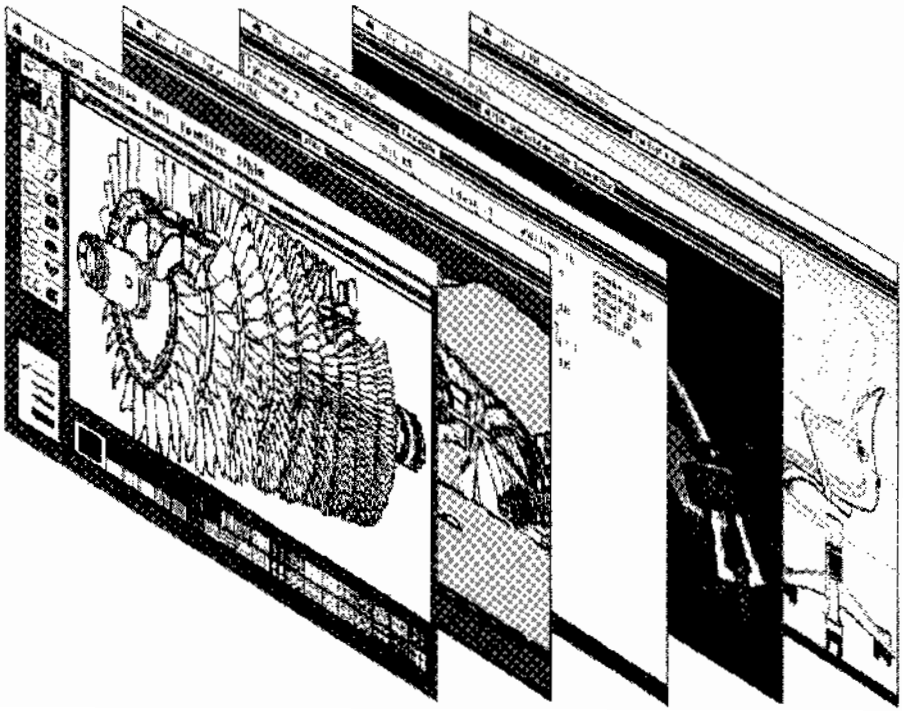
Apple® A/UX™ Local System Administration



Copyright

Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE AND ITS SOFTWARE SUPPLIER MAKE NO WARRANTIES OR REPRESENTATIONS, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO**

SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD AS IS, AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.

IN NO EVENT WILL APPLE OR ITS SOFTWARE SUPPLIER BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple and its software supplier shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

A/UX Local System Administration

Contents

Preface

Conventions Used in This Manual

Throughout the A/UX manuals, words that must be typed exactly as shown or that would actually appear on the screen are in `Courier` type. Words that you must replace with actual values appear in *italics* (for example, *user-name* might have an actual value of `joe`). Key names appear in CAPS (for example, RETURN). Special terms are in **bold** type when they are introduced; many of these terms are also defined in the glossary in the *A/UX System Overview*.

Syntax notation

All A/UX manuals use the following conventions to represent command syntax. A typical A/UX command has the form

`command` [*flag-option*] [*argument*]...

where:

`command`	Command name (the name of an executable file).
flag-option	One or more flag options. Historically, flag options have the form -[*opt*...] where *opt* is a letter representing an option. The form of flag options varies from program to program. Note that with respect to flag options, the notation `[-a][-b][-c]` means you can select one or more letters from the list enclosed in brackets. If you select more than one letter you use only one hyphen, for example, `-ab`.
argument	Represents an argument to the command, in this context usually a filename or symbols representing one or more filenames.

[]	Surround an optional item.
...	Follows an argument that may be repeated any number of times.
`Courier type`	anywhere in the syntax diagram indicates that characters must be typed literally as shown.
italics	for an argument name indicates that a value must be supplied for that argument.

Other conventions used in this manual are:

<CR>	indicates that the RETURN key must be pressed.
^*x*	An abbreviation for CONTROL-*x*, where *x* may be any key.
cmd(*sect*)	A cross-reference to an A/UX reference manual. *cmd* is the name of a command, program, or other facility, and *sect* is the section number where the entry resides. For example, `cat`(1).

Chapter 1
Managing the A/UX System
An Introduction

Contents

Chapter 1

Managing the A/UX System

An Introduction

1. The system administrator

This book is for the A/UX system administrator. The system administrator is responsible for making sure that the system runs smoothly and typically performs the following functions:

- starting up and shutting down the system
- adding and removing user accounts
- adding and managing peripheral devices
- maintaining data integrity
- backing up and recovering data
- monitoring system usage and activity
- setting up and maintaining communication channels with other computer systems

With the exception of communicating with other systems, all of these procedures affect a single computer and are considered **local system administration**. These procedures are described in this manual. For information on setting up and administering networks, see *A/UX Network System Administration.*

2. A/UX system administration manuals

The information you need for managing single (not networked) A/UX systems is presented in two manuals:

- *A/UX Local System Administration* (which you are currently reading) contains nine chapters. This chapter is an introduction and overview, and each of the remaining eight chapters is devoted to one of the categories of system administration described above. These chapters include step-by-step

instructions that guide you through routine procedures as well as background information about the A/UX environment. The background information is useful for analyzing unexpected situations or error conditions and should provide enough understanding to handle these cases. First-time administrators should read these chapters in sequence.

- The *A/UX System Administrator's Reference* is a reference for details and variations on the standard commands and procedures. It summarizes the main system administration commands, but does not contain introductory or overview material. It is recommended that you read *A/UX Local System Administration* first and use *A/UX System Administrator's Reference* for quick reference once you understand how a command works. The commands documented in *A/UX System Administrator's Reference* are cited as *command-name*(1M). You will also find references to commands cited as *command-name*(1). These are contained in *A/UX Command Reference*.

3. For the new system administrator

This book combines theory and practice to help you understand what to do as well as why. If you are not an experienced system administrator, note the following suggestions before you begin:

- Log in as the superuser (`root`, `rootcsh`, or `rootksh`) only when absolutely necessary. The superuser logins have special privileges, and you can perform functions as a superuser that you can't do if you are logged in as a normal user. Thus, as a normal user you have some protection against making mistakes that might affect the operation of your system. If you make mistakes as a superuser, you can crash (or damage) your system.
- While you're logged in as the superuser, record everything you do in a systems administrator's log. This log should contain an exact description of what you do, why you do it, the date and time, and the results of your actions.
- Make backup copies of all important system files before you change them. For example, if you are about to modify the

`/etc/inittab` file, first enter the command

```
cp /etc/inittab /etc/inittab.old
```

Then, if your modifications prove unworkable, it is an easy matter to return the system to its prior state:

```
mv /etc/inittab.old /etc/inittab
```

- Use `cron` to automate routine system administration duties or to remind yourself to do these duties. See Chapter 2, "System Startup and Shutdown," Chapter 7, "System Accounting Package," and Chapter 8, "System Activity Package." Also see `cron`(1M) in *A/UX System Administrator's Reference* and `crontab`(1) in *A/UX Command Reference*.
- Use `find` to locate large and dormant files and directories. Large, unused files are often not needed and consume valuable storage space. When you find such files, you can either remove them (for example, if they are core images of crashed programs), truncate them to zero length (if they are log files), or archive them onto tape or diskettes (if you are even remotely likely to want to use them again in the future). See `find`(1) in *A/UX Command Reference*.
- Watch for files that grow (in general, any file containing the letters `log` or `LOG` as part of its name). The system appends information to these files, and they can grow to excessive sizes. You should regularly delete these files after backing them up.
- Make frequent backups of your files. Backups are your insurance if something should go wrong. How often you perform backups depends upon how much activity there is on your system and how much work you're willing to lose. See Chapter 4, "Backing Up Your System."
- If you are administering a multiuser system, choose low-activity times to perform potentially disruptive tasks. Early morning and late evening are good choices. Check to see who is logged in before beginning; if anyone is actively running processes, notify them with the `wall` command. See `wall`(1M) in the *A/UX System Administrator's Reference.*

- If a catastrophe occurs,

    1. Think carefully about what happened.
    2. Plan your next action before actually doing it.
    3. Anticipate the consequences of implementing your plan.
    4. Act.
    5. Write down in your system log what happened, what you did and how the system responded.

    One of the worst things you can do if a system catastrophe occurs (such as a loss of important data) is to act without first considering the consequences of your actions.

4. Administrative logins on the A/UX system

The standard distribution of the A/UX system contains one normal user login account, `start`, whose home directory is `/users/start`. This login account is provided to give new users a place to store files used in the introductory tutorials; see *Getting Started With A/UX*.

All other logins on the standard distribution are **administrative logins**, login accounts that are used by system administrators or by A/UX programs to perform specialized system tasks. The administrative logins are as follows:

`root`
`rootcsh`
`rootksh`
: Logins for the superuser. These logins have identical permissions in the A/UX system; they differ only with regard to the startup program, which is, respectively, the Bourne shell, the C shell, and the Korn shell. The home directory for all three logins is the root directory, `/`.

`daemon`
: The owner of certain noninteractive, background processes that handle persistent system services such as network communication.

`bin` The owner of most normal A/UX commands and the system directories in which those commands are stored.

`sys` The group to which certain components of the A/UX system belong, such as swap space (`/dev/swap`), core memory (`/dev/mem`), and certain communications ports (for example, `/dev/ttyp1`). The existence of this group allows programs that need access to these critical system components to run setgid `sys` instead of setuid `root`, which is a much greater security risk.

`adm` The owner of most of the system accounting programs and directories, in particular `/usr/adm`. See Chapter 7, "System Accounting Package."

`uucp`
`nuucp`
`uucp` is the owner of programs and directories associated with the UUCP communications package. `nuucp` is the login assigned to incoming UUCP requests. See Chapter 9, "The UUCP System," for further information.

`lp` The owner of commands and processes associated with the `lp` spooling and printing package. See "Setting Up A Printer" in Chapter 5, "Managing Peripheral Devices," for more information on the `lp` system and the `lp` login.

`ftp` The owner of files and directories associated with `ftp`, a file transfer program. See *A/UX Communications User's Guide* for a description of the `ftp` command.

`nobody`
A login assigned as the default login for remote `root` access under NFS. See *A/UX Network System Administration* for further information.

`who` A login provided to allow users who do not have accounts on a system to see who is currently logged into the system. There is no password for this login. The startup program is simply `/bin/who`.

Chapter 2
System Startup and Shutdown

Contents

Chapter 2

System Startup and Shutdown

1. Introduction

This chapter describes how to start up and shut down the A/UX operating system on the Macintosh II. These procedures are largely automatic, but they do require supervision and possible intervention by the system administrator. These procedures may also be customized to suit local needs.

This chapter also briefly describes the stand-alone shell (`sash`(8)). The stand-alone shell is a Macintosh program that may be invoked before A/UX is booted. Like other shells, `sash` is a command interpreter with a command language and environment similar to (but simpler than) that of the Bourne shell, including stand-alone versions of shell variables, comments, input and output redirection, and a number of built-in commands. Your primary use of the stand-alone shell will be to boot the A/UX system (described in "The Booting Sequence"). If you need to run the autorecovery facility, this is also done from the stand-alone shell. See "The Stand-Alone Shell" and `sash`(8) in *A/UX System Administrator's Reference.*

2. Starting up the A/UX system

This section assumes that you have already followed the instructions for setting up your Macintosh II in the *A/UX Installation Guide*. When the set-up instructions have been completed, you are ready to start up (**boot**) the A/UX operating system.

2.1 The booting sequence

A power-on switch is located on the keyboard of the Macintosh II. The exact location on the keyboard varies among the different keyboard models, but on all keyboards the power-on switch is marked with a triangle pointing left. Press this switch to turn on your computer. After 3 to 5 seconds, the screen should light up, the `Welcome to Macintosh` message appears, and the stand-alone shell displays an

Automatic Boot screen.

The Automatic Boot screen will count down to zero, or you can press RETURN or click the mouse button once on the **GO** square to initiate the boot sequence for A/UX. When booting starts, the stand-alone shell causes A/UX to take control of the Macintosh II.

In a second or two, the screen blinks, A/UX copyright and Release ID information is displayed, and the following prompt appears:

```
Do you want to check the root file system? (y or n)
                                          [default: n]
```

Answer yes by typing

```
y
```

followed by RETURN. This invokes the `fsck` (file system checking) program. This is always the safe thing to do when starting up A/UX. `fsck` checks the A/UX root file system in several phases, each of which displays a message on your screen. If `fsck` finds any inconsistencies in the file system, it will either automatically fix them or prompt for your input (see Chapter 6, "Checking the A/UX File System: `fsck`").

If `fsck` has made any corrections to the file system, it automatically reboots A/UX. After booting, `fsck` asks again if you want to check the root file system. The safest thing to do is to repeat the `fsck` process (type `y` again at the prompt).

When the checking is completed, the following message should appear:

```
INIT: SINGLE USER MODE
```

followed by the message of the day and

```
TERM = (mac2)
```

Press RETURN after this prompt to set the terminal type to Macintosh II.

If you have already set your host name during the installation process (see *A/UX Installation Guide*), you will see the shell prompt that uses that host name. For example, if your system's name is `louie` you see the prompt:

```
louie.root#
```

2.2 The system's name

To assign a name to your system or to change the system's name, use a text editor to open the `/etc/HOSTNAME` file. This file should contain two fields separated by spaces or a tab. The name of your system is the word in the first field of this file. Change the first field and save the file. This change will take effect the next time you shut the system down and reboot. See "Shutting Down the A/UX System."

2.3 Single-user mode

Single-user mode is one of several **run levels** available on the A/UX system. The current run level determines which processes are running on the system. In **single-user mode,** all system functions are possible but only the console is enabled. All functions are run with root privilege, so the net effect is a single user (`root`) on a single terminal (the console). On systems that have more than one user, single-user mode is usually used for administrative functions that are best performed when no other users can be logged into the system.

2.4 Entering multiuser mode

The standard run level for A/UX is **multiuser mode.** This is recommended even if you are the only user on a machine. In multiuser mode, all ports designated in the `/etc/inittab` file are enabled and permit user logins (see "The `/etc/inittab` File" below). In starting multiuser mode, the system performs the commands stored in the file `/etc/rc`. In general, low-level processes such as daemons are started when the system enters multiuser mode.

If your system has one or just a few users, the two basic run levels, single-user and multiuser, will probably suffice for normal operation. If you have a system with many users and perhaps many modems as well, you may be interested in A/UX's more complex run-level capabilities, described below under "Changing Run Levels."

To enter multiuser mode from single-user mode, type

```
telinit 2
```

This prints:

```
INIT: New run level: 2
```

You are again asked if you want to check the file systems. Currently, the standard A/UX distribution supports only one user-accessible file system, which is root. If you have not added more file systems (for example, on an external hard disk or 3.5-inch disk) you can simply press RETURN at this prompt.

As the system is going to multiuser mode, commands in `/etc/inittab` and `/etc/rc` are executed (see "`/etc/inittab` Entries" later in this chapter), When this process is complete, you should again be greeted with the message of the day followed by the login banner:

```
Apple Computer, Inc. A/UX

login:
```

The system is now running in multiuser mode and you may log in. After you log in, you see:

```
TERM = (mac2)
```

Press RETURN to set your terminal type to the Macintosh II..

2.5 Message of the day

If you log in as `root` or `rootksh`, the `/etc/profile` system startup script executes several commands, including

```
cat /etc/motd
```

This prints a message contained in the file `/etc/motd` (for "message of the day"). In the standard A/UX distribution this file contains

```
**************************************************
*                                                *
*        W E L C O M E   T O  A / U X            *
*                                                *
**************************************************
```

You can change the contents of `/etc/motd` to print anything you like by editing this file with a text editor. The new text you enter will print in place of the `Welcome to A/UX` message next time you bring the

system down to single-user mode, reboot, or log in.

2.6 Initial processes: /etc/inittab

The first A/UX process to run after booting the system is called `init` ("initial process"). This process runs before you enter single-user mode and reads system data files and executes some default system commands.

The first command listed in `/etc/inittab` on A/UX is the `/etc/sysinitrc` shell program, which performs basic functions like setting the system's clock and running `/etc/fsck` before you see the single-user mode shell prompt.

The second line in `/etc/inittab` specifies the default initial run level:

```
is:s:initdefault: #First Init State
```

The run level in the `/etc/inittab` file is specified in the second field of each entry, in this case `s` for "single-user." Once the initial run level is determined, `init` processes only those `/etc/inittab` entries whose run-level field is the same as the run level currently in effect. When you enter multiuser mode by typing

```
telinit 2
```

you invoke the `init` process with a run level of 2. In this case, `init` reads `/etc/inittab` and invokes every command that specifies a run level of 2.

If you check the contents of `/etc/inittab` on your system, you will see the processes invoked by init, for example,

`/etc/bcheckrc`
: A startup script that prompts and sets the proper system time and date, and runs `fsck` on the remaining file systems.

`/etc/brc`
: A startup script that clears the mounted file system table.

`/etc/rc`
: A startup script that mounts the file systems (if applicable) and starts the error-logging and other daemons.

`/etc/getty`
: A process that enables login, modem, or printer ports on the

system. See Chapter 3, "User Administration," and Chapter 5, "Managing Peripheral Devices."

2.6.1 `/etc/inittab` entry format

Each line in `/etc/inittab` is an entry containing four fields separated by colons, optionally followed by a number sign (#) and a comment. The format of each entry is

id:*run-level*:*action*:*command*

where the fields are interpreted as follows:

id	unique entry identification
run-level	run level where the entry is processed
action	action to be taken with next field
command	command to be executed

The following two lines are in the `/etc/inittab` file distributed with the standard A/UX system:

```
co::respawn:/etc/getty console co_9600
...
00:2:off:/etc/getty tty0 at_9600
```

These lines are followed by comments, respectively

```
# Console port
...
# Port tty0 (modem), set to "respawn"
```

Any text following a number-sign (#) in `/etc/inittab` is a comment documenting the entry. These comments indicate that the first line above is the console port. Because the *run-level* field is empty, the console port is enabled at all run levels, including single-user. The second line is the modem port, TTY0. If the *action* field is changed from `off` to `respawn`, this port will be enabled in multi-user mode. This is explained in more detail in the discussion of the *action* field below.

The *id* field is an arbitrary identifier of one to four characters that makes the entry unique. Although the identifier is arbitrary, convention

dictates that the identifier for an entry that affects a port should be the corresponding port number. In the first line above, `co` refers to the console port.

The *run-level* field tells `init` whether to process the entry. The *run-level* field can be any number from `0` to `6` (`2`, for multiuser, is the most common) or the character `S` or `s` (for single user). If the *run-level* field is empty, `init` processes the entry at all run levels.

The *action* field specifies how to execute the *command* field if the entry's *run-level* field matches the system's run level. For a complete list of possible *action* field entries, see `inittab`(4) in *A/UX Programmer's Reference*. `respawn` and `off` are described here because these are the most important *action* fields for the system administrator.

`respawn`
: Specifies that this command should be run when the designated run level is entered, and if the command terminates for any reason, it should be run again. This ensures that the command runs at all times as long as the present run level matches the run level of the entry in question. If the *run-level* field of the entry is empty, the process runs at all times and all run levels.

`off`
: Specifies that if the command is running when the present run level is entered, it should be terminated. This ensures that the command does not run as long as the specified run level matches the *run-level* field of the entry in question.

The fourth field is the command to be executed. When the run level of each of the entries matches the default run level, the command named in the fourth field is executed. In the example above, the command `getty` would be executed for each entry, as modified by the *action* field.

2.7 Changing run levels: `telinit`

Once the system is started and running at the default run level, you can change the run level. Log in as the superuser and type

```
telinit run-level
```

where *run-level* is an argument to `telinit` that may be a value from `0` to `6`, `s` or `S`. See `telinit`(1M) in *A/UX System Administrator's Reference*.

When you enter this command, `telinit` (and, by extension, `init`) scans `/etc/inittab` again and activates those entries whose run-level value is the same as that of the new run level, leaving all other processes untouched.

For instance, if the following lines appear in `/etc/inittab`,

```
01:23:respawn:/etc/getty tty1 at_9600
02:2:respawn:/etc/getty tty2 at_9600
```

both ports `tty1` and `tty2` have a `getty` running to them when the system is at run level 2. But if you later type

```
telinit 3
```

the `getty` running to `tty2` is terminated, while the one running to `tty1` continues because it has both run level 2 and run level 3 specified.

The only way to modify how a process runs at a new run level is to specify it in `/etc/inittab`. In theory, you should specify every process that affects, for instance, a port, with reference to every single possible run level. In practice, you do that only for those processes that you may want to turn on and off for certain specific run levels. See Chapter 5, "Managing Peripherals" for examples.

If you introduce a change in `/etc/inittab`,

```
telinit Q
```

makes the change known to the system, without having to restart the system. This command forces `init` to reread `/etc/inittab`. However, some of the processes controlled by `init` spawn child processes, and these child processes are not terminated by the `telinit Q` command. You can terminate these processes individually by specifying their process ID number to the `kill` command, or shut the system down and reboot.

The general functioning of your system, then, is determined by the current run level. The current run level is determined at startup by the default entry in `/etc/inittab`, and it can be changed via the `telinit` command. The current run level determines which commands (in the *command* field of `/etc/inittab` entries) are activated.

When running `telinit`, you should be careful that you do not bring about any changes that would disrupt the activities of users currently logged in. For instance, `telinit` could disrupt an operating modem if a `getty` were to be started on that port. Know your run states and make sure that changing them will not disrupt user activity.

3. Shutting down the A/UX system

There are several ways to shut down the A/UX system.

- If you want to shut down all processes and reboot the system, bring the system down (using the `shutdown` program or shutting the system down manually) and then type

  ```
  reboot
  ```

  The `reboot` command starts the boot sequence over again.

- Similarly, to bring the system down and turn the machine off, bring the system down (using the `shutdown` program or shutting the system down manually) and then type

  ```
  powerdown
  ```

  The `powerdown` command turns off the power to the main unit of the Macintosh II. If you have an external hard disk, switch it off.

- If you are hung for some reason and unable to invoke a command, you can bring the machine down by pressing the power switch at the rear of the unit. This is not recommended because it can introduce inconsistencies in the file system. If you bring the system down this way, make sure to run `fsck` on the root file system when you boot up again (see Chapter 6, "Checking the A/UX File System: `fsck`".)

The basic steps in shutting down A/UX are as follows. These can be done by hand or by using the `shutdown` program.

1. Bring the system down to single-user mode (`telinit s`).
2. If you have multiple file systems (for example, if you have an external hard disk), unmount the file systems.
3. Terminate all processes (`killall`).
4. Flush the file system buffers (`sync`).

The aim of these procedures is to bring the system to an almost inactive status before shutting it down. No files should be open, no processes should be running in the background, and everything should be written to disk. Only then can you power down the system with no danger of corrupting the file system.

3.1 Using the `shutdown` program

The automated `shutdown` program automatically performs the steps described above for bringing down A/UX and warns all other users on a system that the system is going down. To use `shutdown`:

1. Log in as `root` (or `rootcsh` or `rootksh`) on the system console.
2. Type

   ```
   cd /
   shutdown
   ```

   The system responds by printing a banner and the date,

   ```
   SHUTDOWN PROGRAM
   Wed Oct 28 12:11:50 PST 1987
   ```

   followed by a shutdown delay (the interval the system will wait before beginning the file system shutdown):

   ```
   Do you wish to enter your own delay to shutdown?
   Default is 1 minute (y or n):
   ```

3. If a 1 minute delay is satisfactory, just press RETURN. If you want more time (for example, if there are still important processes running or if you need to notify other users that you

are shutting down the system) or less time (to proceed directly with the shutdown when there are no other users running processes and no remaining background jobs) type

```
y
```

In this case, `shutdown` prompts again:

```
Enter your delay in minutes (0-60)
(No warning messages will be sent if input is 0):
```

To proceed directly, type `0`. To delay the shutdown type another number, for example, for a 2 minute delay type

```
2
```

4. In this case, `shutdown` asks if want to send a message.

   ```
   Do you wish to enter your own message (y or n):
   ```

   To print the default message, type `n` (no). The following message is broadcast to all users:

   ```
   Broadcast message from root (console)
   Wed Oct 28 12:21:46 PST 1987
   SYSTEM BEING BROUGHT DOWN FOR SYSTEM MAINTENANCE

   Minutes to SHUTDOWN :2
   ```

   To enter your own message, type `y` (yes); `shutdown` will prompt you for a message. Enter whatever text you want to send and then send an *eof* (usually CONTROL-d).

