expand or contract the functionality that your machine agrees to provide to your uucp neighbors. A typical L.cmds file might look like this:

```
rmail
ruusend
lpr
who
```

This would allow a uucp account to run the commands rmail and ruusend (which forward mail to other systems), lpr (to send files to a local line printer), and who (to see who is currently on the system). If you are concerned to limit access to your system (and it is not a network printer server), then your L.cmds file should look like this:

```
rmail
ruusend
```

The who command was disallowed since it is likely to provide potentially useful information to a bad guy attempting to log in to your system under some account other than uucp. You can disallow a remote system from running *any* commands on your system by creating the file L.cmds zero-length.

13.4. Conversation Sequencing

The uucp system can be configured to keep a log of conversations with each remote system, including a count of the number of conversations between the local system and the remote system. By comparing these numbers, the two systems can effectively verify that they have been talking with each other. If the records do not agree, then one system may be trying to impersonate another, in which case the login attempt should be disallowed.

The record of conversations is maintained in the file SQFILE in the uucp administrative directory, /usr/lib/uucp. This file contains a line for each system that is to have conversation sequencing checked. Initially, just the name of the remote system is put into this file. Then, when a successful conversation takes place, each system will update its sequence file to reflect the activity.

By default, the conversation sequencing is disabled, since in practice it is rarely used. In order to configure the software to perform the sequence checking, you must recompile the software with the GNXSEQ preprocessor option enabled. Of course, sequencing must be in effect on both sides of the uucp conversation.

13.5. Summary

Maintaining uucp security is not overly difficult once the system has been securely installed and configured. As you have just seen, it is possible to institute a call sequencing procedure that provides very good assurance that a remote system really is the system that it claims to be when initiating a uucp conversation. Nonetheless, it is simply undeniable the uucp system provides yet another avenue for bad guys to gain access to your system. In view of this fact, many sites prefer to isolate their uucp connection to the outside world from other machines on the local network. Even if a bad guy succeeds in breaching the normal security mechanisms of the uucp node, he will be unable to access the resources of the machines on the local area network.

14. Network Security

If your IBM/4.3 system is connected to a local area network, you will need to consider several issues related to maintaining a secure operating environment across the entire network. The networking capabilities provided by the IBM/4.3 system increase the computing power and flexibility available to both individual users and the system administrator; at the same time, they create additional security holes that must be closed by careful system administration.

The first level of network security is to ensure that configuration files on each host are correctly and securely set up. For example, each network host that permits remote login and remote command execution (using the `rlogin` and `rsh` commands from remote hosts) automatically employs an authentication scheme involving the file `/etc/hosts.equiv`. If a user on a remote machine, say `grunnion`, requests a network service from another machine, say `tuna`, then the server on `tuna` first checks to see whether the host `grunnion` is listed in the file `/etc/hosts.equiv`. If so, and if the user has an account on `tuna`, then the service will be performed as requested. The `/etc/hosts.equiv` file thus lists hosts that are treated as "equivalent" to the local host, at least in terms of allowing logins, command execution, and other network services.

There are two main exceptions to the host authentication scheme using `/etc/hosts.equiv`. First, if the user's home directory on `tuna` contains a `.rhosts` file listing the user's account on `grunnion`, then the service will be performed whether or not `grunnion` is listed in the `/etc/hosts.equiv` file on `tuna`. So users can obtain network services even from machines which are not considered equivalent in the overall network. Second, if the user requesting a service from a remote host is `root`, then only the file `/.rhosts` on `tuna` is consulted to see if superuser privileges should be accorded to the user `root` on `grunnion`. This allows the administrator on `tuna` to deny superuser access to anyone but a local `root`, if a very restricted network is desirable. It also allows the administrator to designate the `root` login on some other machine as equivalent to the `root` login on `tuna`, so that administrative functions can be performed across the network. Obviously, the file `/.rhosts` is an extremely important file that must be protected from writing by normal users. It is probably also a good idea to prevent other users from reading that file.

The file `/etc/hosts.lpd` is used in much the same way as `/etc/hosts.equiv`, to restrict or allow access to a local printer by other network hosts. In addition, the `rs` printer capability can be specified in order to disallow remote printing except to users who have accounts on the print server. For complete details on enforcing network security within the line printer spooling system, refer to Chapter 9.

There is one further configuration file relevant to the maintenance of network security, `/etc/ftpusers`. Recall that the `ftp` server running on a particular machine allows anonymous access to files and directories owned by a user named '`ftp`'. The `ftp` server allows practically anyone to look at and copy files located in such directories. In particular, the `ftp` server does not consult the file `/etc/hosts.equiv` to determine if it should service an `ftp` request. Instead, the `ftp` server consults the file `/etc/ftpusers` and *dis*allows `ftp` services to any users whose names appear therein. A typical `ftpusers` file contains at least the two names '`uucp`' and '`root`' You may wish to include additional names in this file if abuses by particular users warrant removing their `ftp` capability.

The most important concern in maintaining a secure network is to ensure the security of each individual workstation on the network. Remember that users may be prompted for login names and passwords when they attempt to establish a connection with a remote machine and that that information can be intercepted by any machine on the network. More generally, any file copied over the network essentially becomes public information if it is not first encrypted, since it is easy to write programs to monitor network traffic. As a result, if there is even one machine attached to your local area network that maintains user accounts for persons whose commitment to the security of the entire network is in doubt, or which is not itself immune from intruders, it may be impossible to achieve an acceptable level of data security in the networked environment. Even the mechanism of listing "trusted hosts" in the file `/etc/hosts.equiv` can be circumvented in a very straightforward manner. A bad guy who gains control over one machine in a local area network can easily have that machine impersonate any other host on the network.

For better or worse, a network is only as secure as the least secure machine on the network. As the system administrator concerned with maintaining a secure network, you must therefore first make each workstation as secure as possible against outside intruders and against inside bad guys. If a particular machine is consistently lax in security-related areas and you are unable to educate the user population accordingly, your only recourse may be to remove that machine from the network.

15. Miscellaneous Security Tips

- Make sure that the `cron` configuration file `/usr/lib/crontab` (and `/usr/lib/crontab.local`, if it exists) is owned by `root` and is not writable by anyone other than `root`. As distributed, this file is not writable even by `root`. To make changes to the `crontab` file, the superuser must edit it and then save the changes with the `:w!` command (assuming the editor is `vi`). Also, since there is really little difference between commands placed directly into the `crontab` file and any scripts that are called by it, make sure that users cannot alter those scripts (and possibly make one into a Trojan horse).
- Make sure that you completely understand the format of the various configuration files used by the system and its utilities *before* you undertake to modify one of them. You can create tremendous security loopholes if you improperly modify such files. Where possible, use system-provided utilities that are designed to assist you in modifying configuration files. (For example, the utility program `vipw` performs several consistency checks on the file `/etc/passwd` when you try to update it.) Similarly, if local scripts are available for such mundane tasks as adding new users to the system, use them.
- Install all `setuid` and `setgid` programs so that they cannot be either written or read by anyone on the system (i.e., establish the permissions so that they are `--s--x--x` or `--x--s--x`). By preventing read access to such programs, you will also prevent a bad guy from gaining information that may potentially be useful to him in breaking the security of the program.
- Periodically scrutinize the contents of `/etc/rc` and `/etc/rc.local` to ensure that the multi-user initialization commands they contain are indeed current for your installation. A bad guy who succeeds in breaking `root` may plant commands in them that compromise system security.

- Remove the `guest` login account from your system, unless it is absolutely necessary. At the very least, make sure that it has a password (and that the password is different from '`guest`').
- It is a good idea to have a colleague (ideally, an experienced system administrator) attempt to break into your system at irregular intervals. Exactly how you arrange this depends on which aspects of system security you want to test.

16. Conclusion

The task of ensuring the secure operation of the IBM/4.3 system is one that demands your continual attention. Unlike the installation and configuration of a piece of hardware or software, security is never a one-shot deal. You must constantly be on guard against system intruders and against system users who want to win at the ultimate computer "game", achieving superuser status. The price you and your system pay for losing at that game can be a very dear one. In a reasonably dire case, it may involve doing a full backup of all file systems, a task which may inconvenience you and your users for several hours at least. If, however, the physical security of your system was breached and your backup media have been lost or stolen, the price may be even greater, since you risk losing literally years of work because of a moment's indiscretion.

Even in an environment that is devoid of such "bad guys", you must make sure that the user community fully understands the protection mechanisms that are built into the IBM/4.3 system. The size and complexity of the IBM/4.3 system virtually guarantee that mere common sense is never sufficient to maintain a secure and functional computer system. User education, especially concerning file access permissions and password security, is therefore a fundamental part of the system administrator's task. The security of the shared computing environment provided by the IBM/4.3 system is everybody's concern, not just the system administrator's.

The final word on system security (at least in *this* guide) is simply that security is worth whatever additional time and effort it takes to get things right, whether that means periodically traversing the file system to ensure that files and programs have the correct permissions, or scrutinizing some source code to make certain that a new `setuid` program does not contain any inadvertent security loopholes. Every time that you add some new piece of software or hardware to your system, you should carefully consider the impact that the new item will have on your existing security measures. Above all, if a system administrator's manual accompanies the new product, take the time to read it thoroughly, looking especially for any discussions of system security.

CHAPTER 19

Understanding the Andrew File System and the Andrew Toolkit

1. Introduction

This chapter describes the Andrew File System and the Andrew Toolkit applications. The Andrew File System is a distributed file system for a large network of personal workstations; it operates under IBM/4.3 and runs on the IBM RT PC and the IBM 6152 Academic System, and is described in the first section of this chapter. The Andrew Toolkit Applications is a set of four application programs that can be used with or without the Andrew File System. The second major section of this chapter describes these programs.

2. The Andrew File System

The diagram below is an overview of the Andrew File System; definitions of the terms in the diagram follow it.