5. After waiting the specified time interval minus 30 seconds, `shutdown` prints

   ```
   Broadcast message from root (console)
   Wed Oct 28 12:25:11 PST 1987
   SHUTDOWN IN 30 SECS
   LAST CHANCE TO LOGOUT
   ```

6. After 30 seconds, the `shutdown` program performs the following actions:

- stops all daemons and kills all remaining processes
- executes `sync` a number of times to flush the write buffers
- unmounts the file systems
- executes `sync` again
- executes the "`telinit s`" command to bring the system to single-user mode

As these actions occur, you will see messages such as

```
Mounted devices are:
/dev/dsk/c0d0s0 on / type 5.2 (rw)
Unmounting file systems
/dev/dsk/c0d0s0 on / type 5.2 (rw)

****    SYSCON CHANGED TO /dev/console    ****
Killing all processes

INIT: New run level: S

INIT: SINGLE USER MODE
```

This is followed by the message of the day and

```
TERM = (mac2)
```

Press RETURN to set your terminal type to Macintosh II.

The system is now running in single-user mode and the `shutdown` program has exited. To reboot, type

```
reboot
```

This starts the boot sequence over again.

To turn off the Macintosh II type

```
powerdown
```

In this case, switch off any external disks.

3.2 Shutting down the A/UX system manually

To shut down the system by hand:

1. Log in as superuser (`root`, `rootcsh`, or `rootksh`).
2. If necessary, move to the root directory.
3. If necessary, warn your users you're shutting down the system by entering the command

   ```
   wall
   ```
   enter your message here
   eof (usually CONTROL-d)
4. Wait a few minutes, then change to single-user mode with the command

   ```
   telinit s
   ```

   The following message displays:

   ```
   ****     SYSCON CHANGED TO /dev/console     ****
   ```

   and the shell prompt returns. However, you should not type anything at this point. Wait until the following lines have displayed on the screen:

   ```
   INIT: New run level: s
   INIT: SINGLE USER MODE
   ```

   followed by the message of the day and

   ```
   TERM = (mac2)
   ```

   Press RETURN to set your terminal type to Macintosh II.
5. Terminate all remaining processes by entering the command

   ```
   killall
   ```
6. Update the disk to include changes made in the buffer cache with the command line

   ```
   sync;sync;sync
   ```
7. If you have an external hard disk or 3.5-inch disk with a mounted file system, enter the command

   `umount` *file-system*

where *file-system* is the name of any file system other than the root file system. If the unmounted file system is on a 3.5-inch disk, eject the diskette by typing

```
eject
```

as necessary.

8. Update the root file system with any final changes made in the buffer cache with the command line

```
sync;sync;sync
```

Then, to reboot, type

```
reboot
```

This starts the boot sequence over again.

To turn off the Macintosh II type

```
powerdown
```

In this case, switch off any external disks.

4. The stand-alone shell

The stand-alone shell (`sash`) is a command interpreter with a command language similar to the A/UX shells, including shell variables, comments, input and output redirection, and built-in commands. Programs run from `sash` can be read from either a Macintosh or an A/UX file system.

To invoke `sash`, reboot the Macintosh II as described under "Shutting Down the A/UX System." When the Automatic Boot screen displays and begins counting down to zero, click the mouse button once on the **CANCEL** square or press the keystroke sequence COMMAND-SHIFT-. (Command-Shift-period). This places you in the stand-alone shell window with the shell prompt:

```
sash#
```

You can change the prompt to any other string you like by redefining the shell variable `PS1`, for example, if you want the sash prompt to be `hello:` , type

```
PS1='hello:
```

As noted, it is possible to perform some limited interaction with the A/UX file systems from the stand-alone shell. To list the contents of the A/UX root directory, for instance, you could type

```
ls -CF /
```

Similarly, the command

```
ls -CF /users/start
```

lists the contents of the `/users/start` directory.

You should understand that the usual A/UX `ls` command is not being invoked here, but a special stand-alone version that is functionally identical to the A/UX command. Currently there are stand-alone versions of the following A/UX commands:

`cat`	`chgrp`	`chmod`
`cp`	`date`	`dd`
`ed`	`esch`	`fsck`
`kconfig`	`launch`	`ln`
`mkdir`	`mkfs`	`mknod`
`od`	`pname`	`rm`

To boot A/UX from the stand-alone shell, type

```
boot
```

For more information on the stand-alone shell, see `sash`(8) in *A/UX System Administrator's Reference*.

Chapter 3
User Administration

Contents

Figures

Chapter 3

User Administration

1. The user's working environment

Within the A/UX system, each user has a working environment that is analogous to an office worker's work space. Each user's working environment provides the following features:

- A secure place to work: When users want to work with the A/UX system, they must first enter their login names and secret passwords. After logging in, they are automatically located in their own home directories. This permits a private environment where each user has control over who has access to his or her work.
- The ability to share tools and data: The A/UX system also provides a mechanism by which users can share their work with other users. This is done mainly through the formation of groups of users with common tasks. Through the prudent use of groups, environments can be set up to permit the appropriate mix of security and sharing of resources.
- The ability to customize: Users can, in most cases, modify many features of their working environment by making changes to specific files located in their home directories. This gives them the power to personalize their environment, just as workers can set up their own work space in an office. Although users often modify their own working environment, the system administrator is responsible for the initial setup and certain kinds of modifications.

This chapter describes procedures and concepts that assist you in establishing, modifying, and removing users' working environments.

1.1 Components of a user's environment

There are a number of factors that determine the interpretation and operation of commands given by a user logged into an A/UX system. This manual refers to the collection of these factors as the "user's working environment" or more simply as the "environment"; this is a broader use of this term than that defined, for instance, in `environ`(5).

The following are the components of a user's working environment:

- Permissions: Every file in the A/UX system has permissions (or modes) associated with it that determine who can do what with the file. There are three kinds of actions that can be done to files and three corresponding types of permissions: write, read, and execute. These three types of permissions can be set differently for three types of user: the owner of the file, users belonging to the same group as the owner of the file, and other users, regardless of group membership. Users can execute programs or have access to data files only if their user IDs or group IDs are set appropriately for each program and data file. See "File Permissions," "Directory Permissions and Group Membership," and "System File Permissions."
- Login name and password: Before users can gain access to the system, they must enter their login name and password. These are defined in the `/etc/passwd` file; see "Files That Determine a User's Environment."
- User ID: Each user login name has a unique user ID (**uid**) that identifies the user. This is defined in the `/etc/passwd` file; see "Files That Determine a User's Environment." When a user attempts to use a command on some data, this number is compared to the user ID associated with both the command and the data file. If there is a match in either case, the files are checked to see how the permissions are set for the owner of the file. If there is no match for either file, the user's group ID is compared to the group ID associated with the files.
- Group ID: Each user login name has at least one group ID (**gid**) associated with it. This is defined in the `/etc/passwd` file; see

"Files That Determine a User's Environment." The group ID numbers indicate the groups to which the user belongs at the time of login. Group membership is a security feature that permits some users access to files, while denying this access to other users. This inclusion and exclusion can be total or partial. For example, suppose that there is a group called `acctg` and another group called `legal` and that all of the accounting files have the following permissions:

```
-rw-rw-r--
```

Members of the `acctg` group have read and write permission on these files, while all others have only read permission. Those in the `acctg` group are able to read and write each others' files, while those in `legal` are able to read the files that belong to `acctg` but not to write or execute them.

- Home directory: Each login name has a home directory associated with it; this is defined in the `/etc/passwd` file (see "Files That Determine a User's Environment"). When users log into the system, their startup programs use the **home directory** as the home base for creating and using files. In addition, the user can modify special files residing in this directory to tailor his or her environment further.

- Current directory: This is the directory where the user is located at the present time. Every time the user changes to a new directory, the new directory becomes the current directory. When a user specifies a **relative filename** (a filename that does not start with a /), the current directory is searched for a file of that name.

- Startup program: After logging in, the user needs some way of communicating with the A/UX system. The startup program is the program that automatically greets the user after a successful login. This is defined in the `/etc/passwd` file; see "Files That Determine a User's Environment." This variable is usually defined as the Bourne shell, C shell, or Korn shell. If it is not specifically assigned in the `/etc/passwd` file, the default startup program is the Bourne shell. You can use any of the

three A/UX command interpreters as the startup program. Sometimes the startup program is a more restricted and restrictive program than the A/UX shells. See "Changing a User's Startup Program" for more information.

- Other environment variables: The user's shell provides a number of environment variables that can be assigned different values to alter the environment. Some environment variables are automatically assigned values from the `/etc/passwd` entry for each user; these include `LOGNAME` (login name), `HOME` (home directory), and `SHELL` (startup program). Other variables are assigned values at the shell prompt or in files such as `.profile` in each user's home directory. One of these variables that affects the user's working environment is the `PATH` variable. The value assigned to a user's `PATH` variable determines which directories, and in what order, the shell will search for the file corresponding to a command issued by the user. For other environment variables, see *A/UX User Interface*.

1.2 Files that determine a user's environment

The user's environment is normally established at login time from information stored in a number of system files. See "How A/UX Establishes the Environment" for a description of the order in which this is done. This section describes each of these files along with its format and content.

1.2.1 The `/etc/passwd` file

A user's working environment is specified largely by a single entry in the `/etc/passwd` file. You must be logged in as `root` to modify this file.

The `/etc/passwd` file distributed with A/UX has several administrative logins and a user login named `start`. See "Administrative Logins on the A/UX System" in Chapter 1 for a description of the administrative logins.

Each entry consists of one line with seven fields separated by colons (`:`). The form of an entry is

login-name`:`*password*`:`*uid*`:`*gid*`:`*misc*`:`*home-directory*`:`*startup-program*

where the fields are interpreted as follows:

login-name
The name the user must use when logging in. It must be unique in the `/etc/passwd` file.

password
An encrypted version of the actual password the user must use when logging in. The encryption is done automatically when the password is first assigned and whenever it is changed. Having the password encrypted makes it possible to have the `/etc/passwd` file open for reading by everybody.

uid A unique user ID number for each user.

gid The user's default group membership. This means that if the user is not listed in the `/etc/group` file (see below, "The `/etc/group` File"), the user belongs by default to the group whose number appears in the user's *gid* field in the `/etc/passwd` file.

misc Miscellaneous information about the user, such as full name, address, and telephone number. For other uses of the *miscellaneous* field, as well as for the exact specifications of the *password* field, see `passwd`(1) in *A/UX Command Reference* and `passwd`(4) in *A/UX Programmer's Reference*.

home-directory
The name of an existing directory whose permissions, ownership, and group membership are such that the user can have access to it. Whenever the user logs into the system, this is the directory where he or she is initially located.

startup-program
The name of an executable program, usually one of the A/UX shells, that permits the user to communicate with the A/UX system.

The following is a typical `/etc/passwd` entry:

```
joe:AxhlmzGfp:56:100:Joe Doe:/users/joe:/bin/sh
```

1.2.2 The /etc/group file

The *gid* field of a user's entry in the /etc/passwd file establishes a single default group for the user. The /etc/group file is used to establish multiple group memberships for a user.

The /etc/group file contains entries with four fields separated by colons. The form of each entry is

group-name : *password* : *gid* : *list*

where the fields are interpreted as follows:

group-name
The group name. Group names are arbitrary, but the convention is that they should have a self-evident meaning (for instance, acctg rather than xyz24).

password
An encrypted version of the group's password. Although the *password* field can be set for each group, it is common practice to disable group password checking by filling the *password* field with the words VOID or NONE, or with asterisks (*). The reason for thus disabling password checking (as would otherwise occur, for example, with the newgrp command) is that in practice people feel freer giving away group passwords than individual passwords. Disabling the *password* field prevents this common security breach.

gid An arbitrary number closely associated with the group name. For each group ID there is only one group name, and vice versa. The actual group ID numbers that exist in the /etc/group file are the only numbers that should be entered in the *gid* field of /etc/passwd entries. The group ID entered for each user in /etc/passwd should coincide with the group ownership of that user's home directory.

list A list of the login names of the members of the group. The login names must be separated by commas. Entering a user's login name in the group's *list* field is optional for those users who belong to only one group. In this case, their group membership is determined by the value assigned in the *gid* field of their

/etc/passwd entry.

The following is a typical /etc/group partial listing:

```
acctg:*****:101:anne,john,peter
legal:****:102:rose,fred,anne,john,peter
```

By convention, group names are arbitrary though self-evident. The groups themselves, though, should not be arbitrary. As system administrator, you should divide users into groups according to their assigned activities, and you should allow users to belong to different groups only when there is some overlapping of activities. If a user belongs to too many groups, you should ask yourself whether the group distribution you have established is the best one.

As an example of the use of the /etc/group file to establish multiple group memberships for a user, look again at the partial listing of an /etc/group file above. You will notice that Anne is listed both in acctg and in legal. If the group ownership of her home directory is acctg, then every file she creates and every directory she makes below her home directory will also belong to acctg. For various reasons, Anne might want some of the files she creates to belong to the group legal. There are three ways she can accomplish this:

- She can create a file having group ownership acctg and then change its group membership to legal after the fact.
- She can make a directory having group ownership acctg and then change the group membership of the directory to legal; every file created within that directory will also belong to legal.
- She can change her effective group identification number for the duration of her login session (or until she changes it again).

In the first two cases, a file's or a directory's group membership can be changed with the command chgrp. For example, if Anne creates a file named suits in her home directory (group membership acctg), a long listing of the file (ls -l) will show

```
-rw-rw-r--  1 anne  acctg   19900 Oct  3 17:51 suits
```

To change the file's group membership, Anne has to enter

```
chgrp legal suits
```

After this command, a long listing would show the new group:

```
-rw-rw-r--  1 anne  legal   19900 Oct  3 17:51 suits
```

Note that the file permissions are not changed by the `chgrp` command, but now they apply relative to the group to which the file belongs.

To change her group identification for the duration of her login session, Anne can run the `newgrp` command. To move into the `legal` group, she would type

```
newgrp legal
```

All directories and files she creates after running this command will now belong to the group `legal`. For complete details, see `newgrp`(1).

1.2.3 The `profile` startup scripts

When a user logs in and has the Bourne shell or Korn shell specified as the startup program in `/etc/passwd`, the system automatically runs several shell scripts before giving the user a prompt. One of these scripts is the file `/etc/profile`. Like each `$HOME/.profile` file, this is a script of shell commands, which typically exports certain shell variables and sets a file creation mask. This file is readable by all users, but cannot be modified by normal users.

The `/etc/profile` file, if it exists, is run *before* the file `.profile` in the user's home directory and thus serves as a default `.profile`. A user can override any actions performed in the execution of `/etc/profile` by including the appropriate commands in his or her own `.profile`.

1.2.4 The `kshrc` startup script

When a user logs in and has the Korn shell specified as the startup program in `/etc/passwd`, the system runs the `/etc/profile` and `$HOME/.profile` files described above. If one of these files sets the `ENV` variable to any filename (`$HOME/.kshrc` in the standard A/UX distribution), the system also reads the contents of the named file.

1.2.5 The `cshrc` and `login` startup scripts

When a user logs in and has the C shell specified as the startup program in `/etc/passwd`, the system automatically runs several shell scripts before giving the user a prompt. One of these scripts is the file `/etc/cshrc`. Like each `~/.cshrc` file, this is a script of shell commands, which typically exports certain shell variables and sets a file creation mask. This file is readable by all users, but cannot be modified by normal users.

The `/etc/cshrc` file, if it exists, is run *before* the file `.cshrc` in the user's home directory and thus serves as a default `.cshrc`. A user can override any actions performed in the execution of `/etc/cshrc` by including the appropriate commands in his or her own `.cshrc`.

The `.login` script is run *after* `.cshrc`. It is typically used to set up terminal defaults and environment variables.

1.3 How A/UX establishes the environment

A close look at a successful login will help you understand how all the elements mentioned to this point interact in determining a user's environment.

1. Upon a successful login, the `login` program reads the user's *uid*, *gid*, and *home-directory* fields in the `/etc/passwd` file. Next it invokes the `initgroups` program, which reads each line of the `/etc/group` file, looking for a match between the user's login name and the *login-name* field in this file. For each match, `initgroups` assigns to the user the corresponding group specified in the *gid* field. The `login` program then executes the command named as the user's startup program in the *command* field of the `/etc/passwd` entry. This command inherits the user's user ID, group ID(s), and home directory from the `login` process.
2. Usually, the startup program is a shell and its initial invocation is known as the **login shell**. Login shells look for and (if it exists) read a startup file in the directory `/etc`. This file is known as the **default startup file**. If the startup program is the Bourne shell or the Korn shell, this file is called `/etc/profile`; if it is the C shell, the file is called `/etc/cshrc`. In this file, the

system administrator can modify certain aspects of all the users' working environments, such as PATH, HOME, TERM, and EXINIT, as well as set other features such as aliases in the C shell and functions in the Bourne shell. See sh(1), ksh(1), and csh(1) in *A/UX Command Reference*.

3. After reading the default startup file, the shell looks for and (if it exists) reads an initializing file in the directory named in the *home-directory* field of the /etc/passwd file. If the startup program is the Bourne shell or the Korn shell, this file is called .profile; if it is the C shell, there are actually two files, .cshrc and .login. In this file, the user can modify certain aspects of his or her working environment, such as PATH, HOME, TERM, and EXINIT, as well as set other features such as aliases in the C shell and functions in the Bourne shell.
4. Once the startup environment has been thus established, the startup program prompts the user for input.

2. File permissions

Permissions determine who can and cannot have access to and use a particular file or directory. For example, if a file has a read-write permission for all users on the system, anyone on the system can get into the file, and permanently change the file. If a file has read-only permission for all users on the system, all users can read the file, but no one can make any changes to the file.

2.1 File access permissions

A user can set the following permissions on the files he or she owns:

r Read permission. Allows designated users to read a file or to copy its contents.

w Write permission. Allows designated users to modify a file.

x Execute permission. Allows designated users to execute a file (that is, to run it as a command).

These permissions can be set for users in three classifications called **access classes.** An access class is a category of user which may be *user*, *group*, or *other*. Each access class is represented by three

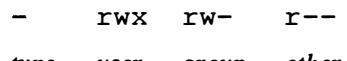
characters; the order of characters is `r`, `w`, `x`, indicating the access permission granted to that category of user. If a hyphen appears instead of an `r`, `w`, or `x`, permission to perform that action is denied to that category of user.

Ten characters are used to represent a file and its permissions, the first character indicates the type of file and the next nine characters indicate the permissions of the three access classes as shown in Figure 3-1.

The file access permissions described below would appear on the screen as

```
-rwxrw-r--
```

Figure 3-1. Access classes



type The first character represents the file type. This is not an access class, but is an essential part of file permissions. In Figure 3-1, the file is a regular file (represented with -). Other file types are `d` (directory), `c` (character device), `b` (block device), or `p` (FIFO or named pipe).

user The next three characters represent the file access permission of the owner of the file. In Figure 3-1, the letters are `rwx`. The owner has permission to read, write, or execute the file. User is represented by `u` when used with `chmod` and `chgrp`. See "Symbolic Terms" below.

group The next three characters represent the group's permissions. In Figure 3-1, the letters are `rw`. Any user in the same group as the owner of the file has permission to read and write (change) the file. Permission to execute the file is left as a hyphen, meaning that the file is not executable by the group.

Group is represented by `g` when used with `chmod` and `chgrp`. See "Symbolic Terms" below.

other The last three characters represent the permissions for all other system users. In Figure 3-1, only `r` appears, meaning that others (who do not belong to the group) can only read the contents of the file. Other is represented by `o` when used with `chmod` and `chgrp`. See "Symbolic Terms" below.

Permissions have to be set for the three access classes for each file. They can be modified manually at any time, or they can be set automatically to be the same every time a file is created.

2.2 Modifying a file's permissions

Only the owner of a file, or the superuser, can change that file's permissions. This is also called changing a file's mode. The command used for this is `chmod`. Anything that `chmod` can do to a file's permission applies also to a directory's permissions, since a directory is treated by A/UX equally as a file. See "Directory Permissions and Group Membership."

2.2.1 Symbolic terms

`chmod` can be invoked with either symbolic or numeric terms. Symbolic terms are straightforward: `u` stands for user (that is, owner) of the file, `g` stands for group, and `o` stands for others; + represents granting permission and – represents removing permission.

The format for invoking `chmod` with symbolic terms is

`chmod` *access-classoperatorpermission filename-list*

(There are no spaces separating the *access-class*, *operator*, and *permission*.)

The arguments are:

access-class
One or more of the three access classes described under "File Access Permissions" above: *user* (`u`), *group* (`g`), or *other* (`o`).

operator
Grant access permission (the + operator) or deny it (the – operator). You can't both grant and deny permissions in a single

command. You must grant permissions to one access class in one command, then deny it to another access class in a second command.

permissions
: Read permission (`r`), write permission (`w`), and execute permission (`x`). You can grant (or deny) more than one type of permission at the same time, but you can't grant *and* deny permission at the same time.

filename-list
: The file(s) whose permissions are to be changed. You may use absolute or relative pathnames.

To change the permissions of a file from

```
-rw-rw-rwx
```

to

```
-rwxrw-r--
```

the sequence of commands is as follows.

1. To grant execute permission to the user:

   `chmod u+x` *filename*

2. And then, to deny write and execute permissions to all others:

   `chmod o-wx` *filename*

2.2.2 Numeric terms

Numeric or absolute terms are based on the combinations allowed by octal numbers where, for each access class, the mode of the file is set as follows:

0	grants no permission
1	grants execute permission
2	grants write permission
4	grants read permission

These numbers can in turn be combined in the following way:

3 (1 + 2) grants execute and write permission
5 (1 + 4) grants execute and write permission
6 (2 + 4) grants read and write permission
7 (1 + 2 + 4) grants all permissions

The format for invoking `chmod` with numeric terms is

`chmod` *permission filename-list*

where *permission* is a composite of the three access classes.

If the permissions on a file are

```
-rw-rw-rw-
```

this has a numeric representation of

```
666
```

To make the file readable, writeable, and executable by the owner and group using numeric terms, use the command

`chmod 770` *filename*

where the first `7` represents `rwx` for the user, the second `7` represents `rwx` for the group, and the `0` represents no access permission for all others. The permissions of the file would then be

```
-rwxrwx---
```

There is another field that can also be set through `chmod`. The meaning of the numbers corresponding to this fourth field is

1 set sticky bit (not applicable to A/UX)
2 set group ID on execution
4 set user ID on execution

In swapping systems, the sticky bit indicates that the file should remain in main memory once it has been loaded in; this can speed up initialization time for frequently used programs at the cost of indefinitely tying up a portion of main memory. Since A/UX is a paging system, however, the sticky bit has no effect. For additional information see Chapter 8, "System Activity Package." Note that neither set user ID or set group ID modes apply to directories or other nonexecutable files.

When a file is executable by people other than the owner and has the user ID set, the person executing the file assumes the user ID of the owner of the file. This user ID becomes the "effective user ID" (and the "effective group ID"). So if the file is owned by `root`, is executable by others, and has the user ID set, whoever executes the file assumes the identity of `root`, and has all the privileges of `root` for the duration of the file's execution.

This is very useful with very specific and restricted programs, such as `passwd`, which permits any user to modify the file `/etc/passwd` and yet allows the system to prohibit writing on that file at any other time through the way its permissions are set. On the other hand, setting the user ID on general-use programs can be dangerous from the point of view of security. For example, a program such as `vi` permits the spawning of new shells, which would give the effective user ID the permissions of the real user ID in subshells. Thus, you should never set the user ID on such programs.

When a file is executable by people belonging to groups other than the group to which the file belongs, and the group ID is set in the file, the person executing the file assumes the group ID of the file during the execution of the file. This is particularly useful for administrative purposes, when group belonging rather than login identity is needed, and it does not pose as many potential security problems as setting the user ID on a file.

To set the group ID on a file with read, write, and execute permissions for all (mode `777`), the `chmod` command is

```
chmod 2777 filename
```

To set the user ID on a file with read, write, and execute permissions for the owner, read and execute permissions for the group, and no permissions for all others (mode `750`), the `chmod` command is

```
chmod 4750 filename
```

The permissions field in the output of the `ls -l` command in the first case would be

```
-rwxrwsrwx
```

where the `s` in the group execution field represents the group ID set.