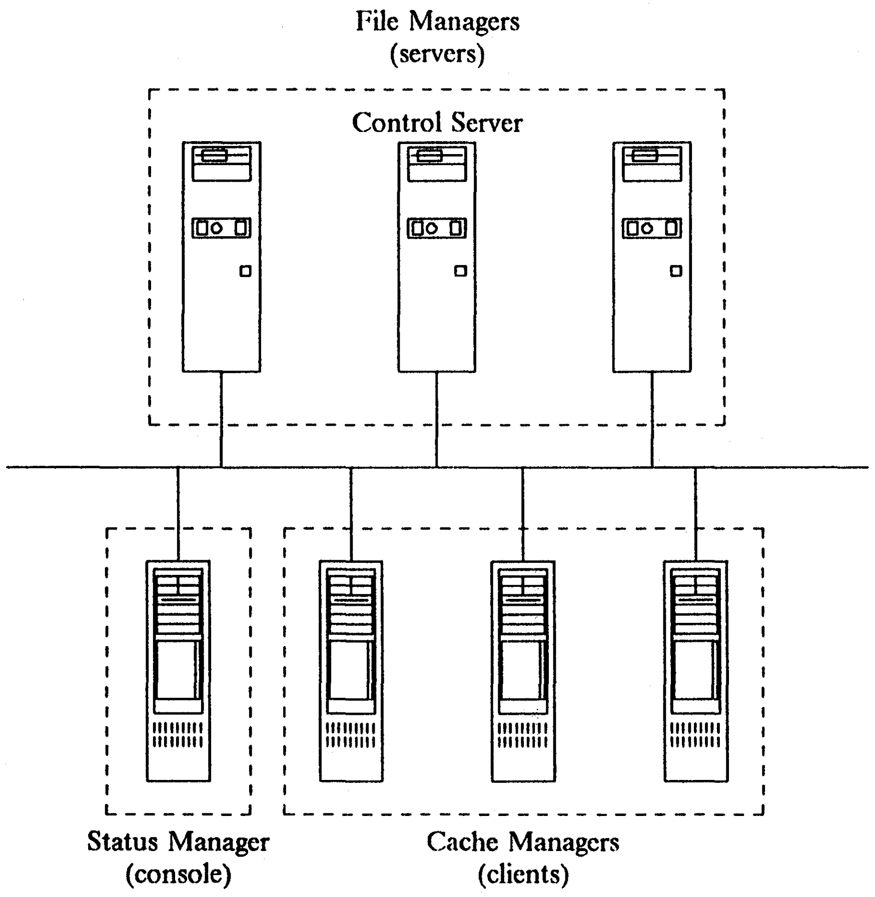


Figure 19-1: Overview of the Andrew File System Architecture

Managers Managers are programs that are responsible for controlling different aspects of the Andrew File System.

- The File Manager runs on the servers and presents files to the workstations.
- The Cache Managers reside on workstations and translate workstation requests into calls to the File Manager to present the requested files.
- The Status Manager provides a status display of all of the file managers.

Servers Servers are machines in which files are stored. The control server is a central point where change is introduced into the Andrew File System. In general, servers are separately administered and are separate from workstations.

Console The console is a machine used to run the file manager monitor program. The console will indicate if a file manager is down and is also useful for determining how well a file manager is running when it is up. The console can be any client workstation. The console does not have to be a dedicated machine; an operator's workstation is often used as the console. Any machine that is running a copy of `vopcon` is a console machine. From a console machine you can monitor the file managers.

Control server The control server administers the other servers centrally; a process runs on the control server that is used by the other servers to keep files on their local file systems up to date.

To users of the Andrew File System the main unit of data is the file, which is the unit passed between the Andrew File Manager and the Andrew Cache Manager. To the administrator, however, the important unit of data is the *volume*. A volume is a hierarchical group of related files, for example, the files belonging to one user. The volume is the basic organizing mechanism for data stored within the file system; they are the units for addressing, storing, and accessing data, for moving data from server to server, the unit to which quotas are applied, and the unit of data that is restored if a user loses a file.

Much of the administrator's work deals with maintaining volumes, including the following tasks, which are discussed in this section:

- Creating and naming volumes for new users
- Moving volumes between servers to handle space or load balancing considerations
- Replicating volumes
- Deleting volumes
- Creating backup volumes

For information on operational tasks, such as adding new users, deleting users, and adding a new server, see the section "Operation Description" in Part I of *The IBM Andrew File System.* You will also find information on monitoring the Andrew File System under the section "How to See What Is Happening" in the same article.

2.1. Creating Volumes

A volume is typically a group of related files, such as all the files belonging to one user. Before creating volumes, you will need to decide on the number of volumes you want, and their sizes and names. Once you've set things up initially, you will create a new volume when you add a new user. You should keep the following considerations in mind when deciding how to divide the file system tree into volumes:

- Volumes should be small enough that moving volumes between partitions is a reasonable approach to balancing server disk utilization and load; for example, you would not want to make each volume just larger than 50% of the available space in a typical partition. Doing so would waste almost half the available space. Making volumes fairly small relative to partition size makes it easier to move volumes around to gain an most efficient use of space.
- The size of the volume must not exceed the capability of the backup media. The Andrew File System backup system expects that the entire contents of a volume can be backed up on a single unit of external storage (e.g., tape). The backup system allows more than one volume to be backed up to the same tape. Using smaller volumes improves tape utilization.

When deciding on volume names, you should keep the following in mind:

- Choose meaningful names, i.e., names that will tell you something about the volume's contents and owner. During system administration tasks, you will probably encounter volume names out of context; using names that give information about the contents of the volume can help you in these tasks.
- Using prefixes can help in assigning meaningful names. For example, you might choose the prefix `user` for all user volumes; names of all user volumes would begin with this prefix, followed by a period and the user's login ID. Using a scheme like this is also useful during backup because volumes on tapes can be grouped according to volume name prefix. A volume name can be more than 32 characters long.
- You'll find it advantageous to develop a naming scheme (such as using prefixes) that is reasonably consistent, particularly when your system grows.
- Quota enforcement is done on a volume basis; accounting can be as well.
- Once you decide on a design, you can move volumes, but you cannot use the rename operation across volume boundaries.
- Manipulating the tree structure on a basis other than a whole volume (you can move the mount point) or less than a volume (you can move directories or files within a volume) is difficult. Moving a subtree of a volume to another volume requires moving every byte of data in the subtree.

Once you've determined which volumes you want to create, their sizes, and names, you use `createvol`(8V) to create volumes at the appropriate server. You don't need to worry too much about where you create a volume; you can create volumes on a single partition within the system, then move volumes to another partition when necessary. When a volume is first created it uses little of the system's resources; it may also take some time to be visible at workstations.

`Createvol` has the following format and arguments:

`createvol` *volname server partition*

The arguments specify the volume name, the server where the volume will be created, and the server where the volume will be created, respectively. The partition name shold always start with the character string `/vicep`.

Once you've created a volume, you can, if you wish, use the `vol-lookup`(8V) command (normally used only for debugging) to make sure the volume was actually created. To use it in this way, type `vol-lookup` followed by the name you specified for the volume

2.2. Moving Volumes

If space within a partition is at a premium, or if a file manager becomes overloaded, you may need to move one or more volumes to free up space. To move a volume from one partition to another, on the same server or on another server in the system, you use the `movevol`(8V) command. The volume being moved can be used continuously while it's being moved; users may get a "volume busy" message from the cache manager, but the wait should be short, from a few seconds to a minute or so. The actual move takes much longer; the amount of time required depends on the size of the volume, how busy the server is, and whether the move has to wait for any other volume operations involving the same servers.

`Movevol` has the following format and arguments:

`movevol` *volname server partition*

For *volname* you should supply the name of the volume you want moved; for *server* you should supply the name of the server or partition to which you want the volume moved. `Movevol` invokes the `vol-dump` and `vol-restore` commands; for more information, see the man pages for them.

2.3. Replicating Volumes

The Andrew File System can replicate volumes for read-only access so that more than one server can access the same volume. This can be especially useful in large systems where many users would be inconvenienced if a server crashed, and where the demand for certain files is sufficiently high to warrant distributing the load over multiple servers. Replicating volumes also helps to avoid a high number of "callbacks." A callback is a promise by the file manager to notify the workstation if a particular file changes; the file manager must keep the promise for every workstation that has recently used a file. Because read-only volumes don't change, callbacks are unnecessary. Workstations accessing replicated volumes will not see changes immediately because of the updating that must take place in the system.

To create a read-only volume, you use `vol-dump`(8V), `vol-restore`(8V), and `vol-clone`(8V) to clone a volume and propagate the changes to other servers. This is called releasing a volume. You will find examples of how it is done in `vol-restore`(8V). The read-only volume that results from this process is an exact copy of the original read-write volume. See the man pages for these programs for more information.

2.4. Deleting Volumes

When a volume is no longer needed, you use the `purgevol`(8V) command. you must use the `fs`(1V) command to do this. Before removing a particular volume you should be sure that there is a dump of the most recent copy of the volume in off-line storage. Also, once you've deleted a volume, you will need to remove related backup volumes as well.

It is also possible to delete a group of volumes from a file system, although you should do so with great care. For information, see `newfs`(8).

To use the `purgevol` command, you simply type `purgevol` followed by the name of the volume you want deleted.

3. The Andrew Toolkit

The Andrew Toolkit is a set of tools you can use for tasks such as writing and editing documents, sending and receiving mail, writing programs, and others. On a workstation you can use more than one applications at a time because each appears in its own window on the screen. The applications included in the toolkit are:

`ez`	An editor that provides full editing and formatting capabilities, and that is integrated with previewing and printing functions.
`console`	A program that monitors the system and displays system functions.
`tx`	A typescript program that provides user access to the full functions of IBM/4.3.
`hz`	A help program that provides detailed reference information in a window on the screen.