The permissions field in the ouptut of the `ls -l` command in the second case would be

```
-rwsr-x---
```

where the `s` in the owner execution field represents the user ID set.

User ID set and group ID set can be combined, as all other numeric terms, so that

`chmod 6755` *filename*

will result in

```
-rwsr-sr-x
```

The "sticky bit" can be set on a file only by `root` with, for instance, the following invocation of `chmod`:

`chmod 1750` *filename*

If the file is already executable, this will result in

```
-rwxr-x--T
```

2.3 `umask` and file permissions

The environment variable `umask` defines the permissions for each file created by a user. This variable can be set for all users by the system administrator in the `/etc/profile` or `/etc/cshrc` files, or it can be set individually for each user in his or her `.profile` or `.login` or `.cshrc` file. See "How A/UX Establishes the Environment." The value assigned to `umask` in the individual files `.profile`, `.login`, or `.cshrc` overrides the values set in `/etc/profile` or `/etc/cshrc`.

`umask` can be assigned a numeric value of three octal numbers, like the permissions associated with `chmod`, and the value of each specified digit is substracted from the corresponding digit specified by the system for the creation of files.

For example, to ensure that all files created by a user have the permissions

```
-rw-rw-rw-
```

the `umask` for that user must be set as

```
umask 111
```

so that when the `111` is substracted from `777`, the files' permissions are `666`.

Similarly, to ensure that all files created by a user have the permissions

```
-rw-r-----
```

the `umask` for that user must be set as

```
umask 137
```

so that when `137` is substracted from `777`, the files' permissions are `640`.

The notation

```
umask 77
```

is shorthand for

```
umask 077
```

that is, leading zeros can be eliminated from the notation.

Note that changing a user's `umask` does not affect the permissions on existing files.

3. Directory permissions and group membership

A directory is a file containing a list of filenames and an index number for each filename. This index number points (indirectly) to the location(s) on disk where the file's contents are stored; see Chapter 6, "Checking the A/UX File System: `fsck`." Commands that manipulate or create files, like `ls`, `rm`, `mv`, `cp`, `ln`, `rmdir` and `mkdir`, perform their work by reading and/or writing a directory's contents. For example, `rm` deletes a directory entry's index number so that the file can no longer be used; thus, permission to remove a file has to do with the modes of the directory it's in, not those of the file itself. The `ls` command simply reads the contents of the directory to find all filenames, while the `cd` command, pathname evaluations, and path

searches work by searching the directories. Thus, directory permissions work a little differently from file permissions and have the following meanings:

`r` Filenames can be read from the directory.

`w` Directory entries can be added or deleted.

`x` The directory can be searched or made the current directory.

When you set permissions on a directory, you define who may list that directory's contents, add or delete files in that directory, or move into that directory. Note that setting permissions on a directory affects only the directory itself and does not change the permissions of any of the directory's files or subdirectories.

Directory permissions are one of the most important and neglected aspects of the user's environment. For example, file permissions that protect against reading and/or writing by other users are not enough to protect the files from being *deleted* if the directory permissions allow write permissions to other users. Similarly, if the directory has group read permission, its files can be listed by a group member even if the files themselves have the group read permission turned off.

Group membership is an important consideration for directory permissions. The group membership of a file or directory is the same as the group membership of the directory in which the file is created. This allows for the creation of hierarchies of directories grouped according to their group membership.

4. Adding a user

Adding a user to your system is a process with two distinct steps: planning the new user's working environment and then specifying it. These steps are equally important. Skipping the planning stage can lead to a very inefficient use of the system.

4.1 Planning the user's working environment

If at this point you do not have an actual user to add to your system but want to practice, you can use the examples listed below. If you are about to add a real user, follow the steps below but provide your own specifications.

Before you begin, you should have a clear idea of who the user is, what his or her tasks will be, what group or groups are currently engaged in similar activities, what parts of the system you want the user to have access to, whether a new group should be created, and where in the system the new user should be located.

In other words, the new user should belong to a group whose members have similar tasks (accounting, legal, programming, documentation, and so on) or to more than one group if the user will have a variety of tasks.

Follow these steps in planning a user's working environment:

1. Keep a hard-copy record of data about the new user. The record should include information like that listed in the form below. This form will make adding a new user's working environment easier and will be useful for your own record keeping.

   User's real full name _____

   Date (year/month/day) _____

   User's telephone number _____

   User's login name _____

   User identification number _____

   Group identification number _____

   Full pathname of the user's home directory

   _____

   Full pathname of the user's startup program

   _____

2. Pick a login name for the user. Login names normally consist of lowercase alphabetic characters only. Make sure that the new user's login name is a new name with the command

   `fgrep` ***login-name*** `/etc/passwd`

   This command searches the `/etc/passwd` file to see if there is

already a user with the login name you have chosen. If a line appears starting with the login name you have chosen, pick another login name and invoke `fgrep` again with the new name. If the shell prompt returns without any line appearing, no one is using that login name. Enter the new login name on your hard-copy record. If you are practicing, enter the name `dummy`.

3. Before selecting a new user identification number, you must find one that is not being used. One method for selecting the lowest unused number is to enter the command

   ```
   cut -f3 -d: /etc/passwd | sort -nr
   ```

   This displays the current user ID numbers in the `/etc/passwd` file. Pick a number that is not being used, and write this number in the space labeled "User identification number" on the form in step 1.

4. Select a group identification number. If you are practicing, use 100; otherwise, see "The `/etc/group` File" for information about selecting and specifying group membership. Write the number in the space labeled "User's group identification number."

5. Select a home directory. Use */fs/login-name*, where *fs* is the file system where you are going to put the new user and *login-name* is the user's login name on the form. If you are practicing, use `/users/dummy`. Think carefully. You may want to use a pathname like

   /fs/gm/login-name

   where *gm* represents a directory above the user's home directory. This *gm* directory should have the same group membership as the user's home directory, but the user should not have write permission on it. All users belonging to the same group should then have their home directories at the same level, that is, under *gm*. This way, the owner of the *gm* directory can be the group's manager. Who should be the group manager is also a matter for thought. Once you have decided, write down the full pathname in the home directory space on the form.

6. Select a startup program, such as `/bin/sh` or `/bin/csh`. If you are practicing, choose `/bin/sh`. Otherwise, you may also ask for the user's preference. Write down the full pathname in the startup program space on the form. See "Changing a User's Startup Program" for information about using different command interpreters as a user's startup program.

4.2 Specifying a user's working environment

Now that you have made your choices and have all the information written down, you can proceed with the practical steps involved in adding the user.

1. If you are not already superuser, log in as `root`.
2. Make a copy of `/etc/passwd`. For instance,

   ```
   cp /etc/passwd /etc/passwd.old
   ```

   This copy is your backup in case you accidentally destroy this critical file.
3. Next, use the `vipw` command to edit the `/etc/passwd` file. You should have all the pieces of information in front of you.

   Note: You should have all the pieces of information in front of you. The `/etc/passwd` file is set as "read-only." The `vipw` editor copies the contents of the password file into a temporary file (`/etc/ptmp`). After you edit and write the file, it copies the changes back to the `/etc/passwd` file.

   See `vipw`(1M) in *A/UX System Administrator's Reference* for more information about editing `/etc/passwd` using `vipw`.
4. Enter the following line as the last line in the file, replacing each italicized word with the new user's information from the form you just completed. *Be careful while you modify this file.* It is central to your users' and your own ability to gain access to the system.

   login-name::*uid*:*gid*::*home-directory*:*startup-program*

Leave the second field blank for now. It will be filled by an encrypted version of the user's password in a few moments. The fifth field, which is also blank in the line above, is for any miscellaneous information you care to enter (for example, the user's real name if it differs from the login name, phone number and address, and so on). Remember to use full pathnames for the user's home directory and startup program. If you want to play it safe, use the following line:

```
dummy::50:100:nice guy:/users/dummy:/bin/sh
```

5. Write the file and quit the editor.
6. Now enter the command

   `passwd` *login-name*

   where *login-name* is the name you filled in on the form in the slot labeled "User's login name." You are asked to enter the new user's secret password. The `passwd` program asks you to enter the password twice. If you do not type the same password two times, it asks you to try again. If the word is too short (fewer than six characters), does not contain at least one nonalphabetic character, or is the same as the user's login name, it asks you to enter a different word (see `passwd`(1)). Write the password down and give it only to the new user.
7. Create the user's home directory, using the full pathname from the form, with the command

   `mkdir` *home-directory*

   If you are practicing, type

   ```
   mkdir /users/dummy
   ```
8. If you have a standard `.profile` file, give the user a copy of it with the command

   `cp` *profile-standard home-directory*`/.profile`

   Replace the words *profile-standard* and *home-directory* with the name of the standard `.profile` file and the user's home directory as entered on the form. Note that the A/UX standard

distribution supplies basic copies of the login and environment files needed for each of the A/UX shells. These are located in `/usr/adm`.

If a user has the Bourne shell (`sh`(1)) as his or her login shell, you can use the command:

```
cp /usr/adm/sh.profile home-directory/.profile
```

If a user has the Korn shell (`ksh`(1)) as his or her login shell, use the commands:

```
cp /usr/adm/ksh.profile home-directory/.profile
cp /usr/adm/ksh.kshrc home-directory/.kshrc
```

If a user has the C shell (`csh`(1)) as his or her login shell, use the commands:

```
cp /usr/adm/csh.login home-directory/.login
cp /usr/adm/csh.cshrc home-directory/.cshrc
```

If you are practicing, use the `.profile` file from the `start` account (used with *Getting Started With A/UX*) with the command

```
cp /users/start/.profile /users/start/dummy
```

9. Now you can change the ownership of the user's home directory, mail file, and login or environment file(s). Again, replace each of the italicized words with the information on the form. Enter the commands

   `chown` *login-name home-directory*
   `chown` *login-name home-directory/login-files*

   where *login-files* are the files you copied from `/usr/adm`.

   If you are practicing, change the ownership as follows:

   ```
   chown dummy /users/dummy
   chown dummy /users/dummy/.profile
   ```

10. Now change the group membership of the user's home directory with the command

```
chgrp group-name home-directory
```

where *group-name* is the name (as listed in `/etc/group`) of the group ID specified in the *gid* field of the user's entry in the `/etc/passwd` file.

If you are practicing, type

```
chgrp project /users/dummy
```

11. Now change the permissions attached to the user's home directory and `.profile` file with the commands

```
chmod 740 home-directory
chmod 640 home-directory/.profile
```

where 740 means the user has write, read, and execution permissions, members of the group have read permission, and other users have no permissions; and 640 means the user has write and read permission, members of the group have read permission, and other users have no permissions.

If you're practicing, change the permissions with the commands

```
chmod 740 /users/dummy
chmod 640 /users/dummy/.profile
```

For more information about the command lines in steps 8, 9, and 10, see "File Permissions" in this chapter and `chown`(1) and `chmod`(1) in *A/UX Command Reference*.

12. Now log out and log back in using the new user's login name and password. Create a new file in the working environment you just established. If you have any trouble, see "Troubleshooting."

5. Modifying a user's working environment

A/UX provides for great flexibility in establishing and modifying a user's working environment. The following are several of the more important parameters that can be modified:

- a particular user's ability to have access to the commands and data stored on the system

- the accessibility of a user's files and directories to other users
- the location or name of any user's home directory
- the command that is used by the user as the shell

5.1 System file permissions

The system administrator can change the permissions of system command and data files so that no users, some users, or all users can have access to them. This is a responsibility that should be exercised with extreme caution, because giving write permission to all users on a file like `/etc/passwd` can have disastrous consequences.

Users can change the permissions associated with their own files. For a discussion on how to do this and what effects these changes have on users' ability to have access to the files, see "File Permissions," "Directory Permissions and Group Membership," and `chmod`(1) in *A/UX Command Reference*.

5.2 Moving a user

Sometimes it is necessary to move a user's working environment. There are a few ways of doing this, and the method you choose depends on the characteristics of the move. If you do move a user's files, remember to change his or her home directory in `/etc/passwd`.

5.2.1 Moving a directory

The simplest move is the one that involves moving a user's directory to another place in the same file system. The command line

`mv` *old-dir new-dir*

moves the *old-dir* directory (including all its files, any subdirectories associated with it, and all of their files) to *new-dir*.

5.2.2 Moving a user across file systems with `tar`

While `mv` works only within the current file system, the `tar` command can be used to copy directories from one file system to another.

Note: In the standard A/UX distribution on a Macintosh II with an 80 Mbyte hard disk, the disk contains only one user-accessible file system. The entire A/UX directory hierarchy and

any specific hierarchy (such as `/usr`) is available on the root file system.

If you have created a new file system (for example, named `/users2`) on an external hard disk or diskette, you can copy all files (except for dot files) and subdirectories contained in the directory `/users/john` to a directory `/users2/john` on the other file system with the command

```
tar cf - /users/john | (cd /users2/john; tar xf -)
```

The parts of this command line are as follows:

`tar` Command name.

`c` Creates a new `tar` image.

`f` Uses the following filename to store the image.

`-` When used with `f`, directs the image to the standard output.

`/users/john`
Name of the directory to start copying from.

`|` Connects standard output of the previous command to standard input of the next command.

`(...)`
Parentheses enclose commands to be executed in a subshell.

`cd /users2/john`
Changes the subshell's current directory to `/users2/john`.

`;` Command separator.

`tar` Command name.

`x` Extracts file(s) from the just-created `tar` image on tape or disk. `tar` does so file by file; if the file is a directory, it will be extracted recursively (that is, until it is exhausted of files and subdirectories).

`f` Extracts the image from the following file.

- When used with `f`, takes the image from the standard input. When - stands for a filename, `tar` uses the standard output as a file with the `x` or `t` options.

This moves a copy of the user's files to a new directory; the original files can be deleted once you are sure the move was successful.

Please keep in mind that this example shows how to move the user and is not a lesson on `tar`; see `tar`(1) in *A/UX Command Reference*. Remember that when you move the user's files you should also change the user's *home-directory* field in `/etc/passwd` and any other references to his or her home directory in files like `.profile`.

5.2.3 Moving a user across file systems with `cpio`

With `cpio`, a directory containing files and subdirectories can be copied elsewhere on the system, with all files maintaining their original ownership, permissions, and modification time.

> *Note:* In the standard A/UX distribution on a Macintosh II with an 80 Mbyte hard disk, the disk contains only one user-accessible file system. The entire A/UX directory hierarchy and any specific hierarchy (such as `/usr`) is available on the root file system.

If you have created a new file system (for example, named `/users2`) on an external hard disk or diskette, you can copy all files (except for dot files) and subdirectories contained in the directory `/users/john` to a directory `/users2/john` on the other file system with the command

```
find /users/john -depth -print | cpio -pdm /users2/john
```

The parts of this command line are as follows:

`find`
: Name of the command that gathers the filenames to pass to `cpio`.

`/users/john`
: Name of the directory to start the search from.

`-depth`
: Forces a depth-first search of the directory, to control the order in which files are copied.

`-print`
: Prints each file or directory name found.

`|`
: Connects standard output of the previous command to standard input of the next command.

`cpio`
: Name of the command that does the actual copying.

`-`
: Character signaling that options follow.

`p`
: Copies (''passes'') the named files to a named directory.

`d`
: Creates new subdirectories as needed.

`m`
: Retains the original file's modification times.

`/users2/john`
: New directory created by `cpio`.

This moves a copy of the user's files to a new directory; the original files can be deleted once you are sure the move was successful.

Please keep in mind that this example shows how to move the user and is not a lesson on `cpio`; see `cpio`(1) in *A/UX Command Reference*. Remember also that when you move the user's files you should also change the user's *home-directory* field in `/etc/passwd` and any other references to his or her home directory in files like `.profile`.

5.3 Changing a user's startup program

The last field in the `/etc/passwd` file determines a user's startup program. Typically the field is `/bin/sh`, `/bin/ksh`, or `/bin/csh` (for the Bourne shell, the C shell, or the Korn shell, respectively). Changing this field is all that is needed to change a user's startup program. Other modifcations to the user's working environment may be necessary, however, particularly with regard to shell startup files in the user's home directory; see ''Files That Determine a User's Environment.''

Any program at all can serve as the startup program. For instance, the last field of the `/etc/passwd` file can be a program like `who`. A user who has this startup program is able to log in, but can only see the output of the program `who` before being logged out, without ever getting a shell. Although `who` is not a very useful working environment, other programs such as `rsh`(1) are. `rsh` allows a user full use of a shell within the home directory but allows no directory changes.

6. Removing a user

Removing a user from your system may be as simple as inserting a word such as `VOID` in the encrypted password field for that user in `/etc/passwd`. However, if the user has created many files that must be saved, you may need to find all files owned by the user, back them up, examine each of them, determine who else uses the files, change the ownership of shared files, remove links, and finally delete the user's password entry.

This section introduces the most moderate form of user removal first and then discusses additional steps that make the removal more extreme.

6.1 Gentle deletion

The first step in removing a user from your system is to deny the user access to it. The cleanest way to do this is to edit the user's `/etc/passwd` entry and enter the word `VOID` in the encrypted *password* field. This makes it impossible for anyone to log in as that user (although that user's files remain unaffected).

> *Note:* Do not leave the *password* field blank. A blank password is a serious security breach because anybody can then log into the system with that login name and no password.

Do not yet delete the whole `/etc/passwd` entry for that user. If you do, you will not only deny the user access to the system, but will also affect the files owned by that user. Commands that use login names as arguments (for example, `chown` and `find`) or print information relating to login names (for example, `ls -l`) check the

`/etc/passwd` file for the user names and/or numbers. If there is no login name for a file's owner, it is replaced by a number (when you enter `ls -l`, for instance). If you delete a few `/etc/passwd` entries, you will probably get confused about which files belong to which ex-user.

6.2 Backup and selective deletion

Deleting a user's files should be done with care. In general, it is a good idea to back up a user's files before deleting them, for two reasons:

- These files may contain information that you will later want access to.
- These files may be used by other users on your system.

To locate all the files owned by the user that are not below his or her home directory, follow these steps:

1. Void the user's password; see "Gentle Deletion."
2. Find all the files belonging to the user, regardless of their location, with the command

   `find / -user` *login-name* `-print >` *somefile*

3. Back up the files using either `tar` or `cpio`; see the information on partial backups in Chapter 4.
4. Delete the user's files, but before you do so, it is a good idea to find out if anyone is currently executing any commands or using any data files owned by that user. Inquire personally or through `mail`, or use the `acctcom` command (see Chapter 7, "System Accounting Package") to see whether any users make regular use of files created by that user. If they do, change the ownership of those files. If a file is linked to that user, remove the link. Then delete the files.

7. Troubleshooting

Most user administration problems can be traced back to ownership and group membership questions or to erroneous entries in the `/etc/passwd` and `/etc/group` files.

The following are error messages and other potential problems, with suggestions for responding.

`No home directory`

If you see this error message at login time, check that the entry for that user in the `/etc/passwd` file is correct. Typically, if the *home-directory* field is empty but surrounded by colons, the message at login time is

`unable to change directory to "`*program*`"`

where *program* stands for the entry in the last field of the `/etc/passwd` file. No login will occur. If the *home-directory* field is missing (that is, if the *startup-program* field appears directly after the *gid* field), then the message is

`unable to change directory to ""`

Can't create files

Check that the home directory, or the current directory, has the appropriate permissions, ownership, and group membership.

Can't list a directory

The user is probably in a directory whose group membership is not one of the user's current groups.

Can't read mail

Check that the file `/usr/mail/`*login-name*, where *login-name* stands for the user's login name, has the appropriate permissions, ownership, and group membership.

Files owned by other user

It is very important to make sure that no two users have the same user ID in the third field of their `/etc/passwd` entries. If this happens, some programs get confused.

One such program is `ls`. A directory does not contain the name of the user to which a specific file belongs; it only contains the user ID. For `ls` to be able to print the user name when it is invoked with the `-l` flag option, it has to go to the `/etc/passwd` file to associate that user ID with a login name. Two users with the same user ID unavoidably confuse `ls`.

Chapter 4
Backing Up Your System

Contents

Tables

Chapter 4

Backing Up Your System

1. Overview

Making regular backup copies of files and file systems is one of the most important duties of the A/UX system administrator. Computer data that is stored on disk can be damaged by hardware failure, or users may accidentally remove it. If you make regular backups, you can restore the data if it becomes damaged or destroyed. When data is properly backed up, all files, directories, and file systems can be restored completely.

When you make a backup, data that is stored on hard disk is copied to an alternate medium such as a 3.5-inch disk. You can use these backup copies to recreate a system in the event of a system crash or to recreate data that may be damaged or lost.

It is a good idea to store backups in a safe place, off-site if necessary. It is also wise to maintain a backup log to give you a written record of what was done, for use in an emergency.

There are many ways to back up data. To decide on the best technique, you should compare the time it takes to complete a backup with the time it may take to restore a backup. You should also consider how often your data is changed, how valuable the data is, and how many people use the system. The safest scheme is to devise an overlapping strategy, combining two or three backup techniques. Generally, if you use a regular schedule for full backups and supplement those with one or two partial backups, you can be assured that you will be able to rebuild your system in the event of a disaster. You may want to customize backup commands in a shell script to keep all backups consistent.

You may also use the A/UX backup utilities to archive directories no longer needed on the system. Storing unused data on 3.5-inch disks frees the system disk for use and improves performance.

The backup and restore utilities available on the A/UX system include `cpio`, `tar`, and the `dumpfs` and `restore` utilities (`dumpfs` and `restore` are mutually dependent and you must use them together). This chapter describes how to use these programs.

1.1 Backup media

Backups can be done on a variety of media. The A/UX system currently supports 3.5-inch disks. In the future, other media will be available for the A/UX system.

To install a diskette disk in the built-in drive, hold the diskette with the arrow facing up and slide it into the drive in the direction of the arrow. To remove the disk, type the command

```
eject
```

and remove the diskette from the drive.

1.2 Preparing diskettes for use

In order to copy files to a 3.5-inch disk, the disk must be initialized; otherwise you'll get error messages. The initialization processing is called **formatting.** To format a 3.5-inch disk, place the disk into the disk drive (or into the right-most drive if there are two 3.5-inch drives on your system). Then type

```
diskformat /dev/rfloppy0
```

You will be asked to confirm your request to format the disk; press RETURN to start formatting or type an *interrupt* to abort the operation. When the shell prompt returns, your disk is formatted.

> *Note:* Under normal use, you only need to format a disk once. If A/UX prints messages about read and/or write errors while using backup utilities, you can try formatting the disk again. If the error messages persist, use a new disk. Note also that formatting erases any information that was previously stored on the disk.

1.3 Device special files

Most of the backup utilities you will be using need to be told where to find the files you want to back up and where to record the backup. In

A/UX all peripherals (hard disk drives, floppy disk drives, terminals, and so on) are known to the system through **device special files** located in the `/dev` directory. When a backup utility reads or writes to such a file, the result is that information is read (or written) to the peripheral corresponding to the special file. ("Device" is a synonym for peripheral.)

Device-special files come in two varieties: block devices and character devices (also known as "raw" devices). Whether you access the device as a block device or a raw device depends on the requirements of the utility you are using. The table below lists the names of the device special files that are standard for the Macintosh II.

Table 4-1. Standard Macintosh peripherals

Device Special File Name	Peripheral	Type
`/dev/floppy0`	floppy drive #0	block
`/dev/rfloppy0`	floppy drive #0	raw
`/dev/floppy1`	optional floppy drive #1	block
`/dev/rfloppy1`	optional floppy drive #1	raw
`/dev/dsk/c8d0s0`	synonym: floppy drive #0	block
`/dev/rdsk/c8d0s0`	synonym: floppy drive #0	raw
`/dev/dsk/c0d0s0`	built-in hard disk	block
`/dev/rdsk/c0d0s0`	built-in hard disk	raw

Note: A user (especially `root`) should never attempt to write data using the name of a device special file that accesses a disk drive. These filenames should only be used on backup utility command lines. Also note that all examples in this chapter use the floppy drive #0, although floppy drive #1 (if this option is present) would work as well.

As shown in Table 4-1, you can use the synonym `/dev/dsk/c8d0s0` for floppy drive #0. This uses the SCSI naming scheme in which devices are named in the `/dev/dsk` and

/dev/rdsk directories according to their controller, drive, and slice number. For example, /dev/dsk/c8d0s0 is SCSI ID 8, the first drive, and the first partition. This is the name for floppy drive #0. An external hard disk installed, for example, in slot 5 would be /dev/dsk/c5d0s0. See gd(7) in *A/UX System Administrator's Reference* for more information on the SCSI naming scheme.

1.4 Full versus partial backups

There are two main strategies for creating backups. The first is a full backup, where all the data on each file system is copied; it is a time-consuming process. Full backups copy a system in such a way that you can reload it if your disk is completely erased.

The second strategy is a partial backup, where only part of the system is backed up. This is less time consuming and more useful for most types of everyday work. Depending on which utility you use, you may do either a selective or an incremental partial backup. A **selective backup** copies the files or directories you specify by category such as user, group, or specific need. Generally, tar or cpio is used for backing up specific data. An **incremental backup** uses the modification dates on files to automatically copy all newly created files or files modified since the date of the last backup. Although tar and cpio can both be used for incremental backups, the dumpfs and restore utilities are generally preferred. Incremental backups save the most recently modified files in the system and will routinely back up files that are changed often, such as /etc/passwd and /etc/groups. One of the main problems with incremental backups is that it can take a lot of time to locate a file and restore it if you don't know the file's last modification date.

2. Backing up a small amount of data

If you want to make a backup copy of a small amount of data, such as a single file or hierarchy, you can

1. Insert a 3.5-inch disk in the floppy drive.
2. Format the 3.5-inch disk and create an empty file system on it.
3. Create a directory and mount the new file system on it.

4. Copy your files to the new file system.
5. Unmount the file system and eject the 3.5-inch disk.

This procedure is described below. For more information about file systems, see Chapter 6, "Checking the A/UX File System: `fsck`." For more information about the `mount` command, see `mount`(1M) in *A/UX System Administrator's Reference.*

2.1 Mounted versus unmounted file systems

A file system exists on a logical portion of a physical device. This logical portion is called a **partition** and is accessed through the device special files explained in the previous section. A file system that is accessible to users is called a **mounted file system**. With the exception of the root file system, which is always mounted, file systems must be explicitly mounted and unmounted with the `mount` and `umount` commands.

> *Note:* Currently the only user-accessible file system on the built-in hard disk is the root file system, which is permanently mounted.

A file system is made accessible by **mounting** it on an empty directory of any other mounted file system. This directory can be any ordinary A/UX directory and is called the **mount point** of the file system. When you change your current directory to the mount point, you may traverse the directory tree of this file system as if it were an ordinary branch of the root file system.

If you want to create a file system on a 3.5-inch diskette, install the diskette in the drive and enter the following command sequence:

```
diskformat /dev/rfloppy0
mkfs /dev/rfloppy0 1600
```

To use this file system, create an empty directory on the root file system (we suggest a name like `/f0`) with the `mkdir` command and then mount the file system with the `mount` command. The command sequence is

```
mkdir /f0
mount /dev/floppy0 /f0
```

The subsequent files and directories which you create in `/f0` will appear as ordinary files and directories even though they reside on the 3.5-inch disk drive.

To remove the file system diskette, enter the commands

```
umount /dev/floppy0
eject
```

> **Warning:** Removing the diskette before issuing the `umount` command can damage the file system because A/UX may still have file system data stored in memory.

The error message

```
/f0: Device busy
```

means you or other users are accessing files or directories on the file system mounted on `/f0`. If you are in single-user mode, simply change your current directory to a directory which is not on this file system and reenter the `umount` command. If other users are accessing this file system, ask them to finish their work and change their current directory to another directory so you can unmount the file system.

2.2 Backup utilities

The A/UX backup utilities fall into two categories, archival and copy utilities. This chapter describes **archival utilities**, which copy data and reference information pertinent to the data. Archival utilities are designed specifically to move files and directories between media; the reference information is used to reconstruct the original data structure.

The A/UX **copy utilities** copy data only; they operate on a single file at a time and disregard any structure that that file may have (for instance, the file may be an entire file system). For information on these programs, see `dd`(1) in *A/UX Command Reference* and `volcopy`(1M) in *A/UX System Administrator's Reference*.

The A/UX archival backup utilities include `cpio`, `tar`, `dumpfs`, and `restore`:

`cpio`
Stands for "copy input to output." Used to back up and restore an entire file system or individual files. While copying files, `cpio` also archives everything passing through it.

`tar`
Stands for "tape archiver." Used to archive a directory or individual files within a file system. The `tar` command is the best utility to use for transferring files from one system to another, for example, transferring ASCII files between UNIX Version 7 systems and System III- or System V-based systems (such as A/UX).

`dumpfs`
Used for incremental or full file system backups. `dumpfs` supports "dump levels"; these levels range from 0 to 9, where 0 represents a full backup of the entire system and the other numbers are used for incremental backups.

`restore`
Recovers files and directories from a backup medium created with `dumpfs`. Single files can be recovered from the backup medium by using the `-x` option for the `restore` program (see "Using `restore`"). When using `restore`, you must be careful not to replace the current file system with an older version of itself.

With the exception of archives made with `tar` and `cpio` (using the `-c` option), A/UX backups are not transferable among different computers; that is, a backup made with a `dumpfs` program from a Macintosh II cannot be restored with the `restore` command on a VAX.

Note: You should generally perform backups (and all major administrative tasks) while the system is running in single-user mode. If you make a backup of a mounted file system that is being altered by frequent writes, you risk backing up an outdated and inconsistent file system.

2.3 A basic overlapping backup scheme

This section shows a sample overlapping backup scheme that is commonly used on A/UX systems. With this scheme, you can recycle backup media when you have reinforced the nightly backup with a weekly backup, when you've reinforced the weekly backup with a monthly backup, and so forth. You can use any of the above utilities in this scheme. Remember that 3.5-inch disks, streaming tapes, and certain other backup media can be used only a finite number of times; check the manufacturer's specifications to determine how many times you can safely recycle your backup media.

Quarterly backups

Make a full system backup when you first start using the system and again at the end of each quarter. You can either store the quarterly backups off site or create duplicate backup media, storing one set on site and another set off site. Quarterly backup media should be stored for an indefinite period of time. *Be sure to verify that full backups are readable.* See "Verifying data on backed up disks" in this chapter.

Monthly backups

Make an incremental backup at the end of each month. You should also store these backups in a safe place for 3 to 6 months, depending upon your site's needs and how valuable your data is. Once you have a monthly backup, you no longer need the previous month's weekly backups, so you can recycle those media. Be sure to verify that new backups are readable before recycling media from old backups.

Weekly backups

Make an incremental backup at the end of each week to capture all files modified or created during the week. Store these backups in a safe place, and reuse them for the monthly backup at the end of the current month. Once you have a weekly backup, you no longer need the previous week's daily backups, so you can recycle those media. Be sure to verify that new backups are readable before recycling media from old backups.

Nightly backups

Make an incremental backup at the end of each weekday to copy all

newly created files or modified files. Set aside one formatted backup medium (or more if needed) for each night. Then, at the beginning of each new week, copy over the previous week's backup media; on Monday, copy over the previous Monday's backup; on Tuesday, copy over the previous Tuesday's; and so forth.

3. Using `cpio`

The `cpio` utility copies files to or from the device or location you specify. There are two ways to use `cpio`: as a command or as an option to the `find` command. When you use it as a command, `cpio` takes input from commands such as `find` or `ls` to determine the pathnames of files to copy and then copies the files to wherever you direct (for example, to a disk or a file).

`-o` Copies out the files to a location you specify (disk, file, and so on).

`-i` Extracts files from a location you specify (disk, file, and so on).

Note: When you're using `cpio` as a command, it can stop when a backup medium is full and wait for you to change disks.

When you use `cpio` as an option to the `find` command, the `find` command locates the files you want to back up. You don't need to specify filenames or pathnames as arguments to `cpio`. However, when you're using this method, there is no way to pass flag options to `cpio`.

Note: When using `cpio` as an option to the `find` command, all data must fit onto one backup volume. If you are not sure whether the files you want to back up will fit on a single volume, you should not use the `-cpio` option to the `find` command.

Examples for creating selective backups, full backups, and incremental backups using `cpio` and `find` are presented below.

3.1 Copying files in a directory to a disk

To recursively copy all the files and directories in a directory to a 3.5-inch disk (assuming that they will fit on a single disk), enter the command:

```
find directory-name -cpio device
```

The components of these commands are as follows:

`find`
: The `find` command using a directory name recursively locates the files and directories in the specified directory.

`-cpio`
: The `cpio` option to `find` indicates that the files will be written in `cpio` format, that is, 5120-byte records. Remember that when you use the `cpio` option to the `find` command, there is no way to pass flag options to `cpio` and all data must fit onto one disk.

device
: The output medium, for example, the disk `/dev/rfloppy0`.

3.2 Creating selective backups

To create selective backups, where only files that fall into a specified category are backed up, type

```
find directory-name -user user-name\
-print | cpio -ovB > /dev/rfloppy0
```

(The above command line has been "folded" onto two lines by preceding the newline with a backslash; this is not necessary to the command.)

The components of these commands are as follows:

`find` *directory-name* `-user` *user-name* `-print`
: The `find` command searches the specified directory name to obtain the files of the specified user name and passes the pathnames of the user's files to `cpio`.

`| cpio -ovB`
: The `cpio` command, with options to copy out files (`o`), display

the name of every file copied on the screen (v), and block output (B) 5,120 bytes per record.

`> /dev/rfloppy0`
The output medium, in this example the (raw) disk.

You can use other options to `find` that select files by other characteristics, such as group ownership or age of the file. See "Creating Incremental Backups" later in this chapter and `find`(1) in *A/UX Command Reference*.

3.3 Creating full backups

To create a full backup of your entire system, type

```
find / -print | cpio -ovB > /dev/rfloppy0
```

The components of these commands are as follows:

`find / -print`
The `find` command, beginning at the root directory (/), passes the file pathnames to `cpio`.

`| cpio -ovB`
The `cpio` command, with options that copy out files (o), display the name of every file copied on the screen (v), and block output (B) 5,120 bytes per record.

`> /dev/rfloppy0`
The output medium, in this example the disk.

If the diskette becomes full but more data still needs to be copied, the following message appears:

```
End of file mark found during write
of output to standard output.
When ready, type (special) filename,
return (default), or 'exit' to quit.
```

To respond to this message, insert the next disk, type the special filename `/dev/rfloppy0`, and press RETURN.

3.4 Creating incremental backups

To create incremental backups, where only files that have been modified or created since a certain time are backed up, type

```
find / -mtime -1 \
-print | cpio -ovB > /dev/rfloppy0
```

(The above command line has been "folded" onto two lines by preceding the newline with a backslash; this is not necessary to the command.)

The components of these commands are as follows:

`find /`
: The `find` command, beginning at the root directory (`/`), passes the file pathnames to `cpio`.

`-mtime -1 -print`
: Select files modified (`-mtime`) since the last day (`-1`) and pass the file pathnames to `cpio`. The `-1` is used for daily incremental backups, and `-7` is used for weekly incremental backups.

`| cpio -ovB`
: The `cpio` command, with options that copy out files (`o`), print the name every file copied on the screen (`v`), and block output (`B`) 5,120 bytes per record.

`> /dev/rfloppy0`
: The output medium, in this example the disk.

3.5 Creating a table of contents for a disk

To list the table of contents for a 3.5-inch disk made with `cpio`, type

```
cpio -it < /dev/rfloppy0
```

The components of these commands are as follows:

`cpio -it`
: The `cpio` command with the `-i` and `-t` flag options. The `-i` flag option specifies that input filenames should be extracted, and the `-t` option generates a table of contents.

`< /dev/dsk/c8d0s0`
: Use the files on the disk as input to the `cpio` command.

3.6 Recovering all files on a disk

To recover all files from a disk created with `cpio`, type

```
cpio -ivdmu < /dev/rfloppy0
```

The components of these commands are as follows:

`cpio -ivdmu`
: The `cpio` command with flag options. The `i` option extracts files from the 3.5-inch disk, the `d` option creates any directories needed to extract the files, the `v` option prints the file name on the screen after it has been extracted, the `m` option preserves the file's modification date, and the `u` option extracts files from the archive unconditionally. (Normally, `cpio` will not extract a file that is older than an existing file with the same pathname.) See `cpio`(1) in *A/UX Command Reference* for additional information.

`< /dev/rfloppy0`
: Use the files on the disk as input to the `cpio` command.

3.7 Recovering selected files from a disk

To recover only certain files from a disk created with `cpio`, first obtain a list of the full pathnames for files on the disk, using `cpio` with the `-i` and `-t` options:

```
cpio -it < /dev/rfloppy0
```

Now select the files you want to extract, and type

```
cpio -ivdmu filename... < /dev/rfloppy0
```

The components of these commands are as follows:

`cpio -ivdmu`
: The `cpio` command with flag options. The `i` option extracts files from the 3.5-inch disk, the `d` option creates any directories needed to extract the files, the `v` option prints the file name on the screen after it has been extracted, the `m` option preserves the file's modification date, and the `u` option extracts files from the archive unconditionally. (Normally, `cpio` will not extract a file that is older than an existing file with the same pathname.) See `cpio`(1) in *A/UX Command Reference* for additional

information.

filename
The name of the file(s) to be extracted.

Note: Because `cpio` restores files *relative to the current directory*, the *filename* you specify must be a pathname relative to the current directory, for example,

```
users/jeffb/file
```

or

```
./users/jeffb/file
```

Never use an initial '`/`' in any `cpio` file specification.

You can use file expansion characters such as `*` and `?`, but these characters must be quoted (enclosed in single or double quotes or preceded by a backslash) to prevent the shell from interpreting them before they are passed to `cpio`.

`< /dev/dsk/c8d0s0`
Use the files on the disk as input to the `cpio` command.

3.8 In the event of hard I/O errors

When you're checking data after you have created backups, if a **hard I/O error** occurs while reading the data, the entire process has to be redone. There is a shortcut, however. Restart `cpio` using the defective disk, and do not change disks until the volume number of the defective disk is reached. Then insert a fresh disk and continue as usual.

`cpio` cannot be interrupted to allow formatting for additional disks. The process must be aborted, fresh disks formatted, and the procedure started again. However, the shortcut for a defective disk can be used in this instance, too.

4. Using `tar`

The `tar` command copies a single file or groups of files to or from a disk. The files are copied sequentially, and no directory structure is created. Files can also be added to files already existing on the disk

Files copied with `tar` can be extracted from the disk with `tar` and are replaced in the directory from which they were originally copied. The `tar` command also displays a table of contents for the files archived on a particular disk.

`tar` uses key arguments to control its actions. Options are described below, and examples illustrate how to use them.

4.1 Copying an entire directory to a disk

One particularly good use for the `tar` command is to archive on disk directories no longer needed on the system. Storing unused data on backup media frees disk space for use.

To copy all files in the `/usr/lib` directory, including subdirectories and their contents, type

```
tar cf /dev/rfloppy0 /usr/lib
```

The components of these commands are as follows:

`tar cf /dev/rfloppy0`
: The `tar` command with option `cf`. The `c` option creates a new backup, writing at the beginning of the disk. The `f` option uses the argument `/dev/rfloppy0` to archive the files.

`/usr/lib`
: The full pathname for the files to be copied in the `/usr/lib` directory.

4.2 Copying specific files

Often, there is a need to save files with important data in them. Such files include `/etc/passwd` and `/etc/group`. Because these files are crucial to the operation of the system, and because they are frequently modified, they are vulnerable to corruption or even loss.

To copy `/etc/passwd` and `/etc/group`, type

```
tar cf /dev/rfloppy0 /etc/passwd /etc/group
```

The components of these commands are as follows:

`tar cf /dev/rfloppy0`
: The `tar` command with option `cf`. The `c` option creates a new backup, writing at the beginning of the disk. The `f` option uses

the argument `/dev/rfloppy0` to archive the files.

`/etc/passwd /etc/group`
Copy the files `/etc/passwd` and `/etc/group`.

4.3 Adding a file to files already on a disk

To add a copy of the file `mvusr` from the current directory (`./`) after the existing files on the disk, type

```
tar rf /dev/rfloppy0 ./mvusr
```

The components of these commands are as follows:

`tar rf /dev/rfloppy0`
The `tar` command with options `rf`. The `r` option writes the file to the disk. The `f` option uses the argument `/dev/rfloppy0` to archive the files.

`./mvusr`
Write the contents of `mvusr` file from the current directory.

4.4 Adding a later version of a file to a disk

To add a later version of the file `curses.mail` from the current directory (`./`) to a disk, type

```
tar uvf /dev/rfloppy0 ./curses.mail
```

The components of these commands are as follows:

`tar uvf /dev/rfloppy0`
The `tar` command with options `uvf`. The `u` option adds the named files to the disk if they are not there or if they are modified. The `v` option displays the file size and filename. The `f` option uses the argument `/dev/dsk/c8d0s0` to archive the files.

`./curses.mail`
Copy the file `curses.mail` in the current directory.

The `tar` command responds with the message

```
a ./curses.mail 3 blocks
```

If the file has been modified, it is then copied. The `a` indicates the file has been added to the archive. No message is printed if the file is

identical to the copy in the archive.

4.5 Extracting a specific file

To recover a specific file from a disk created with `tar`, use the command below. In this example, you are recovering `/etc/passwd`.

```
tar xf /dev/rfloppy0 /etc/passwd
```

The components of these commands are as follows:

`tar xf /dev/rfloppy0`
: The `tar` command with option `xf`. The `x` option extracts the specified files from the disk. The `f` option uses the argument `/dev/rfloppy0` to archive the files.

`/etc/passwd`
: The file to be extracted.

The `tar` command then responds with the message

```
Tar: block size = 40
```

`block size` is the number of blocks kept in `tar`'s buffer before sending any output. The default is 40 blocks at a time.

When you're extracting a file with `tar`, change directories to the target directory and specify the name as it appears in the `tar` table of contents listing.

4.6 Creating a table of contents from a `tar` disk

To list the files on the disk, type

```
tar tvf /dev/rfloppy0
```

The components of these commands are as follows:

`tar tvf`
: The `tar` command with options `tvf`. The `t` option displays a table of contents for the files on the disk. The `v` option displays the file size and filename. The `f` option uses the argument `/dev/rfloppy0` to archive the files.

`/dev/rfloppy0`
: The output medium, in this example the disk.

The `tar` command responds with a message reporting the block size and then displays the files with their permissions, ownerships, and dates:

```
Tar: block size = 40
rwxr-xr-x102/202    978 Feb  1 14:16 1984 ./envelope
rwxr-xr-x102/202    211 Apr 16 11:29 1984 ./proofread
rwxr-xr-x102/202    978 May 10 10:34 1984 ./envelope
```

The same filename can appear more than once; `tar` allows multiple copies of a file on the same disk.

4.7 Recovering the latest version of a file

To recover the latest version of a file, in this example the file `curses.mail`, type

```
tar xvf /dev/rfloppy0 ./curses.mail
```

The components of these commands are as follows:

`tar tvf`
: The `tar` command with option `tvf`. The `t` option displays a table of contents for the files on the disk. The `v` option displays the file sizes and filenames. The `f` option uses the argument `/dev/rfloppy0` to archive the files.

`/dev/rfloppy0`
: The output medium, in this example the disk.

`./curses.mail`
: The file to be extracted from the current directory.

The `tar` command then responds with the message

```
Tar: block size = 40
x ./curses.mail, 1135 bytes, 3 tape blocks
```

In this output example, the `x` at the beginning of the line indicates the file has been extracted.

4.8 Recovering a particular version of a file

To recover a particular version of a file, determine which version is needed by displaying the contents of the disk using the `t` option; see "Creating a Table of Contents From a `tar` Disk."

Once you determine the file you want to extract, use the w option with tar. This option waits for you to confirm before executing the indicated action. To extract the May 10 version of mvusr, type

```
tar xvwf /dev/rfloppy0 ./mvusr
```

The components of these commands are as follows:

`tar xvwf /dev/rfloppy0`
: The tar command with the options xvwf. The x option extracts the file from the disk, the v option displays the file size and filename, the w option waits for user confirmation before extracting the file, and the f option uses /dev/rfloppy0 as the archive file.

`./mvusr`
: The file to be extracted in the current directory.

When tar displays the correct filename, type y, as shown in the example below:

```
Tar: block size = 40
x rwxr-xr-x  0/1    140 May  4 18:14 1984 ./mvusr:
x rwxr-xr-x  0/1    162 May 10 10:46 1984 ./mvusr: y
x ./mvusr,
162 bytes, 1 tape blocks
x rwxr-xr-x  0/1    140 May 10 10:48 1984 ./mvusr:
```

A y (yes) indicates that the file should be extracted.

Other responses include n and RETURN, both indicating a "no" response.

5. The dumpfs and restore utilities

The dumpfs and restore backup utilities are recommended for multiuser installations. When you use dumpfs to create a backup, you must use restore to recall a file, directory, or file system.

> *Note:* The /etc/dumpdates file must exist or dumpfs prints the error message:

```
/etc/dumpdates: No such file or directory
```

Therefore, before you run `dumpfs` for the very first time on your system, create an empty file with the command

```
touch /etc/dumpdates
```

You will not need to enter the above command again unless `/etc/dumpdates` is removed by mistake.

5.1 Dump levels

When you use the `dumpfs` command, you specify an incremental backup using dump levels, which are integers that can range from 0 through 9. Instead of specifying a date to indicate that you want to back up everything that has been created or modified since that date, you specify a dump level to indicate that you want to back up everything that has been created or modified since you made a backup with a lower dump level.

For example, a level 7 `dumpfs` backs up all files modified since the most recent backup at dump level 6 or lower. Thus, dump level 0 represents a full backup.

5.1.1 Using dump levels in a monthly backup strategy

Most multiuser installations use a dump strategy based on once-a-month full dumps, as described below:

- Every month do a full dump (level 0):

  ```
  dumpfs 0uF
  ```

- At the end of each week do a weekly dump (level 4):

  ```
  dumpfs 4uF
  ```

- At the end of each working day do a daily dump (level 7):

  ```
  dumpfs 7uF
  ```

The `F` on the command line tells `dumpfs` to write the backup to the floppy drive (`/dev/rfloppy0`). By default `dumpfs` reads the files to back up from the built-in hard disk (`/dev/rdsk/c0d0s0`). `dumpfs` has many keys which can alter its default operation. See `dumpfs`(1M) in *A/UX System Administrator's Reference*.

The level numbers mnemonically represent weeks (4 weeks in a month) and days (7 days in a week). With this strategy, the weekly dump backs up all files modified since the last monthly backup, and daily dumps back up files modified since the last weekly dump.

For the daily and weekly dumps, you can reuse the same backup media, overwriting your previous backups. For monthly dumps (when a level 0 dump is performed), use a set of fresh media and save them for an extended period of time (generally, 6 months to a year).

5.1.2 Restoring from multiple `dump` levels

In the following example, the root (`/`) file system is restored after it was accidentally removed.

To restore the file system to the state at which it existed at the beginning of the month, place the first disk from the current month's level 0 dump disk in the floppy disk drive, and enter the command

```
restore r
```

Note: If the backup archive is comprised of more than one diskette (the usual case), `restore` will prompt you when it is ready for you to insert the next diskette in the series.

When the level 0 restore is complete, install the first disk from the current week's level 4 dump disk in the drive and type

```
restore r
```

If the level 4 backup archive is comprised of more than one disk `restore` will prompt you when it is ready for you to install the next disk in the series. When the level 4 restore is complete, the file system will be restored to its state at the beginning of the week.

To complete the restore, remove the disk from the floppy drive, place the first disk of yesterday's level 7 dump disk in the drive, and type

```
restore r
```

When the level 7 restore is complete, the file system will be restored to its state when you made your most recent backup (last night).

This example demonstrates that using this scheme for routine backups allows you to restore an entire file system to its current status using only three levels of restore.

5.2 Using dumpfs

The dumpfs command operates on the file system mounted on the specified disk partition. It copies all files modified after a certain date to a 3.5-inch disk (or other backup medium).