This sections explains two tasks involved in setting up a workstation for the Andrew Toolkit: configuring the file system, and setting up the user's home environment. For information on using the applications, see "The IBM Andrew Toolkit Applications User's Guide."

3.1. Configuring the File System

How you configure the file system for the Andrew Toolkit depends on whether you are running the toolkit applications in a standalone mode or in conjunction with the Andrew File System.

3.1.1. For a Standalone System

To configure the file system for a standalone system, you need to install the Andrew Toolkit from tape as described in "Installing and Operating IBM/4.3." The following directories are important to this process:

`/usr/andrew`	Contains the Andrew Toolkit subtree
`/usr/andrew/bin`	Contains the Toolkit executable code
`/usr/bin/X11`	Contains required X Window System components (Xibm)
`/usr/lib/X11/fonts`	Contains the X Window System fonts
`/usr/lib/font`	Contains printer-specific fonts

Additionally, the following directories and files are important for those wishing to use the Andrew Programmer's Toolkit (see Programmer's Guide To The Andrew Toolkit):

`/usr/include/X11`	Contains the X Window System include files
`/usr/lib/libX11.a`	Contains the X Window System Xlib library
`/usr/lib/libXt11.a`	Contains the X Window System Toolkit library
`/usr/lib/liboldX.a`	Contains the old Xlib library for back-compatibility

3.1.2. For a System Connected to the Andrew File System

For a system that is connected to the Andrew File System, you will need to create symbolic links to the appropriate Andrew File System directories and files. You do not need to do this if those directories or files are already local on the system as described above. The following commands are suggested. (Note that these assume a particular Andrew File System structure and should not be executed verbatim.)

```
ln -s /usr/andrew          /vice/usr/andrew
ln -s /usr/bin/X11         /vice/usr/bin/X11
ln -s /usr/lib/X11/fonts   /vice/usr/lib/X11/fonts
ln -s /usr/lib/font        /vice/usr/lib/font
ln -s /usr/include/X11     /vice/usr/include/X11
ln -s /usr/lib/libX11.a    /vice/usr/lib/libX11.a
ln -s /usr/lib/libXt11.a   /vice/usr/lib/libXt11.a
ln -s /usr/lib/liboldX.a   /vice/usr/lib/liboldX.a
```

Again, the actual location of the Andrew File System directories and files (e.g., `/vice/usr/andrew`) may vary from site to site and the preceding commands should be used only as examples.

3.2. Setting Up a User's Home Environment

To set up a user's home environment, you need to do the following:

(1) Add the following lines to the `.login` file:

```
        setenv DISPLAY :0
```

```
setenv PRINTER <printername>
setenv CLASSPATH /usr/andrew/dlib/be2
setenv ANDREWDIR /usr/andrew
setenv BE2WM x11
```

(2) Make sure the path statement in the `.cshrc` file includes the following:

```
/usr/andrew
/usr/andrew/bin
/usr/bin/X11
```

(3) Make sure that an Andrew preferences file exists in the user's home directory. The file `/usr/guest/guest/andrew/preferences.aug` may be used as a starting point and will provide the functions described in "The IBM Andrew Toolkit Applications User's Guide." You can copy this file into the user's home directory then either rename it to `preferences` (from `preferences.aug`), or else append it to an existing preferences file.

(4) In the same manner, make sure an `.Xdefaults` file exists in the home directory; you can use `/usr/guest/guest/andrew/.Xdefaults.aug` as a starting point.

3.3. Invoking the Andrew Toolkit

Executing the command `Andrew` invokes the Andrew environment as described in "The IBM Andrew Toolkit Applications User's Guide."

Glossary

If you are a new system administrator, or new to the IBM/4.3 system, some of the terminology employed in this guide may be unfamiliar to you. The following glossary is provided to give short but helpful characterizations of the terms. For a fuller discussion of any item, you should refer to the Index to see where it is discussed in more detail in this guide.

System V
System V is a version of the UNIX operating system.

account
A collection of files, directories, programs, and other objects that together allow a user to log in to the system and execute commands. The primary repository of account information is the file `/etc/passwd`.

accounting
Accounting is the process of keeping track of user login sessions, CPU usage, command usage, disk usage, etc., so that users may be charged for the resources they utilize or so that the system administrator can intelligently balance the resource utilization.

Andrew
The Andrew system is a set of software tools that allow the user to manage work in a distributed, multi-processing, windowing environment. The Andrew system consists of two major parts, the Andrew Toolkit Applications and the Andrew File System.

Andrew File System
The Andrew File System is a distributed file system for a large network of personal workstations.

Andrew Toolkit Applications
The Andrew Toolkit is a set of user applications that operate under the Andrew System. The four major applications are `ez`, `console`, `typescript`, and `help`.

baud
The baud rate of a device is a measure of how much data the device can process in a given amount of time. Technically, the baud rate of a device is the number of signal changes per second that the device is capable of decoding. Generally this means that a 2400-baud modem can send and receive 2400 bits per second. See also **modem** and **terminal**.

block
A block is a division of a file system, at least 4096 bytes in size (under the new "fast file system"). The IBM/4.3 system allows different file systems to have different sized blocks.

boot
To boot a system is to cause it to begin operation from a powered-down state.

byte
A byte is a sequence of eight adjacent binary digits that are operated upon as a unit. A byte is usually large enough to hold a single character.

console
A console is a terminal dedicated to system administration use. When the IBM/4.3 system begins operation, it writes some messages on the system console and takes input from it. In single-user mode, the console is the only operative terminal and is automatically given superuser privileges. In multi-user mode, the console is treated like any other terminal.

core dump
A core dump is a file named 'core' that is created by the operating system when a program terminates abnormally. You can examine the core dump with a debugging utility (such as adb or sdb) to try to determine what caused the abnormal termination of the program.

crash
A system crashes when it halts operation without having been told to do so. A variety of conditions,both software and hardware-related, can cause a system to crash.

cron
Cron executes commands at specified dates and times according to instructions in the file /usr/lib/crontab.

cylinder group
A cylinder group is a subdivision of a file system (and hence of a disk partition) consisting of one or more consecutive cylinders on a disk. A cylinder group contains a copy of the superblock, some i-nodes, a bitmap describing the free data blocks in the group, and some data blocks.

daemon
A daemon is a process that is not controlled by a terminal. Once launched, it sits quietly waiting for requests. When a request arrives, the daemon services it and then goes to sleep to await further requests. lpd and syslogd are two daemons that are usually running on a IBM/4.3 system.

datagram
A datagram is a collection of data that forms all or part of a message sent across a network.

directory
A directory is a special type of file that contains entries (called "links") that are references to other files. By convention, a directory contains at least two links, '.', which refers to the directory itself and '..', which refers to the parent directory (if it has one).

Ethernet
An Ethernet is a physical network that allows transmission of messages in any of a number of protocols. The IBM/4.3 system supports both TCP/IP and Xerox Network/System communication across an Ethernet. See also **TCP/IP**.

file
A file is a sequence of bytes. The operating system imposes no other particular structure on a file. System-related information about the file (who owns it, who may change it, how big it is, etc.) is not stored in the file itself, but in the file's identification node. See also **i-node**.

file descriptor
A file descriptor is an integer assigned by the system to a file when it is opened by certain system calls.

file name
A file name is a string of characters by which a file may be accessed. The name of a file may be up to 255 characters in length. Generally, it is unwise to use special characters (such as '*', '?', '[', ']') as parts of a file name.

file system
A file system is a hierarchical structure of directories and files used to manage the storage of data on a secondary storage medium (typically a hard disk). The top directory of a file system may be located at some point in another file system, called the "mount point", or it may be the root directory of the entire file system on your machine. See also **Andrew File System**, **directory**, and **i-node**.

finger
Finger is a program that lists such information as login name, terminal name, idle time, login time, and so forth.

gid

A gid is an integer assigned to a particular group that the system uses to store information about the group. 'Gid' stands for 'group identification number'. See also **uid**.

group

A group is a collection of one or more users. Groups are useful to allow several users to access a common set of files and directories. Under the IBM/4.3 system, a user may belong to several groups at once. The primary repository of group membership information is the file `/etc/group`.

head

`Head` is a program that lists the first few lines of one or more files.

home directory

A user's home directory is the top-level directory assigned to (and owned by) the user. It is the working directory that the user will be located in at login time. The user's start-up files and personal files and directories are located within the home directory.

i-node

An i-node is a part of the file system used to hold information relating to a file, such as the size of the file, the owner, the permissions, and the data blocks in the file system occupied by the file. 'I-node' is short for 'identification node'.

Internet Protocol Suite

A set of rules (or "protocols") that govern the transmission of data and messages among cooperating computers in a network. TCP and IP are the best known protocols in this suite, which is therefore sometimes referred to as "TCP/IP". The IBM/4.3 system provides TCP/IP services through a combination of kernel facilities (primarily sockets), daemons (such as `rlogind` and `ftpd`), and user-level commands (`rlogin`, `ftp`, etc.). See also TCP and IP.

interrupt

An interrupt is a signal sent to a program to cause the program to stop executing. Interrupts are also sent from I/O devices to the central processor when an error has occurred or when assistance is needed to complete the input/output request.