Because many disks are used to create backups, dumpfs sets a checkpoint for itself at the beginning of each disk. If writing a disk fails at some point, dumpfs waits until you have removed the old disk and inserted a new one, then (after prompting with a question) restarts itself from the checkpoint.

dumpfs prompts with questions when any of the following occurs:

- It reaches the end of a disk.
- It reaches the end of a dump.
- A hard I/O error occurs.

You must answer either y (yes) or n (no) to any of dumpfs's questions.

The following example shows the basic format of a dumpfs command:

```
dumpfs 0uF
```

This command backs up the file system on /dev/rdsk/c0d0s0 (the built-in hard disk). 0 represents a complete backup (not incremental), u updates the system file /etc/dumpdates with the time of this dumpfs, and F tells dumpfs to write the dump on the floppy disk drive. (/dev/rfloppy0).

5.2.1 dumpfs keys

The dumpfs command requires keys to control how it acts. You must specify at least one key or dumpfs will not work. Keys are similar to flag options, except that one must be specified. The keys are described below.

`F` Specifies that the dump is to be written to the floppy disk.

Note: Unless you are writing to an external hard disk or other external device, you should *always* specify this option.

`0-9` Specifies the dump level. `dumpfs` uses this number and the system file `/etc/dumpdates` to determine when the file system was last dumped (at a dump level lower than the number specified) and which files have been modified since.

```
dumpfs 0F
dumpfs 4F
```

In these examples, a level 0 does a full backup, whereas a level 4 only backs up only files modified since a level 3 or lower-level backup.

`u` Writes the date of the beginning of the dump to the file `/etc/dumpdates`. The file records a separate date for each file system and each dump level.

```
dumpfs 7uF
```

Note that `/etc/dumpdates` must exist or `dumpfs` will print an error message. Therefore, before you run `dumpfs` for the first time on your system, create an empty file with the command

```
touch /etc/dumpdates
```

You will not need to enter the above command again unless `/etc/dumpdates` is removed by mistake.

`f` *filename*

Backs up data to the specified device or file, other than the default disk. If the filename is `-`, `dumpfs` writes to the standard output, in which case it can be used as part of a pipeline.

```
dumpfs 4uf /dev/rdsk/c5d0s0 /dev/rdsk/c0d0s0
```

In this example, the contents of `/dev/rdsk/c0d0s0` (the root file system on the internal hard disk) is written to

/dev/rdsk/c5d0s0 (in this case, an external disk).

Note: Be very careful not to transpose the name of file being written (the first filename argument) with the name of file being read (the second filename argument). An empty file system is the likely result of such an action.

W Displays the file system that needs to be dumped. The information is gathered from the files /etc/dumpdates and /etc/mtab.

When using W, dumpfs displays the most recent dump date and level for each file system in /etc/dumpdates. The file systems that need to be dumped are highlighted.

All other options are ignored when W is used, and dumpfs exits immediately.

```
dumpfs W
```

```
Last dump(s) done (Dump '>' file systems):
> /dev/rdsk/c0d0s0 (     /)
      Last dump: Level 5,
      Date Sun Nov 23 16:21
  /dev/rdsk/c0d0s2 (  /usr)
      Last dump: Level 0,
      Date Sun Nov 30 22:01
```

In this example, the file system to be dumped is preceded by the > symbol and not highlighted.

w Similar to W, but displays only those file systems that need to be dumped.

```
dumpfs w
```

```
Dump these file systems:
 /dev/rdsk/c0d0s0 (     /)
      Last dump: Level 5,
      Date Sun Nov 23 16:21
```

n Notifies all operators in the group `operator` that `dumpfs` requires attention.

```
dumpfs 0un /dev/dsk/c0d0s0

DUMPFS:  NEEDS ATTENTION:  Do you want to
abort dump?:
("yes" or "no") yes
DUMPFS:  The ENTIRE dump is aborted.
```

5.3 Using `restore`

The `restore` command reads the backup media created with the `dumpfs` command. The following example illustrates the basic form of the `restore` command:

```
restore r
```

This command reads the default diskette drive (`/dev/rfloppy0`) and expects to find a diskette containing a previously recorded `dumpfs` archive. If the archive spans multiple disks (the usual case), `restore` expects to read the first disk of the archive.

The `r` key tells `restore` to load the entire contents of the archive into the current directory. (`restore` will recreate the entire file and directory hierarchy of the archive beginning with the current directory.)

> *Note:* The `r` key should only be used to restore a complete `dumpfs` archive onto an empty hierarchy or to restore an incremental `dumpfs` archive after a full level 0 restore. Be very careful about where you are in the file system when you use the `r` key. If you start `restore r` from the top of a full hierarchy you will replace current files with older versions.

Like the `dumpfs` command, the `restore` command requires keys to control its actions, and accepts other arguments to specify files or

directories to be restored. See "`restore` Keys" for more information.

5.3.1 Interactive mode for `restore`

The `restore` command features an interactive mode for extracting files from a dumped disk. You can use the `i` key option with `restore`:

```
restore i
```

When `restore` is used in the interactive mode, it reads the directory information from the disk and then creates a shell-like interface complete with the following prompt:

```
restore >
```

This interface lets you move around the directory tree, selecting files to be extracted. The interface also supports commands that aid in locating files.

The commands are described below. If a command needs an argument and one is not provided, the current directory is used by default.

`ls` [*arg*]
Displays the contents of the current directory or the specified directory used as its argument. In the display output,

- Directory names are appended with a `/`.
- Entries selected for extraction are prepended with an `*`.
- If the `v` key (verbose) is set, the inode number for each entry is also displayed.

`cd` *arg*
Changes the current working directory to the directory specified as its argument.

`pwd`
Displays the full pathname for the current working directory.

`add` [*arg*]
Adds the current directory or specified directory to the list of files to be extracted. If a directory is used as an argument, it is

recursively extracted, unless the `h` option is used in the `restore` command line.

`delete` [*arg*]
Deletes the current directory or specified directory from the list of files to be extracted. If a directory is used as an argument, it is recursively extracted, unless the `h` option is used in the `restore` command line.

The easiest way to extract most files from a directory is to add the directory to the extraction list and then delete those files not needed.

`extract`
Extracts all the files on the extraction list from the dumped disk. `restore` then asks which volume you want to mount. The quickest way to extract a few files is to start with the last disk and work toward the first.

`setmodes`
All directories added to the extraction list have set their own owner, modes, and times. If a restore prematurely aborts, this option preserves directories' ownership, modes, and times. Nothing is extracted from the backup medium.

`verbose`
The `ls` command displays the inode number for all entries. `restore` also displays information for each file extracted.

`help`
Displays a list of all available commands in the interactive mode.

`quit`
Immediately terminates the `restore` program, even if all files or directories are not extracted.

5.3.2 `restore` keys

The most commonly used keys to the `restore` command are described below, and examples illustrate how to use them.

`r` Reads and loads the contents of the disk to the current directory. This key should only be used to restore a complete dump tape onto

a clear file system, or to restore an incremental dump tape after a full level 0 restore.

> *Note:* Be very careful about where you are in the file system when you use the `r` key. If you start `restore r` from the top of a full hierarchy you will replace current files with older versions.

`R` Used when a restore has been interrupted. It requests a particular disk from a multivolume disk set to restart a full restore.

```
restore R
```

`x` [*arg*]
Specifies files to be extracted from the disk.

If no file or directory is specified, `restore` begins recursively extracting from the root directory; the entire file system is extracted. If a directory name is specified, it also is recursively extracted, unless the `h` option is used.

The quickest way to extract a few files is to begin with the last disk and work toward the first. For example, using the command

```
restore x filename
```

only the file represented by *filename* is extracted from the backup media. Using

```
restore xh directory-name
```

extracts files from the directory represented by *directory-name*. The command

```
restore x directory-name
```

recursively extracts the entire directory hierarchy represented by *directory-name.*

`t` [*arg*]
Lists the contents of the backup media. If a filename or directory name is used as an argument, the corresponding files that reside on the disk are listed. Like the `x` option, a directory is recursively

listed, unless the h option is also used.

5.3.3 `restore` options

`restore` also uses options that can accompany keys. The options are described below, and examples illustrate how to use them.

`f` *filename*
Counterpart to the `f` option for the `dumpfs` command. Like `dumpfs`, the `f` option used with a filename argument restores data from the file, instead of the disk. If - is used as the filename, `restore` restores from the standard input, allowing `restore` to be used as part of a pipeline.

```
restore rf /tmp/save.level4
```

`v` Stands for "verbose." During a restore, the filename and file type are displayed on the standard output.

```
restore rv
```

`y` Asks whether to abort a restore if it receives a hard I/O error status. Otherwise, `restore` tries to skip a bad disk block and continues.

```
restore ry
```

In the event that a hard I/O error occurs, the `restore` command responds with the message

```
Should I abort restore? yes or no
```

`h` Extracts the actual directory and not the files that it references. This prevents hierarchical restoration of complete subtrees from the disk. (See option `x` above.)

```
restore rh directory-name
```

6. Verifying data on backed-up disks

The `dd` command is used for all backup methods to find hard I/O errors. After that, the appropriate option for each backup method is used to display a table of contents.

Insert each disk and type

```
dd if=/dev/rfloppy0 of=/dev/null bs=90b
```

The components of this command line are as follows:

`dd`	command name
`if=`	input filename
`/dev/rfloppy0`	input file (floppy disk)
`of=`	output filename
`/dev/null`	output file (special file used to discard output)
`bs=90b`	sets both input and output block size to 90 blocks

If the data is successfully read, the following messages appear:

```
17+1 blocks in
17+1 blocks out
```

If messages appear indicating hard I/O errors on any of the disks, the entire backup has to be redone, using newly formatted disks to replace the faulty ones.

Chapter 5
Managing Peripheral Devices

Contents

Chapter 5

Managing Peripheral Devices

1. Introduction

Managing the **peripheral devices** of your A/UX system consists of connecting devices such as terminals, modems, and printers to your computer, maintaining them, and, when necessary, disconnecting them. This chapter helps make these tasks as painless as possible.

This chapter is organized into three main sections. The first section discusses how to connect an additional terminal, the second section describes how to connect a modem, and the third section tells how to hook up a printer and administer the `lp` spooler system.

You might want to connect another computer to your A/UX system. This can happen in at least two different ways. You can connect both computers to a standard network, such as Ethernet. This is known as **networking**, and you should refer to *A/UX Network System Administration* for information on how to do it. Alternatively, you can connect a serial port on the second computer to a serial port on the Macintosh II and then treat the second computer as a peripheral device of the Macintosh II.

To fully understand some sections in this chapter, you should be familiar with the file `/etc/inittab`. This file is discussed in Chapter 2, "System Startup and Shutdown" and `inittab`(4) in *A/UX Programmer's Reference*. You should also note in the section on changing run levels in Chapter 2 that incautious use of the `telinit` command may affect other users.

1.1 Ports

An important concept to understand when you are working with peripheral devices is the **port**. A computer communicates with other equipment through a port. You can hook up peripheral devices such as printers and additional terminals by connecting them (via cables or connectors) to the ports.

There are two aspects to a peripheral connection: the hardware that connects the device to the computer and the software that enables the two to communicate. This chapter concentrates primarily on the software. For the hardware aspect of connecting a device, you should refer to the manual that comes with the hardware.

For now, look at the back of your computer and find the two serial ports. They are round, about a half inch in diameter, and have eight holes each. The port identified by the phone icon on the back of the Macintosh II computer has the A/UX device name `/dev/tty0`; it is also called `/dev/modem`. The port identified by the printer icon has the A/UX device name `/dev/tty1`, and is also called `/dev/printer`. You will be connecting your devices to these serial ports. (Note that these ports can be switched if desired; you can have a modem on port `tty1` and a printer on port `tty0`, even though the the hardware ports have a modem icon at `/dev/tty0` and a printer icon at `/dev/tty1`.)

In addition to physically connecting a device to the computer, which in most cases is a rather simple operation, you must let the computer know what type of device is attached to which port and what the computer must do in each case. This is more involved, but not difficult to do.

2. Setting up a terminal

When you set up your computer and start A/UX running, you'll have at least one terminal working. This first terminal is referred to as the **console**. You can add additional terminals as you need them.

Before the computer can communicate with a peripheral device, it has to know what is expected of it at each port. To do this, the computer uses the system initialization instructions found in the `/etc/inittab` file and the TTY definitions (`gettydefs`) found in the `/etc/gettydefs` file.

The `/etc/gettydefs` file contains information used by the `/etc/getty` program (the process that waits for a user to log in) to determine settings of a `getty` at a given port. It is also where the login prompt for each port is set.

2.1 The /etc/inittab file

The first step in telling the system about a new terminal is to change the `/etc/inittab` file. The `init` program uses this file to decide which processes to run when the system comes up; see Chapter 2, "System Startup and Shutdown." *It is a good idea to make a copy of the* `/etc/inittab` *file before you make any changes.* This provides protection against errors when you're modifying the file. Change to the `/etc` directory and type

```
cp inittab inittab.old
```

Now open the `/etc/inittab` file using a text editor (for example, `vi`). Three of the lines in it should look like these, though not necessarily in this order:

```
co::respawn:/etc/getty console co_9600
00:2:respawn:/etc/getty tty0 at_9600
01:2:respawn:/etc/getty tty1 at_1200
```

These lines are each followed by comments documenting what the entry represents and (if applicable) how to enable it. These comments, respectively, are

```
# Console port
# Port tty0 (modem); set to "respawn"
# Port tty1 (print); set to "respawn"
```

Each of these entries is for a given port. The entries are divided into fields, separated by colons, with the form

id:*run-level*:*action*:*command*

The following is a typical entry:

```
00:2:respawn:/etc/getty tty0 at_9600    # port tty0
```

In this case, `/etc/getty` accepts two arguments. The first, `tty0`, specifies the port on which it should run. The second, `at_9600`, is a label. It tells `init` to read the `/etc/gettydefs` file and use the information contained in the entry beginning with `at_9600` to set up communications with port `tty0`.

The *action* field in `/etc/inittab` entries for terminals should be `respawn`. If you have an entry that relates to a terminal but has anything other than `respawn` in the *action* field and you want people to be able to log in at that terminal, correct the *action* field and close the file.

2.2 The `/etc/gettydefs` file

Another file that is very important in peripheral management is `/etc/gettydefs`. This file is composed of individual entries and each entry has five fields separated by a number sign (`#`). *Note that each entry is separated from the others by a blank line.* Except for the *prompt* field, you may insert whitespace (blanks or tabs) between the fields for readability. The entries are of the form

label # *initial-flags* # *final-flags* # *flow-control* # *prompt* # *next-label*

where the fields are interpreted as follows:

label String that `getty` tries to match in order to use the entry. If the second argument to `/etc/getty` in an `/etc/inittab` entry is for example `co_9600`, the entry that starts with this string is used.

initial-flags
Used to decide how the terminal is set up before log in. The only critical flag at this point is the B flag, which is used to decide the communications baud rate (speed). In the example below, the flag is set to 9600, but it could be any valid baud rate.

final-flags
Take effect when `login` is executed. Again, speed (for instance, `B9600`) is critical. `SANE` is a composite setting; it takes care of other important terminal settings without your having to set them individually. `TAB3` specifies that tabs will be sent to the terminal as spaces. `HUPCL` specifies that the line should hang up on closing the connection. This is generally set for terminals that use a phone line through a modem. A particular attribute can be subtracted from a setting by prefixing it with a tilde (`~`). Thus, `SANE ~PARENB` will set the terminal to have all the attributes of `SANE` with the exception of `PARENB` (parity). For more information on these flags, see `termio`(7) in

A/UX System Administrator's Reference and `gettydefs`(4) in *A/UX Programmer's Reference*.

flow-control
: Specifies the type of flow control to be used on the line. The settings can be `APPLE`, `DTR`, `MODEM`, and `FLOW`. A setting can also be subtracted by prefixing it with a tilde (`~`).

prompt
: Login prompt that will appear at the terminals.

next-label
: Label for `getty` to try in case the current entry causes a failure. If `getty` cannot read the keyboard input using the current definitions, it will try the `gettydef` specified in this field. For instance, if the user logs in at a terminal set up to communicate at 4800 baud but the `getty` being sent to that terminal specifies 9600 baud, the `getty` will not understand the input. When this happens, `getty` will look at this field for an alternative setting. It will try entries in sequence until it finds one expecting input at 4800 baud.

Type

```
more /etc/gettydefs
```

Two of the entries that scroll on the screen after you use this command should look something like the following, though the whole file may be several pages long.

```
co_9600# B9600 # B9600 SANE TAB3 # ~MODEM ~DTR ~FLOW
# \r\n\nApple Computer Inc. A/UX\r\n\nlogin: # co_4800

tt_9600# B9600 # B9600 SANE TAB3 ~MODEM ~DTR ~FLOW
# \r\n\nApple Computer Inc. A/UX\r\n\nlogin: # tt_4800
```

Note: In the above example, output lines have been wrapped onto two lines. When a line in the file has more characters than will fit on a single terminal line, the line will "wrap" onto the next screen line even though the line in the file is actually a single line.

2.3 Using another computer as a terminal

It is possible to attach another computer to an A/UX system in the same way that a terminal is attached, through a serial line. For example, a Macintosh computer running MacTerminal can be treated exactly like a terminal. As long as the files `/etc/inittab` and `/etc/gettydefs` are configured to allow logins on the appropriate port, successful communication can take place, even though the A/UX system has no way of knowing that it is communicating with a computer and not merely a terminal.

There are numerous advantages to thus replacing a terminal with a personal computer that is emulating a terminal. Some of these advantages include the abilities to scroll back through output and to transfer files between the computers. The *A/UX Installation Guide* has step-by-step instructions on how to configure and attach a Macintosh Plus computer as a terminal for A/UX. For more information on performing these functions with MacTerminal, see the MacTerminal user's guide.

3. Setting up a modem

A modem can function in incoming or outgoing mode. On the A/UX system, however, it cannot do both at one time. You can switch between these modes using multiple run levels; see `init`(1M) in *A/UX System Administrator's Reference* and Chapter 2, "System Startup and Shutdown." Or, you can set up a dial-out modem on one port and a dial-in modem on the other port. This section uses the example of a terminal on port `tty0` and a modem on port `tty0`. This is arbitrary. If you want, you can have a modem on port `tty1` and a printer on port `tty0`, even though the the hardware ports have a modem icon at `/dev/tty0` and a printer icon at `/dev/tty1`.

3.1 Dial-out access

If `/etc/inittab` file has the following entries:

```
co::respawn:/etc/getty console co_9600 # Console port
00:2:respawn:/etc/getty tty0 at_9600   # Port tty0
01:2:respawn:/etc/getty tty1 at_9600   # Port tty1
```

and you want the modem on port `tty0` to work as an outgoing device, find the line

```
00:2:respawn:/etc/getty tty0 at_1200    # Port tty0
```

and change it to

```
00:2:off:/etc/getty tty0 at_1200        # Port tty0
```

You can also change it to

```
00:2:off:/etc/getty tty0 mo_1200 #Port tty0 dialout
```

making sure there is a `mo_1200` entry in `/etc/gettydefs`. This kind of change is optional, but it makes it easier for you to read the `/etc/inittab` file.

Because the third field reads `off`, the line now says that at run level 2 there is no `getty` present on port `tty0` (see Chapter 2, "System Startup and Shutdown" for a discussion of run levels). In this condition, the modem functions as an outgoing device only.

You have one more file to modify. Change directories to `/usr/lib/uucp` (this is where most computer communications programs are located). Make a copy of the file `L-devices` and tuck it away somewhere for safety's sake. Now open the file with a text editor and add the following lines at the bottom:

```
DIR tty0 tty0 1200
DIR tty0 tty0 300
```

These two lines tell the system that port `tty0` is connected physically via a cable to another computer and that this cable can be used for transmission speeds of either 300 or 1200 baud. If 1200 baud is not the speed you want (either because you want to transmit at 300 baud or because the other computer receives at 300 baud), you can select 300 baud. For more information about the `L-devices` file, see `cu`(1C) and `uucp`(1C) in *A/UX Command Reference* and Chapter 9, "The UUCP System."

Now that you've made the necessary preparations, you're probably anxious to use your modem. To enable outgoing calls, reboot your system (see "Using the `shutdown` Program" in Chapter 2). When the system comes up multi-user the new entry in `/etc/inittab` will take effect and your modem will function in outgoing mode.

To make a call, you need to know the phone number of a modem attached to another computer set up to receive calls. Once you have a number, one of the commands you can use to get your modem to talk to other computers is `cu`. This is the command used in this manual for testing purposes.

Most modems operate at 300 or 1200 baud (or higher). Using a modem at 300 baud can be extremely slow. If you can, operate the modem at 1200 baud. You can do this using the `cu` flag option `-s`:

```
cu -s[speed]
```

In this case,

```
cu -s1200
```

Another key to getting `cu` to work is the `-l` option. `l` stands for "line" and tells `cu` which port the modem is on. The command line that will allow you to dial out is

```
cu -s1200 -ltty0
```

Type this in. The word `Connected` should appear. This means you are connected to the modem. If you have any problem at this stage, or later, check that the ownership of `/dev/tty0` is the same as the ownership of `/usr/bin/cu`.

The next sequence works only if you are using a Hayes-compatible modem, but the principle is the same in all cases: Once you are connected to the modem, there is a specific way of communicating with it and making it do what you want. If your modem is not Hayes compatible, your modem owner's manual will have information on how to communicate with it. Type in the command that tells your modem to dial a phone number. If your modem is Hayes compatible, try this:

```
ATDT phone-number
```

phone-number must be the number of a computer ready to receive a call. If you are in an office and must request an external line by dialing a number, say 9, before the phone number, then the command to try is

`ATDT 9,`*phone-number*

This tells the modem to dial 9, wait, and then dial the phone number. Once you are connected to the other computer, you will see that computer's login prompt appear on your screen. You can now log in and treat this remote computer as if you were sitting at a terminal in front of it. See `cu`(1C) in *A/UX Command Reference.*

When you are ready to break the connection with the other computer, you must log out.

1. Once you are logged out, type +++ slowly. Your modem answers `OK` on your screen.
2. Type `ATH`. This tells your modem to hang up. Again, if your modem is not Hayes compatible, the manual that comes with it describes how to disconnect from a remote computer. Once again, the modem responds with `OK`.
3. Type the two-character sequence "tilde-dot":

   ```
   ~.
   ```

   This terminates the session with `cu`. The word `Disconnected` appears, and you are back at your own computer.

3.2 Dial-In access

It's often to your advantage to allow other people or other computers to use your system via dial-in access. Once your A/UX system is set up to receive calls, anyone dialing your system can use it as if he or she were connected directly to your computer via a terminal. If you want the modem to work as an incoming device, find the line in `/etc/inittab` that reads

```
00:2:off:/etc/getty tty0 mo_1200 # Port tty0 dialout
```

and change it back to

```
00:2:respawn:/etc/getty tty0 mo_1200 # Port tty0 dialup
```

This line now says that at run level 2 there will be a `getty` present on port `tty0`. This will allow the modem on port `tty0` to function as an incoming device only.

To enable incoming calls, reboot your system (see "Using the `shutdown` Program" in Chapter 2). When the system comes up multi-user the new entry in `/etc/inittab` will take effect and your modem will function in incoming mode. You might also have to instruct your modem to auto-answer; consult the modem's manual for details.

3.3 The modem as a dial-in and a dial-out device

As mentioned earlier, the modem cannot operate as a dial-in device and a dial-out device at the same time. This is because if there is a `getty` present, it will prevent dial-out access. Likewise, if there is no `getty`, there is no way to dial into the computer.

Using multiple run levels can make switching the modem from a dial-out device to a dial-in device and back again an easy process. A few changes are required for this.

To initiate these changes, call up the file `/etc/inittab` using a text editor. Find the line referring to `tty0`. Change it to

```
do:2:off:/etc/getty tty0 mo_1200 # Port tty0 dialout
```

Note that the *id* field on this entry should be changed before you add the next line, which also refers to `tty0`:

```
du:3:respawn:/etc/getty tty0 mo_1200 # Port tty0 dialup
```

These two lines mean that at run level 2 there will be no `getty` at port `tty0` and at run level 3 there will be a `getty`. Therefore at run level 2 the modem attached to port `tty0` can dial out, and at run level 3 it can be called up. If the *id* field is not unique for each entry, you will see an error message when you change to run level 3.

Now you can type from the shell

```
telinit 2
```

to enable dial-out mode. Conversely, you can enter the command

```
telinit 3
```

to enable dial-in mode. Once again, you may need to instruct the modem to enter auto-answer mode.