IP

IP is a member of the Internet protocol suite that provides message addressing services. IP routes the datagrams handed to it by other protocols (usually TCP) to the desired destination. 'IP' stands for 'Internet Protocol'. See also TCP.

kernel

The kernel is the heart of the IBM/4.3 operating system. It controls all processes running in the system and allocates resources to them as necessary. It manages the storage of all data on the system's disks. The kernel also takes instructions from a shell and executes them, making certain that the user issuing the instructions has the appropriate permissions on any programs and files that must be accessed in order to perform the requested task. The disk image of the kernel is located in the file `/vmunix`.

link

A link, or "hard link", is an entry in a directory that points to an i-node. There may be several links (possibly in different directories) pointing to the same i-node, and hence the same file. Hard links cannot, however, cross file systems. See also **symbolic link**.

login

Generally, to login is to connect to a computer system. On the IBM/4.3 system, the login sequence consists of the user providing a user name and a password. The system will respond by printing the file `/etc/motd` and by running any commands located in the file `.profile` (if the login shell is the Bourne shell) or in the two files `.login` and `.cshrc` (if the login shell is the C-shell).

login name

The login name, or user name, is one identifier by which the system recognizes a user. It is used in the output of many commands (such as `who`) or as an argument to many commands (such as `finger`). A login name is uniquely associated with a user identification number by the account entry in the file `/etc/passwd`.

login shell
The login shell is the program launched by the system at the completion of the login process. The login shell is specified by the last field of an entry in the file `/etc/passwd`. The most common login shells are `/bin/csh` and `/bin/sh`, although in theory any program can serve as a login shell.

mode
A mode is a method of operating. For example, the vi editor has two modes: text entry (for normal typing) and command (for issuing commands to the editor).

modem
A modem is a device used to allow two computers (or a computer and a terminal) to communicate across normal telephone lines. The modem translates electronic signals into audible tones, suitable for transmission across voice lines. 'Modem' stands for 'modulator-demodulator'. See also **baud.**

mount
To mount a file system is to make it accessible. Unless a file system is mounted, it cannot be read from or written to.

multi-tasking
Multi-tasking is running one or more processes (tasks) at the same time.

multi-user mode
Multi-user mode is the state in which numerous persons can log in to the system and execute commands. The IBM/4.3 system enters this mode automatically at boot time. Multi-user mode is not recommended for adminis.GE network

operating system
The operating system is the software that controls the execution of programs. The IBM/4.3 operating system provides resource allocation, process scheduling, I/O cont.GE password

path name
A path name is a string of characters that refers to a file. A path name may be either absolute (meaning that it begins with a slash, '/', and gives the entire path from the root directory to the file) or relative (meaning that the name starts in some directory other than the root directory). A path name may be at most 1024 characters in length.

peripheral device
A peripheral device is any piece of computing equipment that is not built-in to your computer but which is connected to it in some way, typically through a cable. Common examples of peripheral devices are terminals, modems, printers, and tape drives.

permissions
Each file and directory in the IBM/4.3 system has associated with it a set of permissions that determine who is allowed to do what with the file or directory. By suitably restricting permissions, a user can prevent other persons from looking at, copying, or removing personal files and directories. Permission information about a file or directory is stored in its i-node.

pid
Each active process in the system is assigned a unique positive integer in the range 0 to 30000 by which it is identified to the system. The pid is used by certain commands (such as `kill`) to select processes. 'Pid' stands for 'process identification number'. See also **process.**

printcap
`Printcap` is the name of a data base used to describe line printers. Each entry in the data base is used to describe one printer.

process
A process is an instance of a running program. Associated with each process is a process identification number (pid) which are used in various shell commands (such as `kill` or `stop`) to control the

operation of the process. See also **pid**.

quota
A quota is a limit placed by the system administrator on a user's ability to consume system resources. Under the IBM/4.3 system, an administrator may limit the total amount of disk space owned by a user, the total number of files owned by a user, or both.

root
Root is another name for the superuser.

root directory
The root directory is the top level of a tree-structured directory system.

run level
The system's run level determines what file systems are mounted and how many persons may log in. Single-user mode allows one user only, who must communicate through the system console; multi-user mode allows many users to log in, through any terminal attached to the system (or across a network).

search path
A user's search path is a list of directories in which the user's shell program will look in an effort to find a command given by the user. To see your current search path, look at the value of the `path` variable.

setgid
A program runs setgid when it allows the person executing it to assume the permissions of the group of the program, for the duration of the program. 'Setgid' stands for 'set group identification'.

setuid
A program runs setuid when it allows the person executing it to assume the permissions of the owner of the program, for the duration of the program. 'Setuid' stands for 'set user identification'.

shell
A shell is a command interpreter. It takes commands from a user and translates them into the appropriate kernel instructions. The shell is responsible for interpreting metacharacters (such as '`*`' and '`?`' in file names), quotation, redirection symbols, etc. The two most common shells are the C shell (`/bin/csh`) and the Bourne shell (`/bin/sh`).

shell script
A shell script is an executable text file containing shell commands and/or comments. A script is invoked in precisely the same way as a command, by giving its name to the shell.

single-user mode
Single-user mode is the state in which only one person is able to access machine resources. Some daemons are not yet running and file systems other than the root file system may not yet be mounted. When the IBM/4.3 system is in single-user mode, only the console terminal is active as a port into the system. Single-user mode is recommended for system administration tasks, since the file systems are in a relatively unchanging state and may be mounted or unmounted as needed. See also **multi-user mode**.

socket
A socket is an endpoint for communication between processes, especially between processes communicating over a local area network. Each socket has queues for sending and receiving data. Sockets originated in the BSD releases and are generally absent from System V-based machines.