When you set up things this way in `/etc/inittab`, you must also make sure that all other processes set up to run at run level 2 continue running at run level 3. That is, if your printer is set up to run at run level 2, you probably want it to run also when your modem is in dial-in mode. To enable this, supposing that your printer is connected to port `tty1`, the printer's entry in `/etc/inittab` should read

```
01:23:off:/etc/getty tty1 at_9600 # Port tty1 (print)
```

Notice the entry in the *run-level* field. It is now `23`, to signify that the entry applies to run levels 2 and 3.

At this point, some of the entries in `/etc/inittab` should look like this:

```
co::respawn:/etc/getty console co_9600 # Console port
do:2:off:/etc/getty tty0 mo_1200       # Port tty0 dialout
du:3:respawn:/etc/getty tty0 mo_1200   # Port tty0 dialup
01:23:off:/etc/getty tty1 at_9600      # Port tty1 (print)
```

4. Setting up a printer

This section introduces the commands necessary to use, configure, and maintain one or more printers using the `lp` system. Users accustomed to BSD systems may continue using the `lpr` command to use the `lp` system; for more information on this command, see `lpr`(1).

The **`lp` system** is a collection of programs and files used to manage your printer operations. ("lp" stands for "line printer," a historical acronym.) It is composed of two subsystems: the spooler and the administrative system. The **spooler** system schedules documents to be printed on a particular printer or class of printers and then selects the appropriate interface to run the selected printer. The **administrative system** is a series of commands for configuring and maintaining the entire `lp` system.

The purpose of a print spooling system is to allow users orderly access to printers. Orderly access means that the `lp` system—not users—controls access to a given printer. Users submit **requests** (print jobs) to the spooler the with `lp` command and may specify a particular printer, class of printers, or have the job printed on a default printer.

When configuring the spooler, the system administrator assigns each printer a unique **name** to identify it to the spooler. After naming the printer, the printer may be assigned to a particular named **class** of printers. A name or class of printers is also known as a **destination**.

The spooler maintains a list of user print requests organized by printer name and printer class. Implementing printer classes is not required, but is strongly recommended if you have multiple printers. Classes allow a user to print a job on the first available printer in the class rather than wait for a particular printer. It's usually best to organize printer classes along the lines of printer types. For example, a class named "Laser" might contain only laser printers, while a class named "Matrix" might contain only dot-matrix printers.

The `lp` system is controlled and managed through a set of commands that

- queue or cancel requests
- query the status of requests or of the `lp` system itself
- prevent or allow queuing requests to specific printers
- start or stop the `lp` system
- change the printer configuration

4.1 Definitions

This section defines the important terms used throughout this chapter.

- A **request** is a print job submitted to the `lp` system using the `lp` command.
- A **printer** is a name that uniquely identifies a specific printer to the `lp` system.
- A **class** is a name used to identify a defined list of printers. Each printer may be a member of zero or more classes.
- A **destination** is a printer or a class. Output is normally routed to the **system default destination** unless the user explicitly requests a particular printer or printer class on the `lp` command line. See `lp`(1) in *A/UX Command Reference*.

- A **device** is a device special file in the A/UX `/dev` directory (such as `/dev/iw2`). Each RS-232 port on the back panel of your computer has one device special file associated with it. When the `lp` system writes on the device special file, output is sent to the port. The `lp` system maintains information necessary to associate each printer with a particular device.
- The `lp` **scheduler**, called `lpsched`, schedules print requests received from the `lp` command. `lpsched` runs continuously in the background and is usually started by the `init`(1M) process when A/UX enters the multiuser state. See `lpsched`(1M) and `init`(1M) in *A/UX System Administrator's Reference*.
- Each printer is controlled by an **interface program**. An interface program may be shared by more than one printer. Interface programs perform such tasks as setting port speed, selecting printer options, printing banners, and perhaps filtering certain characters a particular printer may not know how to handle. The `lp` system maintains the information necessary to select the proper interface for a given printer.

4.2 `lp` commands

The commands used to administer the `lp` system can be divided into two categories: those that any user can use, and those that only the `lp` administrator can use. This section gives a short description of what each command does.

4.2.1 Commands for general use

`lp` Submits a print request to the `lp` system. The request will be printed on the default system destination or optionally routed to a specified printer or printer class. A successful request prints a message on the user's terminal similar to

`request id is` *dest-seqno* `(1 file)`

where, *dest* is the name of a printer or printer class and *seqno* is a number unique across the entire `lp` system. See `lp` in *A/UX Command Reference*.

`cancel`
Cancels requests by printers name or request ID number (*dest-*

seqno supplied by `lp`). Specifying printer name cancels the job currently printing. See `lp`(1) in *A/UX Command Reference.*

`lpstat`
Gives various status information about the `lp` system. Also see `lpstat`(1) in *A/UX Command Reference.*

`disable`
Prevents `lpsched` from routing output requests to specified printers.

`enable`
Allows `lpsched` to route output requests to printers. See `enable`(1) in *A/UX Command Reference.*

4.2.2 Commands for `lp` administrators

Each `lp` system must designate a person or persons as `lp` administrator to perform the restricted functions listed below. The `lp` login (provided with the standard A/UX distribution) owns all the files and commands associated with the `lp` system. Either the superuser or any user logged into the A/UX system as `lp` qualifies as the `lp` administrator.

The following commands are described in more detail later in this chapter.

`lpadmin`
Modifies the `lp` configuration. Many features of this command cannot be used when `lpsched` is running. Also see `lpadmin`(1M) in *A/UX System Administrator's Reference.*

`lpsched`
Routes user print requests to interface programs, which do the printing on devices. Also see `lpsched`(1M) in *A/UX System Administrator's Reference.*

`lpshut`
Stops a running `lpsched`. All printing activity is halted, but other `lp` commands may still be used. Also see `lpsched`(1M) in *A/UX System Administrator's Reference.*

`accept`
: Allows `lp` to accept output requests for destinations. Also see `accept`(1M) in *A/UX System Administrator's Reference*.

`reject`
: Prevents `lp` from accepting requests for particular destinations. Also see `reject`(1M) in *A/UX System Administrator's Reference*.

`lpmove`
: Moves output requests from one destination to another. Whole destinations may be moved at one time. This command cannot be used when `lpsched` is running. Also see `lpmove`(1M) in *A/UX System Administrator's Reference*.

4.3 Determining `lp` system status

The `lpstat` command lists status information about printing requests, destinations, and the scheduler (`lpsched`) on your screen. Also see `lpstat`(1) in *A/UX Command Reference*.

Examples

- List the status of all currently printing and pending requests you have made:

```
lpstat
```

- List all currently printing and pending requests of all users:

```
lpstat -o
```

  The status information for a request includes the request ID, the login name of the user, the total number of characters to be printed, and the date and time the request was made.

- Determine whether a printer is available to print requests.

```
lpstat -r -a -p
```

  In order to print a request, the scheduler (`lpsched`) must be running and the particular printer must be accepting requests and enabled. This command produces the necessary information for all printers. See `accept`(1M) in *A/UX System Administrator's Reference* and `enable`(1) in *A/UX Command Reference*.

4.4 The lp scheduler

lpsched is the program that routes the output requests (made with lp) through the appropriate printer interface programs to be printed on line printers. As noted previously, in order to print a request, the scheduler (lpsched) must be running and the particular printer must be accepting requests and enabled.

4.4.1 Installing the scheduler

To install lpsched, make sure the following line in the /etc/rc file is not "commented out" with number signs (#) at the start of the line:

```
rm -f /usr/spool/lp/SCHEDLOCK
```

Also be sure that the following line appears in /etc/inittab:

```
lp:2:once:/usr/lib/lpsched >/dev/syscon 2>&1
```

This starts the lp scheduler each time the A/UX system enters run level 2.

4.4.2 Stopping and starting the scheduler

Each time the scheduler routes a request to an interface program, it records an entry in the log file, /usr/spool/lp/log. This entry contains the login name of the user who made the request, the request ID, the name of the printer on which the request is being printed, and the date and time that printing first started. If a request has been restarted, more than one entry in the log file may refer to the request. The scheduler also records error messages in the log file. When lpsched is started, it renames /usr/spool/lp/log to /usr/spool/lp/oldlog and starts a new log file.

Note: The log files can grow without bounds and eventually occupy an incredible amount of disk space. You should inspect these files periodically and if necessary, truncate them to a zero length with the commands

```
cp /dev/null /usr/spool/lp/oldlog
cp /dev/null /usr/spool/lp/log
```

As mentioned earlier, the lp system won't perform any printing unless lpsched is running. Use the command

```
lpstat -r
```

to find the status of the `lp` scheduler.

`lpsched` is normally started when the `init`(1M) process executes the entry in the `/etc/inittab` file (described previously in "Installing the scheduler") and continues to run until the A/UX system is shut down.

The scheduler operates in the `/usr/spool/lp` directory. When it starts running, it checks whether a file called `SCHEDLOCK` exists; if it does, `lpsched` exits immediately. Otherwise, `lpsched` creates `SCHEDLOCK` to prevent more than one scheduler from running at the same time.

Occasionally, it is necessary to shut down the scheduler to reconfigure the `lp` software. The command

```
/usr/lib/lpshut
```

stops `lpsched`, removes the `SCHEDLOCK` file, and terminates all printing. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted.

To restart the `lp` scheduler, use the command

```
/usr/lib/lpsched
```

Shortly after you enter this command, the `lpstat -r` command should report that the scheduler is running. If not, it is possible that A/UX crashed or was halted improperly, leaving `SCHEDLOCK` in the `/usr/spool/lp` directory.

The command

```
ls /usr/spool/lp/SCHEDLOCK
```

will determine if `SCHEDLOCK` exists. If it does, enter the commands

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
```

Wait about 15 seconds and then determine if the scheduler is running with the `lpstat -r` command.

4.5 Configuring the lp system

The lp system configuration is determined by a set of data files in the /usr/spool/lp directory. Some of these files are ordinary text files, while others contain binary data.

> *Note:* While it is possible to change the text files with a text editor, in a word, *don't!* Altering these files by hand while lpsched is running may cause strange spooler behavior. The lpadmin command is designed with these conditions in and will not allow certain configuration changes to take place until you terminate lpsched. Always use the lpadmin command to reconfigure the lp system.

The lpadmin command is used to change the content of these files. The lpadmin command can have one of the following forms:

```
lpadmin -pprinter [-vdevice] [options]
lpadmin -xdest
lpadmin -d[dest]
```

The three flag options -p, -x, and -d are mutually exclusive. Also, lpadmin will not attempt to alter the lp configuration when lpsched is running, except where explicitly noted below.

In the rest of the chapter, you will be given a series of examples. These examples illustrate possible invocations of the commands in the lp system. As you read, you should get a clear idea (perhaps in the form of a diagram you draw as you read) of the printer classes the examples establish, which printers belong to which classes, what models apply to what printers, and so on.

4.5.1 Introducing new destinations

You can add a new printer with the command

```
lpadmin -pprinter -vdevice [options]
```

where the fields are interpreted as follows:

printer Arbitrary name that must:

- contain no more than 14 characters
- contain only alphanumeric characters and underscores
- not be the name of an existing `lp` destination (whether a printer or a class)

device Pathname of a hard-wired printer or other file that is writable by `lp`.

options Any of the following:

`-c` *class*
Inserts the specified *printer* into the class *class*. The *class* will be created if it does not already exist.

`-e` *printer-to-copy*
Copies the interface program for *printer-to-copy* to be the new interface program for *printer*.

`-h` Indicates that the device associated with *printer* is hardwired (plugged directly in to the computer). This option is always assumed, unless the `-l` option is selected.

`-i` *interface*
Establishes the program found in *interface* as the new interface program for *printer*.

`-l` Indicates that the device associated with *printer* is a login terminal.

`-m` *model*
Selects *model* as the model interface program for *printer*. (See "Printer interface programs" below.)

`-r` *class*
Removes printer *printer* from the specified *class*. If the specified *printer* is the last member of the class, the class will also be removed.

`-v` *device*
: Associate a new device *device* with the printer *printer*. The *device* must be a pathname of a file that is writable by `lp`.

When adding a new printer to the `lp` system, you must select the printer interface program. You may specify this in one of three ways:

- You may select it from a list of model interfaces supplied with `lp` in the `/usr/spool/lp/model` directory (`-m` *model*).
- It may be the same interface that an existing printer uses (`-e` *printer-to-copy*).
- It may be a program supplied by the `lp` administrator (`-i` *interface*).

You may add the new printer to an existing class or to a new class (`-c` *class*). New class names must conform to the same rules for new printer names.

Examples

- Create a printer called `pr1` whose device is `/dev/printer` and whose interface program is the model `hp` interface:

```
/usr/lib/lpadmin -ppr1 -v/dev/printer -mhp
```

- Add a printer called `pr2` whose device is `/dev/tty1` and whose interface is a variation of the model `prx` interface:

```
cp /usr/spool/lp/model/prx newint
edit newint and introduce variations
/usr/lib/lpadmin -ppr2 -v/dev/tty1 -inewint
```

- Create a printer called `pr3` whose device is `/dev/tty1`. `pr3` will be added to a new class called `cl1` and will use the

same interface as printer `pr2`:

```
/usr/lib/lpadmin -ppr3 -v/dev/tty1 -epr2 -ccl1
```

4.5.2 Modifying existing destinations

You can modify existing destinations using `lpadmin`. You must always make modifications with respect to a printer name (`-p`*printer*). The form of the command is

```
lpadmin -pprinter options
```

The options available for modifying existing destinations are

`-c`*class*
: Add the printer to a new or existing class.

`-e`*printer-to-copy*
: Change the printer interface program.

`-i`*interface*
: Change the printer interface program.

`-m`*model*
: Change the printer interface program.

`-r`*class*
: Remove the printer from an existing class. Removing the last remaining member of a class causes the class to be deleted. A destination cannot be removed if it has pending requests. In that case, you should use `lpmove` or `cancel` to move or delete the pending requests.

`-v`*device*
: Change the device for the printer. If this is the only modification, then this may be done even while `lpsched` is running.

Examples

These examples are based on the `lp` configurations created by previous examples.

- Add printer `pr2` to class `cl1`:

```
/usr/lib/lpadmin -ppr2 -ccl1
```

- Change `pr2`'s interface program to the model `prx` interface, change its device to `/dev/tty0`, and add it to a new class called `cl2`:

```
/usr/lib/lpadmin -ppr2 -mprx -v/dev/tty0 -ccl2
```

  Note that printers `pr2` and `pr3` now use different interface programs even though `pr3` was originally created with the same interface as `pr2`. Printer `pr2` is now a member of two classes.

- Add printer `pr1` to class `cl2`:

```
/usr/lib/lpadmin -ppr1 -ccl2
```

  The members of class `cl2` are now `pr2` and `pr1`, in that order. Requests routed to class `cl2` will be serviced by `pr2` if both `pr2` and `pr1` are ready to print; otherwise, they will be printed by whichever one is next ready to print.

- Remove printers `pr2` and `pr3` from class `cl1`:

```
/usr/lib/lpadmin -ppr2 -rcl1
/usr/lib/lpadmin -ppr3 -rcl1
```

  Because `pr3` was the last remaining member of class `cl1`, the class is removed.

- Add `pr3` to a new class called `cl3`:

```
/usr/lib/lpadmin -ppr3 -ccl3
```

4.5.3 Altering the system default destination

You can change or specify the system default destination even when `lpsched` is running. You can do this using the `lpadmin` command with the `-d` flag option. The form of the command is

```
lpadmin -d[dest]
```

The destination *dest* may be omitted; if so, then no destination is established as the system default.

Examples

- Establish class `cl1` as the system default destination:

```
/usr/lib/lpadmin -dcl1
```

- Establish no default destination:

```
/usr/lib/lpadmin -d
```

4.5.4 Removing destinations

You can remove classes and printers using `lpadmin` only if there are no pending requests routed to them. Pending requests must be either canceled using `cancel` or moved to other destinations using `lpmove` before you can remove destinations. If the removed destination is the system default destination, the system will have no default destination until a new default destination is specified. When the last remaining member of a class is removed, the class is also removed. In contrast, removing a class never implies removing printers (see the third example following).

The form of the `lpadmin` command used to remove destinations is

`lpadmin -x`*dest*

The destination being removed must be specified, and no other options are allowed.

Examples

- Make printer `pr1` the system default destination:

```
/usr/lib/lpadmin -dpr1
```

  Then remove printer `pr1`:

```
/usr/lib/lpadmin -xpr1
```

  Now there is no system default destination.

- Remove printer `pr2`:

```
/usr/lib/lpadmin -xpr2
```

  Class `cl2` is also removed because `pr2` was its only member.

- Remove class `cl3`:

```
/usr/lib/lpadmin -xcl3
```

Class `cl3` is removed, but printer `pr3` remains.

4.6 Setting up a printer (example)

As an example of how to set up a hard-wired device for use as an `lp` printer, consider using `tty0` as printer `iw2`, assuming that the printer is an ImageWriter II. As `root`, do the following:

1. Avoid unwanted output from non-`lp` processes and ensure that `lp` can write to the device:

```
chown lp /dev/tty0
chgrp lp /dev/tty0
chmod 600 /dev/tty0
```

2. Change the file `/etc/inittab` so that `tty0` is not a login terminal. In other words, ensure that `/etc/getty` is not trying to log users in at this terminal. Change the entry for `tty0`:

```
00:2:off:/etc/getty tty0 at_9600
```

Enter the command

```
telinit Q
```

to tell the `init`(1M) process to reread the `/etc/inittab` file and remove the `getty`(1M) process from the port.

3. Check the status of `lpsched`. Enter the command

```
lpstat -r
```

If you get a message that says

```
scheduler is running
```

enter the command

```
/usr/lib/lpshut
```

4. Introduce printer `iw2` to `lp` using the model ImageWriter II interface program (the directory `/usr/spool/lp/model`

contains all the interface programs for your system):

```
/usr/lib/lpadmin -piw2 -v/dev/tty0 -miw
```

5. If you are going to use at least two printers, you might consider creating at least a class of printers. To do this at the same time as you introduce `iw2`, replace the line in step 4 with

   ```
   /usr/lib/lpadmin -piw2 -v/dev/tty0 -cclass1 -miw
   ```

   Otherwise, if `iw2` has already been introduced, enter

   ```
   /usr/lib/lpadmin -piw2 -cclass1
   ```

6. You can now let `lpsched` run. Enter

   ```
   /usr/lib/lpsched
   ```

7. When `iw2` is created, it will initially be disabled and `lp` will be rejecting requests routed to it. Allow `lp` to accept requests for `iw2`:

   ```
   /usr/lib/accept iw2
   ```

   This will allow requests to build up for `iw2` and to print when it is enabled at a later time.

8. When you want to print something, be sure that the printer is ready to receive output. This means that the top of form has been adjusted and the printer is online. Enable printing to occur on `iw2`:

   ```
   enable iw2
   ```

   When requests have been routed to `iw2`, they will begin printing.

9. You can now print. To test this, print the password file:

   ```
   lp /etc/passwd
   ```

   In a few seconds, you should get a message with the destination and the request number from `lp`. A few seconds later, the file should start printing.

4.7 Writing printer interface programs

Every `lp` printer must have an interface program that does the actual printing on the device that is currently associated with the printer. Interface programs may be Bourne shell procedures, C programs, or any other executable program.

This section is intended to provide very general guidelines for writing printer interface programs. It assumes you have a working a knowledge of Bourne shell or C-language programming principles. As with any kind of programming, the best teacher is studying examples of code that works. The `lp` model interfaces supplied with A/UX are all written as Bourne shell procedures and can be found in the `/usr/spool/lp/model` directory. This directory contains interface programs for various printer models.

`lpsched`'s home directory is `/usr/spool/lp`. When `lpsched` is ready to print a user request on the printer *printer*, it invokes the appropriate interface program with a command line in the form

`interface/`*printer id login-name title copies options file(s)*

where the fields are interpreted as follows:

printer	name of the interface program in `/usr/spool/lp/interface` directory
id	request ID returned by `lp`
login-name	login name of the user who made the request
title	optional title specified by the user
copies	number of copies requested by the user
options	blank-separated list of class or printer-dependent options specified by the user
file(s)	full pathname(s) of file(s) to be printed

To print a request, the `lp` scheduler, `lpsched`, invokes a printer interface program. The command line argument to the program is based on the spooler configuration and information contained in the `lp` command line entered by the user. The following example

assumes a user named Smith and system default printer called `iw2`.

The command

```
lp /etc/passwd -t"passwords" -n5 -oa -ob
                    /etc/passwd /etc/group
```

(this command should appear on one long line, but it is broken here for formatting purposes) causes `lpsched` to invoke the `iw2` interface program with the line

```
interface/iw2 iw2-52 smith "passwords" 5 "a b"
                     /etc/passwd /etc/group
```

(again, this should appear on one long line.) With the above information you can now write a simple routine to have your interface program interpret the arguments and process the print request appropriately.

When the interface program is invoked, its standard input is `/dev/null` and both the standard output and standard error output are directed to the printer's device. Devices are opened for reading as well as writing when file modes permit. When a device is a regular file, all output is appended to the end of the file.

Given the command line arguments and the output directed to a device, interface programs may format their output in any way they choose. Interface programs must ensure that the proper `stty` modes (terminal characteristics such as baud rate and output options) are in effect on the output device. This may be done in a shell interface only if the device is opened for reading:

```
stty mode... <&1
```

That is, take the standard input for the `stty` command from the device.

When printing is complete, the interface program should exit with a code that indicating the success or failure of the print job. Exit codes (generated by the Bourne shell `exit` *n* and the C statment `exit(`*n*`)` are interpreted by `lpsched` as follows:

`0` The print job has completed successfully.

`1` to `127`
A problem was encountered in printing this request (for example, too many nonprintable characters). The problem will not affect future print jobs. `lpsched` mails the user a letter stating that there was an error in printing the request. The letter contains the error exit code.

greater than `127`
These codes are reserved for internal use by `lpsched`. Interface programs must not exit with codes in this range. As above, `lpsched` uses mail to notify the user of the abnormal exit and exit code.

When problems that are likely to affect future print jobs occur (for example, a device filter program is missing), the interface programs would be wise to disable printers (with the `disable` command) so that print requests are not lost.

4.8 Using the `lp` system

Once `lp` destinations have been created, users may route output to a destination by using the `lp` command. The basic form of the `lp` command is

`lp` [*options*] *files*

There are various options available to the `lp` command; see `lp`(1) in *A/UX Command Reference*. You can use the request ID returned by `lp` to see if the request has been printed or to cancel the request.

The `lp` program determines the destination of a request by checking the following list in order:

- If the user specifies `-d`*dest* on the command line, the request is routed to *dest*.
- If the environment variable `LPDEST` is set, the request is routed to the value of `LPDEST`.
- If there is a system default destination, the request is routed there.

- Otherwise, the request is rejected.

Examples

- There are at least four ways to print the password file on the system default destination:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp -c /etc/passwd
```

  The first way prints the file directly, whereas the last three ways print copies of the file. With the first command, if the file is modified between the time the request is made and the time it is actually printed, the changes will be reflected in the output.

- Print two copies of file `abc` on printer `iw2` and title the output `myfile`:

```
pr abc | lp -diw2 -n2 -t"myfile"
```

- Print file `abc` on a Diablo 1640 printer called `jerry` in 12 pitch, and write to the user's terminal when printing has completed:

```
lp -djerry -o12 -w abc
```

  In this example, `12` is a command to the model Diablo 1640 interface program to print output in 12-pitch mode. Other models may require different options. See `lpadmin`(1M) in *A/UX System Administrator's Reference*.

  Note: The `lp` command runs with with an effective user ID (EUID) of `lp`. This means that A/UX system behaves as if a user named `lp` is reading the files to be printed. Therefore, any files to be printed with a command of the from

  `lp` [*options*] *files*

  must have read permission set for *others*. If this

poses a security problem, you can use the `lp` command in a pipe (`|`) or with the shell input redirect character (<). The reason these two methods work is because the user is feeding input to the `lp` command via the standard input. Two examples:

```
lp < /etc/passwd
pr abc | lp
```

4.8.1 Allowing and refusing requests

When a new destination is created, `lp` at first rejects requests that are routed to it. When you are sure that it is set up correctly, you should allow `lp` to accept requests for that destination. You do so by using the `accept` command. See `accept`(1M) in *A/UX System Administrator's Reference*.