source code
The source code for a program is a file or set of files containing the human-readable instructions that must be converted by a compiler (usually `cc`) into an executable program.

special process
Processes with pid's of 0,1, and 2 are called special processes, since these pid's are reserved for them and not reused. Process 0 is the scheduler. Process 1 is the `init` process and is the ancestor of all

other processes in the system. Process 2 is the paging daemon.

spool
To spool output is to place it in a special spooling directory where it will await further processing. For example, the program `lpr` places print requests into the directory `/usr/spool/lpd` where they are found and processed by the daemon `lpd`. Generally, spooled requests will be serviced in the order received. 'Spool' stands for 'simultaneous peripheral output off-line'.

standalone utility shell
The standalone utility shell allows the administrator to perform certain tasks without even having the IBM/4.3 system running.

start-up file
A start-up file is a file that is executed by the system when a user logs in. If the login shell is the Bourne shell, the system executes the file `.profile` in the user's home directory. If the login shell is the C shell, the system executes the two files `.cshrc` and `.login` (in that order) in the user's home directory. In either case, however, if the person logging in is not the owner of a start-up file, then it will not be executed.

sticky bit
If a program's sticky bit is set, its text portion will remain in the system swap area permanently (or until the next reboot). For often-executed programs, setting the sticky bit can improve performance. If a directory's sticky bit is set, then to remove or rename a file within the directory you must have write permission on the directory and be the owner of the file or the parent directory (or be the superuser). Often the sticky bit is set on `/tmp` and `/usr/tmp` to prevent users from removing files that do not belong to them.

subdirectory
A subdirectory is any directory that is located within another directory. For example, the directory `/usr/spool` is a subdirectory of the directory `/usr`.

superuser
A superuser is a user who has absolute privileges on the IBM/4.3 system. The superuser can run any command, enter any directory, kill any process, and remove or alter any file in the system. Generally you will be logged in as the superuser (`root`) only while performing system administration activities, since you will need to be able to traverse all parts of the file system and manipulate many files within it.

swap
To swap is to move a process from main memory into secondary storage, or *vice versa*. By swapping processes in and out, a computer system is able to manage processes that collectively require more memory than is available on the system.

symbolic link
A symbolic link is a special kind of file that contains an arbitrary path name. When the kernel encounters a symbolic link in a path name when searching for a file, it interprets the remainder of the file name relative to the file name contained in the symbolic link. The two files linked in this way need not be on the same file system. See also link.

system
The word 'system' can be used to designate any of the following items, listed from most specific to most general: (1) the kernel or operating system, `/vmunix`, which manages the entire computer and provides the essential connection between user-specified commands and the hardware they execute on; (2) the kernel together with the normal tools and utilities that altogether comprise the computer software; (3) the combination of software and computer hardware in a single machine; (4) a collection of machines into a network.

system administrator
The system administrator, or system manager, is responsible for keeping a computer system running smoothly and efficiently. This includes adding and removing peripheral devices, installing new software, and any other tasks related to the operation of the system. The system administrator is

often called a superuser, since that person has the ability to access any file or directory within the system.

tail

`Tail` is a program that lists the last few lines of one or more files.

TCP

TCP is a member of the Internet protocol suite that provides for reliable message transmission across a patcket-switching network. TCP packages messages into datagrams and tracks their delivery. 'TCP' stands for 'Transmission Control Protocol'. See also **IP**, **Ethernet**, and **datagram**.

termcap

`Termcap` is the name of a data base that describes terminals. Each entry in the data base includes a description of a terminal's capabilities and operations.

terminal

A terminal is a peripheral device used to communicate with a computer. Typically a terminal consists of a keyboard and a video display screen; a more elaborate terminal may have a mouse and/or a modem attached to it.

uid

A uid is an integer assigned to a particular user that the system uses to store information about the user. 'Uid' stands for 'user identification number'. See also **gid**.

unmount

To unmount a file system is to make it inaccessible, so that users cannot read from it nor write to it.

user

A user is a person who has an account on a computer.

UUCP

The UUCP system is a family of programs that allow file transfer from one machine to another and remote command execution. Historically UUCP has operated over dial-up telephone lines or direct-connect serial lines, although recently it has been expanded to allow communications between machines on a local area network. 'UUCP' stands for 'UNIX-to-UNIX Copy'.

working directory

Your working directory is the directory that you are currently located within. The command `pwd` lists the current working directory. You can change from one working directory to another using the `cd` command.

X-Windows

The X-Windows package is a TCP/IP-based window system capable of providing a window environment and graphics capability on bit-mapped terminals. The Andrew Toolkit is built on top of the X-windows system.

format

```
/etc/fd/format -h /dev/rfd0
```

save

```
tar -cvf /dev/rfd0 -C /full file name /...
```

dir

```
tar -tvf /dev/rfd0
```

This page intentionally left blank.

Index

210

IBM Academic Operating System 4.3
System Administration Guide
READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

If you wish, give your name, university or site, mailing address, and date:

____

____

____

____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments, or you may mail directly to the address in the Edition Notice on the back of the title page.)