Sometimes it is necessary to prevent `lp` from routing requests to destinations. If printers have been removed or are waiting to be repaired, or if too many requests are building for printers, you may want to have `lp` reject requests for those destinations. The `reject` command performs this function; see `reject`(1M) in *A/UX System Administrator's Reference*. After the condition has been remedied, you should use the `accept` command to allow requests to be taken again.

The acceptance status of destinations is reported by the `-a` option of `lpstat`.

Examples

- Cause `lp` to reject requests for destination `iw2`:

  ```
  /usr/lib/reject -r"printer iw2 needs repair" iw2
  ```

  Any users that try to route requests to `iw2` will encounter the following:

  ```
  lp -iw2 file
  lp: cannot accept requests for destination "iw2"
          -- printer iw2 needs repair
  ```

- Allow `lp` to accept requests routed to destination `iw2`:

```
/usr/lib/accept iw2
```

4.8.2 Allowing and Inhibiting printing

The `enable` command allows the `lp` scheduler to print requests on printers. The scheduler routes requests only to the interface programs of enabled printers. By issuing the appropriate `enable` and `reject` commands, you can enable a printer and at the same time prevent further requests from being routed to it. This can be useful for testing purposes.

The `disable` command reverses the effects of the `enable` command. It prevents the scheduler from routing requests to printers, independently of whether `lp` is allowing them to accept requests. Printers may be disabled for several reasons including malfunctioning hardware, paper jams, and end-of-day shutdowns. If a printer is busy at the time it is disabled, the request that was printing will be reprinted in its entirety either on another printer (if the request was originally routed to a class of printers) or on the same one when the printer is re-enabled.

The `-c` option cancels the currently printing requests on busy printers in addition to disabling the printers. This is useful if strange output is causing a printer to behave abnormally.

Examples

- Disable printer `iw2` because of a paper jam:

```
disable -r"paper jam" iw2
printer "iw2" now disabled
```

- Find the status of printer `iw2`:

```
lpstat -piw2
printer "iw2" disabled since Jan 5 10:15 -
        paper jam
```

- Re-enable `iw2`:

```
enable iw2
printer "iw2" now enabled
```

4.8.3 Moving requests between destinations

Occasionally, it is useful for `lp` administrators to move output requests between destinations. For instance, when a printer is down for repairs, you may want to move all of its pending requests to a working printer. This is one way to use the `lpmove` command. The other use of this command is moving specific requests to a different destination. `lpmove` will refuse to move requests while the `lp` scheduler is running.

Examples

- Move all requests for printer `abc` to printer `bobby`:

  ```
  /usr/lib/lpshut
  /usr/lib/lpmove abc bobby
  /usr/lib/lpsched
  ```

  All of the moved requests are renamed from `abc-nnn` to `bobby-nnn`. As a side effect, destination `abc` will not accept further requests.

- Move requests `jerry-543` and `abc-1200` to printer `bobby`:

  ```
  /usr/lib/lpshut
  /usr/lib/lpmove jerry-543 abc-1200 bobby
  /usr/lib/lpsched
  ```

  The two requests are now renamed `bobby-543` and `bobby-1200` and will be printed on `bobby`.

4.8.4 Canceling requests

You can cancel `lp` requests with the `cancel` command. You can give two kinds of arguments to the `cancel` command: request ids and printer names. Requests named by request ids are canceled, and requests named by printer cause all jobs currently printing on the named printers to be canceled. Both types of arguments may be intermixed. See `lp`(1) in *A/UX Command Reference*.

Example

- Cancel the request that is now printing on printer `bobby`:

```
cancel bobby
```

If the user who is canceling a request is not the same one who made the request, mail is sent to the owner of the request. `lp` allows any user to cancel requests to eliminate the need for users to find `lp` administrators when unusual output should be stopped.

4.9 `lp` system troubleshooting

4.9.1 Problems starting `lpsched`

`lpsched` is usually invoked by the `init`(1M) process when A/UX enters the multiuser state. The invocation is a two-step process:

1. The `/etc/rc` script runs `rm` to remove the `SCHEDLOCK` file in the `/usr/spool/lp` directory.
2. The `init` process invokes `lpsched`.

The purpose of the `SCHEDLOCK` file is to prevent more than one invocation of `lpsched` from running simultaneously. If two (or more) copies of `lpsched` are running at the same time, there is contention over system resources resulting in confused spooler behavior and no files printed.

When `lpsched` finds something wrong in the `lp` system, it attempts to mail an error message to `root` and make an entry in the `/usr/spool/lp/log` file. The `SCHEDLOCK` file is not removed under these conditions, as invoking `lpsched` again without clearing the trouble would likely produce the same error conditions.

4.9.2 Restarting `lpsched`

When `lpsched` stops due to error conditions

- Check `root`'s mail for correspondence from `lp`
- Check `/usr/spool/lp/log` for error messages.

- Check the spooler status additional messages about individual printers with the `lpstat -t` command.
- Use the `ps -ulp` command to determine if multiple copies of `lpsched` are running. (`lpstat` will not report multiple copies of `lpsched`.) Write down the process ID of each `lpsched` that you find.

Clear the error conditions:

- If `lpsched`'s messages indicate damaged spooler configuration files (see "`lp` system files"), remake the `lp` system using the `lpadmin` command (see "Configuring the `lp` system" in this chapter.)
- Use the `kill`(1) command to kill *all* of the `lpsched` processes. (See `kill`(1) in *A/UX Command Reference*.)
- Clear any other error conditions indicated by the error messages.

Restart `lpsched` with the commands

```
rm /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
```

Finally, check the status of the entire `lp` system with the `lpstat -t` command. If everything appears normal in `lpstat`'s report, perform the ultimate test: print a file.

4.9.3 Repairing a damaged `outputq` file

The `lp` system keeps all queue data in the binary file `/usr/spool/lp/outputq`. If this file has been damaged the spooler will not run correctly and old job files will remain in the subdirectories of `/usr/spool/lp/request`. To correct this condition

1. Check the file system with the `fsck` utility (see Chapter 6, "Checking the A/UX File System: `fsck`").
2. Stop the `lp` spooler with the `/usr/lib/lpshut` command.

3. Remove the contents of the directories under `/usr/spool/lp/request`, but *do not* remove the directories themselves. Use the `rm ./*` command, but with caution! Use `pwd` to be sure you are in the directory you think you are in!

4. Null out the corrupted `outputq` file with the command

```
cp /dev/null /usr/spool/lp/outputq
```

5. Restart the spooler with the `/usr/lib/lpsched` command.

4.9.4 `lp` system files

This section describes the system files used by `lp`.

`/usr/spool/lp/FIFO`
: A named pipe, readable and writable only by `lp`. Any `lp` command may write on this file but only `lpsched` may read it.

`/usr/spool/lp/default`
: A text file containing the name of the system default printer; empty if there is no default printer or destination.

`/usr/spool/lp/log`
: A text file containing a record of all printing requests.

`/usr/spool/lp/oldlog`
: Each time `lpsched` is started, it copies `/usr/spool/lp/log` into this file and then truncates `/usr/spool/lp/log` to zero length.

`/usr/spool/lp/outputq`
: A binary data file which holds the `lp` request queue information.

`/usr/spool/lp/pstatus`
: A binary data file which contains status information for each printer (whether a printer is enabled or disabled).

`/usr/spool/lp/qstatus`
: A binary data file which contains the acceptance status for

each printer (whether a printer is accepting or rejecting requests).

`/usr/spool/lp/interface/`*printer*
: *printer* is the name of a particular printer interface program in the `/usr/spool/lp/interface` directory. All files in this directory should be executable by `lp` only.

`/usr/spool/lp/seqfile`
: A text file containing the sequence number assigned to the last request printed. The number will always be in the range `1-9999`.

`/usr/spool/lp/class`
: A directory containing one text file for each printer class. The file name corresponds to the class. Each class file contains the names of the printers belonging to the class.

`/usr/spool/lp/member`
: A directory containing one text file for each printer. The file name corresponds to the printer name. The contents of the file is the name of the device-special file in the `/dev` directory corresponding to the printer.

`/usr/spool/lp/model`
: A directory containing sample printer interface programs (Bourne shell scripts).

`/usr/spool/lp/request`
: A directory containing subdirectories named for each destination (class or printer) known to the `lp` system. The subdirectories are used for temporary storage of spooler commands and print requests (text).

`/usr/spool/lp/SCHEDLOCK`
: The purpose of the `SCHEDLOCK` file is to prevent more than one invocation of `lpsched` from running simultaneously. See "Stopping and Starting the Scheduler."

`/usr/spool/lp/OUTQLOCK`
`/usr/spool/lp/PSTATLOCK`
`/usr/spool/lp/QSTATLOCK`
`/usr/spool/lp/SEQLOCK`

Various lock files for preventing `lp` system commands from modifying data in the data files described above. Each file has an expiration time, after which any `lp` system command may remove it and then modify the data file which it is locking. The principle of these lock files is similar to the `SCHEDLOCK` described above.

4.9.5 `lp` system command permissions

All `lp` system utilities with the exception of `lpsched` should be owned by `lp` with the set user-id (SUID) bit turned on (see `chmod`(1) and `chown`(1) in *A/UX Command Reference*). `lpsched` may be owned by either `root` or `lp`.

Chapter 6
Checking the A/UX File System
fsck

Contents

Figures

Chapter 6

Checking the A/UX File System

`fsck`

1. Introduction to `fsck`

The `fsck` program (for "File System Check") locates and resolves inconsistencies within a file system. It is intended for use during normal booting procedures and at any other time that the system administrator suspects inconsistencies within the file system (such as immediately following a system crash).

The A/UX operating system treats almost everything in its environment as a file. To operate on a file in the A/UX environment, you need only refer to it by name. The general function of the A/UX file system is to

- support the seemingly simple interface on A/UX mass storage media (disks, tapes, and tape cartridges)
- permit the kernel to find data on the disk
- load the data into main memory
- periodically update the disk with the modifications performed on the data in main memory

Sometimes this updating fails, usually because of power failures or improper system shutdown, and inconsistencies within the file system result. Fortunately, in most cases you can resolve these inconsistencies using the A/UX `fsck` program.

`fsck` checks the location of files on disk and uses redundancies and known parameters to resolve inconsistencies. A **known parameter** is information about the file system that does not change, such as the number of characters per block or the number of blocks in a disk. A **redundancy** is information that the system maintains in more than one place, such as the size of each file and the number of blocks not currently in use.

In many cases, `fsck` can fix the damage. Sometimes, however, `fsck` can only report cryptic messages about the damage that has been done. In these cases, a system administrator who knows the structure and functioning of the file system is required. The goal of this chapter is to provide you with this knowledge.

Section 2 of this chapter describes the structure of the A/UX file handling system. Section 3 discusses how `fsck` operates and when you should run it. Section 4 discusses how to use `fsck`. Section 5 describes `fsck`'s phases and error messages.

Before you begin this chapter you should know how to use the commands `ls`, `rm`, `cp`, `mv`, `cd`, and `ln` (for more information about these commands, refer to Chapter 4, "The A/UX File System," in *Getting Started With A/UX*). You should also be able to bring the system to single-user or multiuser status (see Chapter 2, "System Startup and Shutdown" for more information).

2. Overview of the A/UX file system

It is important to understand the organization of the A/UX file system and some of the commands that manipulate this organization before you begin to work with `fsck`. This section gives you a brief overview of the relevant file and directory information.

2.1 Partitions, file systems, and hierarchies

The A/UX **file handling system** is the complete set of data structures, commands, and subroutines used to manipulate data stored on a physical device. On the A/UX system, a physical storage device (usually a disk) is divided into logical devices called **partitions**, each of which may contain an A/UX file system. You can establish or remove partitions using the `dp` command; see `dp`(1M) and `gd`(7) in the *A/UX System Administrator's Reference*, and `bzb`(4), and `dpme`(4) in *A/UX Programmer's Reference*.

A **file system** is a logical device that contains the data structures (directories, files, and inodes, among others) that implement all or part of the A/UX directory hierarchy. The A/UX **directory hierarchy** is simply the collection of all files currently available to the system. This coincides with the collection of all files on currently mounted file

systems; see `mount`(1M) in *A/UX System Administrator's Reference*. From the user's point of view, the directory hierarchy resembles an inverted tree, branching out from the root directory (`/`).

The general term "hierarchy" is used for the collection of files and subdirectories of a given directory; for instance, we may speak of the `/usr` hierarchy, meaning all files and directories under the `/usr` directory. A hierarchy will coincide with the A/UX directory hierarchy only when the directory under consideration is the root directory.

2.2 Bytes, blocks, and disk allocation

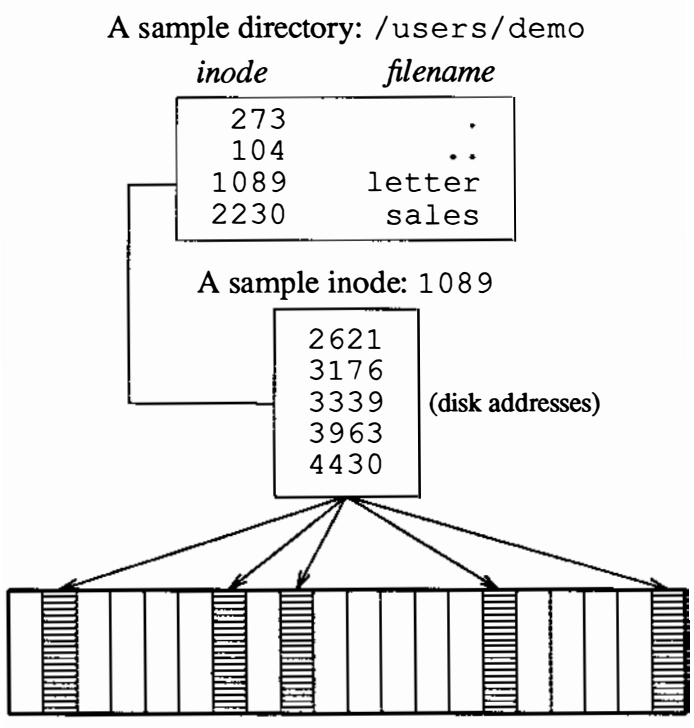
A file system is divided up into units called **blocks**. A block is a contiguous sequence of 512 bytes (characters). Each character in a file is represented by a byte of information and each byte resides in a block.

A file may reside in many blocks (also called **data blocks**) located on different portions of the disk. Ideally, a file's data blocks should be as near to one another as possible to optimize disk I/O operations. However, in reality, as the disk is used and files are deleted, data blocks become increasingly scattered. To handle this, inodes are made to point to each block of the file in sequence.

2.3 Inodes

Inodes contain information about files, the most important of which is a list of locations on the disk where the file's contents are to be found. Note that the inode does not point to a single location on the disk, but to several discrete locations (see "Direct and Indirect Blocks," later in this chapter). Figure 6-1 illustrates the relationship between the directory `/users/demo`, a file in that directory named `letter`, the i-number and inode associated with `letter`, and the disk locations where `letter`'s contents are stored.

Figure 6-1. I-number relationships



To find out how many blocks each of the files in your current directory occupies, enter the command

```
ls -s
```

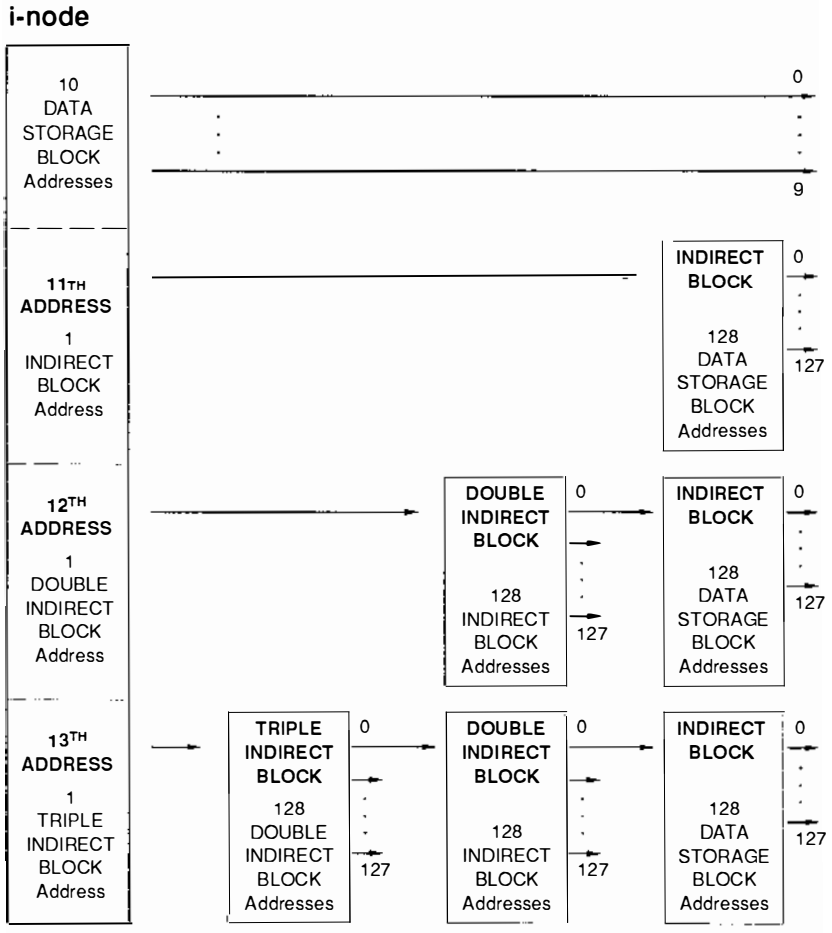
The number that appears next to each filename is a count of the blocks used by that file. There should be 1 block for each 512 characters and an additional block for any remaining characters. For example, a file with 512 characters occupies 1 block, while a file with 513 characters occupies 2 blocks.

2.4 Direct and indirect blocks

The inode contains 13 disk addresses. The first 10 addresses point to the first 10 blocks of the file. These blocks are the **direct data blocks**. The first 512 characters of a file reside in the block located at the first address to which the inode points. If a file has more than 512 characters, it will continue at the 2nd address to which the inode points. If it has more than 1024 characters, it will continue at the 3rd address, and so on, until the first 10 addresses (5120 characters) have been used.

If a file has more than 5120 characters, the 11th address given in the inode is then taken into consideration. The 11th disk address points to an **indirect block**. An indirect block contains the addresses for the next 128 blocks that the file can use. Figure 6-2 illustrates these connections.

Figure 6-2. Indirect blocks



If the file is larger than 138 blocks (10 blocks for the first 10 addresses, 128 for the 11th), it continues at the 12th address given in the inode. The 12th address points to a **double indirect block**, which refers to up to 128 indirect blocks that the file can use. This gives a total of 8,459,264 bytes of storage: 10 from the first 10 addresses + 128 from the 11th address + (128 * 128) from the 12th address * 512 (per block) = 8,459,264 bytes.

If the file is even larger, the 13th address in the inode is called into action. This last address points to a **triple indirect block,** which references up to 128 double indirect blocks, each of which in turn refers to up to 128 indirect blocks, and so on. now you have 1,082,201,088 bytes:

10 + 128 + (128 * 128) + (128 * 128 * 128) * 512 = 1,082,201,088

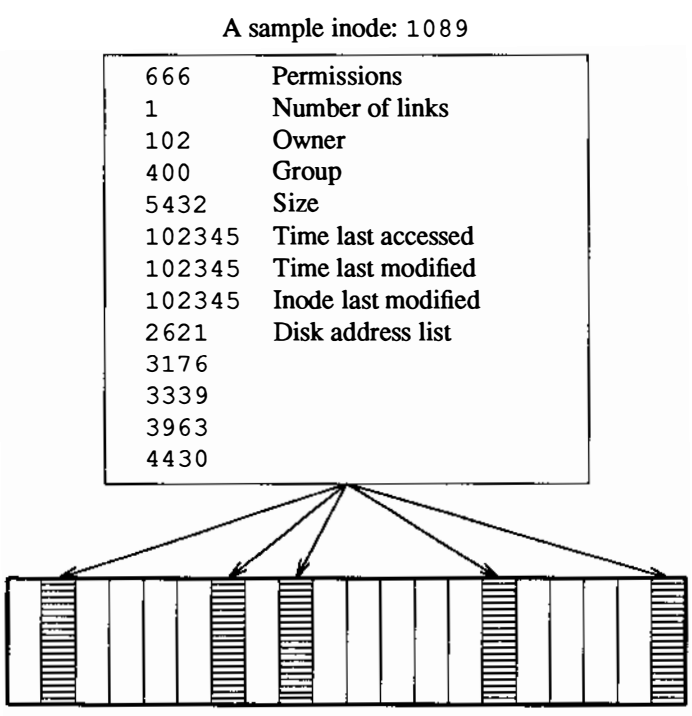
This is the maximum size a file can be in a file system that has 512-byte blocks. A system with a larger bytes-per-block value could have a much larger theoretical limit. But because the file size is represented in an inode by a signed 32-bit quantity, a file can never get larger, theoretically speaking, than about 2 gigabytes.

However, these are theoretical maximum file sizes. In practice, a file can never be this big because the physical storage device imposes a ceiling on the maximum file size. A file as big as the theoretical maximum size would be unmanageable, and in most cases it would not fit on a single disk.

2.5 More on Inodes

Inodes contain information about the location of the data blocks that make up a file. Figure 6-3 illustrates the other important information that inodes contain about a file.

Figure 6-3. Additional information in an inode



You will note from the figure that inodes record three different time-related statistics about a file: access time, modification time, and inode modification time. **Access time** is the last time the file was read and **modification time** is the last time the file was written.

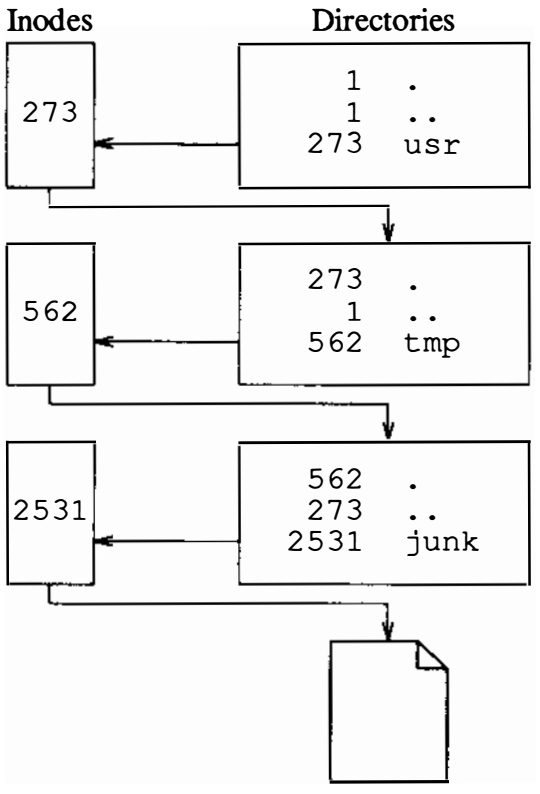
Inode modification time is referred to as "creation time" in some publications. This is really a misnomer, because modifying, changing permissions or changing ownership all update the inode modification time on a file.

You can use the `ls -l` command to see some of this additional information. For additional information, see `ls`(1) in *A/UX Command Reference*.

2.6 Starting from the top

Figure 6-4 illustrates the connection, through multiple inodes, between the root directory and a file located several levels below the root directory.

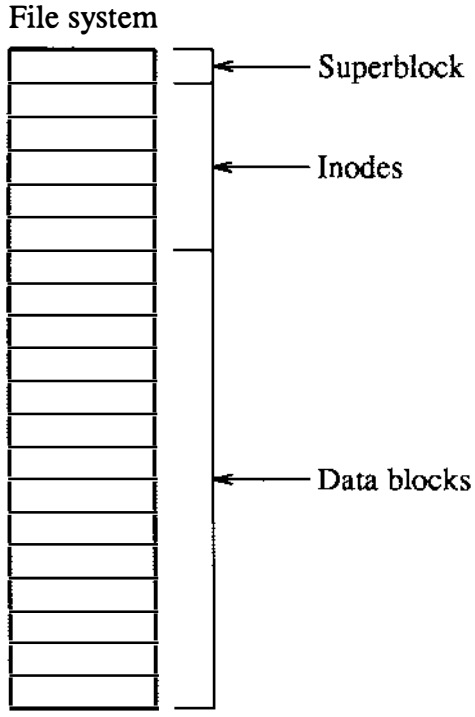
Figure 6-4. File-directory connection through inodes



2.7 Inode location

Unlike files and directory files, inodes are a fixed size (64 bytes) and reside in fixed locations on the disk. This is why inodes have i-numbers instead of names. The i-number 30 points to the 30th inode in the inode area on the disk. Figure 6-5 shows inode locations in a file system.

Figure 6-5. I-numbers and inode locations



2.8 The superblock and i-list

The system keeps track of all free inodes and all active inodes in a twofold way. First, it uses the second block in the file system (address 1) to store information about the file system itself. This block is called the **superblock**. (Address 0, the first block, is used to maintain disk information.) Second, it uses a list called the **i-list**, discussed below.

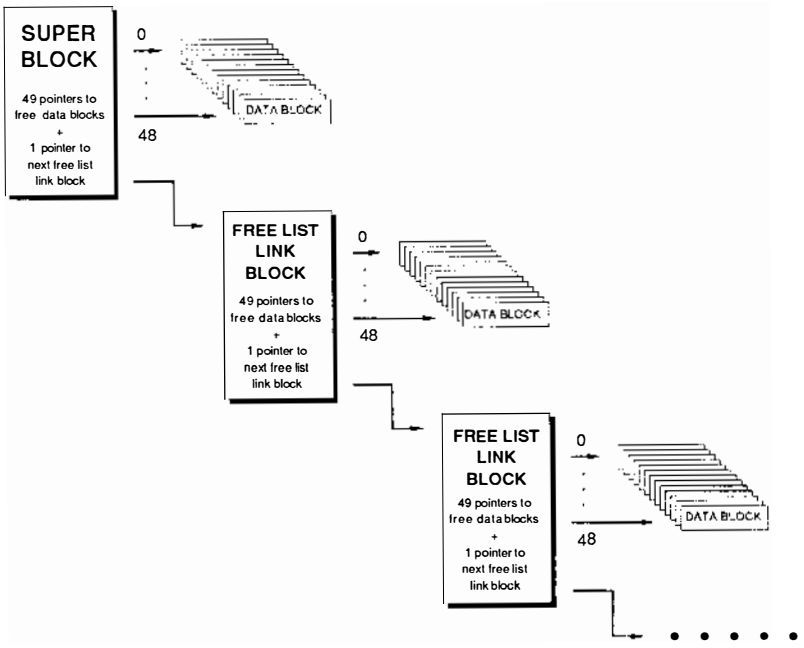
2.8.1 The superblock and the free list

Among other things, the superblock contains information about the size (in blocks) of the i-list, the size (in blocks) of the entire file system, the total number of free blocks, and, most importantly, the free inode list and the total number of free inodes.

The superblock also contains the beginning of a linked list that maintains information about the free blocks. This list is called the **free list**. The free list is a list of addresses of all the free blocks in the file system. The free list is maintained in discrete chunks, in the following way. The first 50 addresses in the free list are contained within the superblock; this is usually referred to as the **superblock free list.** 49 of these addresses are simply the addresses of the 49 next free blocks. The 50th address is the address of a block containing 50 more addresses. There is no standard name for this block of 50 addresses; let us call it a **free list link block**.

Each free list link block contains 49 direct addresses and 1 indirect address, until the end of the free list is reached. (A link block could, in theory, hold as many as 128 addresses, but only 50 are stored in a free list link block so that a link block can be read entirely into the superblock when the addresses it contains are needed.) The beginnings of a free list are illustrated in Figure 6-6.

Figure 6-6. The superblock and the free list



If you want to know more about the superblock, the file `/usr/include/svfs/filsys.h` contains the declaration for a structure called `filsys` and all the items that the structure maintains updated in the superblock. The inode structure is defined in `/usr/include/svfs/ino.h`.

2.8.2 The i-list

The second way the system keeps track of inodes (necessary because the superblock deals only with the free inodes) is the i-list, which is located on disk immediately after the superblock. The **i-list** is a list of file inodes. Each inode is a 64-byte structure containing information about the physical location of the file associated with it.

The important thing is that the superblock and the i-list are located not only on disk but also in main memory, and the main memory versions are the updated ones. This is important when it comes to the interaction between main and secondary memory.

2.9 Block I/O and the buffer cache

2.9.1 Block I/O

It would be both risky and expensive to keep all data in main memory (also called **primary memory, in-core memory,** or **RAM**). So instead, most files are in secondary memory (for example, a disk) and the system brings them into main memory as necessary. If you modify files, the system writes the modified version back into secondary memory for future use.

When a program reads (or writes) data from a tape or a disk, the system extracts blocks of 512 bytes and brings them into main memory. It would be impractical and even unsafe to bring it into main memory without imposing limits and some degree of organization on the amount of data transferred. It would also be highly inefficient to be constantly doing physical I/O operations whenever data has to be transferred from primary to secondary memory and vice versa. For that reason, the system maintains a list of **buffers** for each device. This buffer pool is said to constitute a **data cache** for the device in question.

2.9.2 The buffer cache

When a program asks the system to read data from a file, the system first searches the cache for the desired block.

If the block is found (when, for instance, the system opens a file that is already open), the data is made available to the requesting process without a physical I/O read operation. If the data is not found in the cache, the buffer that has been unused for the greatest period of time is renamed and data is transferred into it from the disk and made available.

When a process writes a file, the operation occurs in reverse order. A write request first writes data to the buffer cache. Things are written to disk only when the cache is full. Therefore, information about bad writes refers generally to unusual bad writes to the buffer, and bad disk writes are generally reported too late to prevent the file system from being corrupted.

2.10 Special files and the `/dev` directory

There are three types of files in A/UX: regular files, directory files, and special files.

When you list your files using the `ls -l` command, the system response looks like

```
-rw-rw----   1 groupname   13 Sep 25 11:28 file
drwxrwx---   2 groupname   32 Sep 25 11:28 directory
```

For regular files, such as the first one listed in the example, the first character in the permissions field is `-`. In the case of directory files, this character is always `d`. However, if you were to look at `/dev/rdsk/c0d0s0` and `/dev/dsk/c0d0s0` with the command

```
ls -l /dev/rdsk/c0d0s0 /dev/dsk/c0d0s0
```

you'd get

```
crw-------  2 bin  24, 0 May 25  1987 /dev/rdsk/c0d0s0
brw-------  2 bin  24, 0 May 25  1987 /dev/dsk/c0d0s0
```

`/dev/rdsk/c0d0s0`'s first character in the permissions field is `c` and `/dev/dsk/c0d0s0`'s is `b`. Either a `b` or a `c` in this position indicates that a file is a special file.

2.10.1 The contents of special-file inodes

The files in the `/dev` directory are all **special files** that the system uses to select a device driver (see below) for performing physical I/O.

These files are actually just names and inodes with no associated data on disk (and thus a size of zero bytes). Instead of storing information about the number of bytes in a file, these inodes contain a major and minor device number for each special file. These are the numbers you see listed from the `ls -l` command displayed above.

A **device driver** is a program that controls the actual physical I/O to the devices listed in the `/dev` directory. However, the device driver itself doesn't reside in the `/dev` directory; rather, it is compiled directly into the kernel.

There is a different device driver for each kind of device (disk, printer, and so on). The system uses the major device number to access the correct device driver. The minor device number is passed to the driver, which uses this argument to select the correct physical device.

In summary, the system takes the following steps in response to requests to open special files (such as `fsck` may make):

- looks in `/dev` directory for a file with the requested name
- gets the i-number associated with the filename
- finds the inode specified by the i-number
- gets the major device number stored in the inode
- uses this number to select the appropriate device driver
- passes the minor device number to the device driver

The driver then uses the minor device number to select the correct physical device (the proper partition in the case of the disk device).

As you may have already guessed, devices (and therefore device drivers and their corresponding special files) come in two forms, `b` (block) and `c` (character)—hence the `b` and `c` in the `ls -l` listing. These names refer to the method of I/O used with each type of device.

Block devices such as disks use the block I/O buffer cache mentioned previously, and are thus written to, and read from, one block at a time. Character devices such as terminals, line printers, and modems are written to, and read from, one character at a time.

Disk partitions are peculiar beasts because each partition has both a character and a block device driver associated with it and thus two special files in the `/dev` directory. This means you can access disk partitions in two ways. They're normally accessed as block devices through the A/UX directory hierarchy. But programs like the backup program `dumpfs` use the character device drivers, and programs like `dd`, `cpio`, `volcopy`, and `fsck` should be used that way (that is, `fsck /dev/rdsk/c0d0s0` rather than `fsck /dev/dsk/c0d0s0`) because they perform much faster when they run on the character devices.

Now the listing

```
crw-------  1 bin  24, 0 May 25  1987 /dev/rdsk/c0d0s0
brw-------  1 bin  24, 0 May 25  1987 /dev/dsk/c0d0s0
```

is clearly a listing of a character (raw) device and a block device. One has a `c` and the other a `b` as the first entry in their permissions fields. We see that the same device can have two different interfaces.

3. How `fsck` works

As you open, create, and modify files, the system keeps track of all pertinent information about them. This information—including their block sizes, their i-numbers, file system active and free inodes and total number of active, used, and free blocks—is maintained and updated in main memory. If the system crashes or becomes corrupt for any reason, chances are the various file systems will become inconsistent. This is because the information in the main memory superblock was not written to disk before the problem, while the table of active inodes probably was, because it is written to disk whenever a file is released.

A system crash is not the only cause for file system corruption. Time and normal wear and tear of the disk will produce inconsistencies that have to be detected and corrected.

`fsck` works by comparing one or more items of information to one or more items of equivalent information. For instance, it compares the number of free blocks available to the number of total blocks in the file system minus the number of blocks in use. If the two numbers are not equal, `fsck` gives you an error message. This kind of error comes

from an inconsistent update or one that was performed out of order. To understand the problems `fsck` is designed to solve, it is necessary to understand these updates.

3.1 File system updates

This section describes the various file system updates the system performs every time you create, modify, or remove a file. There are five types of file system updates:

Superblock

Contains information about the size of the file system and inode list, part of the free list, a count of free blocks, a count of free inodes, and part of the free inode list.

A mounted file system's superblock is written to disk whenever the file system is unmounted or a `sync`(1M) command is issued. The exception to this is that the root file system is always mounted.

A file system's superblock is prone to inconsistency because every change to the file system's blocks or inodes modifies it.

Inode

Contains information about the inode's type (which may be directory, data, or special), the number of directory entries linked to it, the list of blocks claimed by the inode, and the inode's size. An inode is written to disk when the file associated with it is closed. In fact, all **in-core blocks** are also written to disk when a `sync` system call is issued, so the period of danger when inconsistencies can appear is reduced to that between `sync` calls.

Indirect blocks

Can be one of three types: single-indirect, double-indirect, or triple-indirect.

Indirect blocks, as well as the first ten blocks of a file, are written to disk whenever they have been modified or released by the operating system. More precisely, they are queued in the buffer cache for eventual writing. Physical I/O is deferred until the buffer is needed by the A/UX operating system or a `sync` command is issued.

Inconsistencies in an indirect block directly affect the inode that owns it.

Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system.

Data block

Written to disk whenever it has been modified.

There are two types of data blocks: plain and directory. **Plain data blocks** contain the information stored in a file. **Directory data blocks** contain directory entries.

`fsck` does not attempt to check the validity of the contents of a plain data block.

In contrast, `fsck` checks each directory data block for inconsistencies involving the following:

- directory inode numbers pointing to unallocated inodes
- directory inode numbers greater than the number of inodes in the file system
- incorrect directory inode numbers for `.` and `..`
- directories disconnected from the file system

If a directory entry inode number (that is, a file's inode) points to an unallocated inode, `fsck` may remove that directory entry, depending on how `fsck` was invoked. (For instance, it will remove the entry if invoked with the `-y` flag option.) See "`fsck` Messages." This condition might occur when the data blocks containing the directory entries are modified and written out while the inode is not yet written out.

If a directory entry inode number points beyond the end of the inode list, `fsck` may remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for `.` should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

The directory inode number entry for `..` should be the second

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

If the directory inode numbers are incorrect, `fsck` may replace them with the correct values.

`fsck` checks the general connectivity of the file system. If directories are found not to be linked into the file system, `fsck` will link the directory back into the file system's `lost+found` directory. This condition can be caused if inodes are written to disk but the corresponding directory data blocks are not.

Free list

The system updates the free list when a file has been deleted or when a file has been enlarged past a block boundary. As described previously (see Figure 6-6), the free list begins in the superblock, which contains 49 addresses of blocks available for data storage plus the address of the next free list link block. When the first 49 addresses are used up, the kernel uses the address of the next free list link block to reinitialize the free list in the superblock. As long as there is disk storage available, this new list will contain the address of the next 49 available free blocks plus the address of the next free list link block.

Free-list link blocks are chained from the superblock. Therefore, inconsistencies in free list link blocks ultimately affect the superblock.

Inconsistencies that you can check are the following: a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

3.2 `fsck`'s phases

There are six file system check phases in `fsck` (one of which is generally optional) as well as an initialization phase. Each phase of the `fsck` program passes through the whole file system. If you invoke `fsck` without a device name in the command line it will repeat all its phases for all devices.

3.2.1 Phase 1: Check blocks and sizes

`fsck` checks the inode list. In this phase, `fsck` may discover error conditions that result from checking inode types, setting up the zero-link-count table, checking inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format. Phase 1B runs only if any duplicate blocks (that is, blocks that belong to multiple inodes) are found.

3.2.2 Phase 2: Check pathnames

`fsck` removes directory entries that point to inodes that have error conditions from Phase 1 and Phase 1B. In this phase, `fsck` may discover error conditions that result from root inode mode and status, directory inode pointers out of range, and directory entries pointing to bad inodes.

3.2.3 Phase 3: Check connectivity

`fsck` checks the directory connectivity seen in Phase 2. In this phase, `fsck` may discover error conditions that result from unreferenced directories and missing or full `lost+found` directories.

3.2.4 Phase 4: Check reference counts

`fsck` reports messages that result from unreferenced files; a missing or full `lost+found` directory; incorrect link counts for files, directories, or special files; unreferenced files and directories; bad and duplicate blocks in files and directories, and incorrect total free inode counts.

3.2.5 Phase 5: Check free list

`fsck` checks each free list link block for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

3.2.6 Phase 6: Salvage free list

`fsck` concerns itself with the free list reconstruction. It lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

4. Using `fsck`

4.1 When to use `fsck`

If a file system is inconsistent, it will be made worse if you continue to use it (thus modifying it further) without running `fsck`. Because it is so important to keep your file systems consistent, the `fsck` program is programmed into the system startup procedure (see Chapter 2, "System Startup and Shutdown") and is automatically run each time you start up A/UX.

You can also invoke `fsck` at any time in a session by bringing the system down to single-user mode and typing

`fsck` [*options*] [*file-systems*]

You can specify *options* to direct `fsck` to run in a different way; see `fsck`(1M) in *A/UX System Administrator's Reference* for a complete list of options. You can specify *file-systems* to run `fsck` on a different file system. File system names are defined in the file `/etc/fstab`; see `fstab`(4) in *A/UX Programmer's Reference* for details. If you type `fsck` without options or file system names, it will run on all file systems.

Note that the file system on which `fsck` is running should be unmounted, or at least no writes should occur while `fsck` is running. This is important because `fsck` performs more than one pass on the file system and if it is modified from pass to pass, the results will be unpredictable.

When `fsck` finds an inconsistency in a file system, it informs you with a message such as

```
/dev/dsk/c0d0s0 POSSIBLE FILE SIZE ERROR I=2405
```

The message can also look like

```
/dev/dsk/c0d0s0 FREE INODE COUNT WRONG IN SUPERBLK
FIX?
```

The second message is an example of one of `fsck`'s interactive error messages. `fsck` will perform the corrective action only if you confirm that it should do so by typing `y`. Otherwise it will either continue or terminate, depending on the nature of the problem encountered.

4.2 fsck options

You can specify the following flag options on the `fsck` command line:

`-y` The `-y` flag option automatically provides a yes response to all questions asked by `fsck`.

`-n` The `-n` flag option automatically provides a no response to all questions asked by `fsck`, and does not open the file system for writing.

`-f` Fast check. This causes `fsck` to check block and sizes (Phase 1) and check the free list (Phase 5). The free list will be reconstructed (Phase 6) if it is necessary.

`-p` [*pass-to-start*]
The `-p` flag option causes `fsck` to automatically fix those things in the file system that it can fix without assistance. The optional *pass-to-start* argument specifies the starting pass number (that is, the phase number to start). The default is 1.

`-q` Quiet `fsck`. With the `-q` option, `fsck` does not print size-check messages in Phase 1. Unreferenced fifo's will silently be removed. If `fsck` requires it, counts in the superblock will be automatically fixed and the free list salvaged.

`-s` *x*
The `-s` option causes `fsck` to ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the superblock of the file system. The file system should be unmounted while this is done; if this is not possible, you should be careful that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The `-s` *x* flag option allows for creating an optimal free-list organization and has the following form:

`-s` *blocks-per-cylinder*:*blocks-to-skip*

If *x* is not given, the values used when the file system was created are used. If these values were not specified, then the

value `400:7` is used.

`-S` *x*

Conditionally reconstruct the free list. This flag option is like `-s` *x* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using `-S` will force a no response to all questions asked by `fsck`. This flag option is useful for forcing free-list reorganization on uncontaminated file systems.

`-t` *file*

If `fsck` cannot obtain enough memory to keep its tables, it uses a scratch file. If the `-t` option is specified, the file named in the next argument is used as the scratch file, if needed. Without the `-t` flag, `fsck` will prompt the operator for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when `fsck` completes.

`-D` [*options*]

The *options* argument may be missing, in which case directories are merely checked for bad blocks, or it may be any of the following:

`B` to check for and clear parity bits in filenames

`C` to check whether all trailing characters in the filename are null

`CZ` to check and write nulls in all trailing characters in the filename

4.3 `fsck`: A sample interaction

If you bring the system down to single-user mode and type

```
fsck -n
```

`fsck` starts running. Because you didn't specify a file system, `fsck` reads the file `/etc/fstab`, which contains a list of the files to be checked in a case like this. Also, because you invoked `fsck` with the `-n` option, `fsck` assumes you're always answering no to its prompts and it doesn't open the file system for writing. In other words, you are

safe.

> *Note:* Unless you know exactly what you're doing, it's recommended that you always invoke `fsck` a first time with the `-n` option, read the messages, and decide in advance the course of action to take before you invoke it a second time without the `-n` option.

For each file system checked, you will see something like the following on your screen:

```
/dev/dsk/c0d0s0 (NO WRITE)
File System: root Volume: 001

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
402 files 12374 blocks 950 free
```

`fsck` makes five passes (sometimes six, which is discussed later in this chapter) and lets you know at what phase it is at any given time and in what file system (in this example, `/dev/dsk/c0d0s0`). Different and separate file systems are discussed in "Multiple File Systems and `fsck`" later in this chapter.

The above example was a routine check that found no problems or inconsistencies. If `fsck` finds something, a response like the following appears on your screen:

```
/dev/rdsk/c0d0s1 (NO WRITE)
File System: usr Volume: 003

** Phase 1 - Check blocks and sizes
POSSIBLE FILE SIZE ERROR I=1147

POSSIBLE FILE SIZE ERROR I=1195

** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
1350 files 20582 blocks 18686 free
```

However, the messages can be more obscure and require some action on your part. This is discussed in "`fsck` Messages" later in this chapter.

You can also invoke `fsck` with the `s`*x* or the `S`*x* flag options discussed earlier. The `s`*x* flag option unconditionally makes a new free list, discarding the present one; the `S`*x* flag option does the same only if nothing is wrong in the free list. The immediate result of these options is speeding up disk operations in busy file systems, because a new free list is likely to have more contiguous portions of free space.

The `D` flag option is useful for doing extra consistency checks on directories, and it does not add significant time to the process. It is included in your startup `fsck` procedures. For more information about these and other options, see `fsck`(1M) in *A/UX System Administrator's Reference*.

4.4 Multiple file systems and `fsck`

As stated previously, to run `fsck` on all of your file systems you simply enter the command:

```
fsck
```

You can run `fsck` on an individual file system with the command

```
fsck file-systems
```

5. fsck messages

When fsck detects an inconsistency, it reports the error condition in a message. If a response is required, fsck waits for it. This section explains the possible messages in each phase, the meaning of each message, the possible responses, and the related error conditions. The messages are organized by the fsck phase in which they can occur.

5.1 fsck initialization phase messages

Before a file system check can be performed, certain tables have to be set up in memory and certain files have to be opened. This is the "initialization phase" of fsck.

This phase may discover error conditions that result from errors in command line options, memory requests, file opening, file status, file system size checks, and scratch file creation.

5.1.1 fsck option errors

The following messages may occur when you specify the fsck command line incorrectly. See fsck(1M) in *A/UX System Administrator's Reference* for further details.

`c option?`

c stands for any character that is not a legal flag option to fsck. Legal options are -y, -n, -s, -S, -q, -D, and -t. fsck terminates on this error condition.

`Bad -t option`

The -t option was not followed by a filename. fsck terminates on this error condition.

`Invalid -s argument, defaults assumed`

This is only a warning. The -s option was not followed by 3, 4, or *blocks-per-cylinder*:*blocks-to-skip*. fsck assumes a default value of 400 blocks per cylinder and 9 blocks to skip.

`Incompatible options: -n and -s`

It's not possible to salvage the free list without modifying the file system. fsck terminates on this error condition.

5.1.2 Memory request errors

The following messages may occur when `fsck` detects errors in memory allocations requests. These messages may indicate a serious problem that could require technical assistance.

`can't fstat standard input`

`fsck`'s attempt to `fstat` standard input failed. `fsck` terminates on this error condition.

`can't get memory`

`fsck`'s request for memory for its virtual memory tables failed. `fsck` terminates on this error condition.

5.1.3 Errors in opening files

The following messages may occur when `fsck` cannot open a file or file system.

`can't open:` *f*

The default file system check file *f* (`/etc/fstab`) cannot be opened for reading. `fsck` terminates on this error condition. Check the access modes of *f* and modify accordingly.

`can't open` *f*

The file system *f* cannot be opened for reading. `fsck` ignores this file system and continues checking the next file system given. Check access modes of *f* and modify accordingly.

f `is not a block or character device`

`fsck` has been given a regular filename by mistake. `fsck` ignores this file system and continues checking the next file system given. Check file type of *f* and modify accordingly.

5.1.4 File status errors

The following messages may occur when `fsck` cannot obtain a file's status. These errors may indicate a serious problem requiring additional technical assistance.

`can't stat root`

`fsck`'s request for statistics about the root directory (`/`) failed.

`fsck` terminates on this error condition.

`can't stat` *f*

`fsck`'s request for statistics about the file system *f* failed. It ignores this file system and continues checking the next file system given. Check the access modes of *f*.

5.1.5 File system size and inode list size

The file system size and inode list size are critical pieces of information to the `fsck` program. While there is no way to actually check these sizes, `fsck` can check if they're within reasonable bounds. The file system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The number of inodes must be less than 65,535. All other file system checks depend on the correctness of these sizes.

The following message may occur when `fsck` detects an inconsistency in the file system size.

`Size check: fsize` *x* `isize` *y*

More blocks are used for the inode list *y* than there are blocks in the file system *x*, or there are more than 65,535 inodes in the file system. `fsck` ignores this file system and continues checking the next file system.

5.1.6 Scratch file errors

A **scratch file** is a temporary file that `fsck` creates when you invoke `fsck` with the `-t` flag option. This option might be necessary if there is not enough core memory to hold `fsck`'s tables. This message may occur if `fsck` cannot create its own scratch file for a particular file system.

`can't create` *f*

`fsck`'s request to create a scratch file *f* failed. It ignores this file system and continues checking the next file system given. Check access modes of *f*.

5.1.7 Interactive messages

These messages require your yes or no response to the `CONTINUE` prompt. "Yes" may be `Y` or `y` and "no" may be `N` or `n`; these are

shown in uppercase below. These messages may indicate a serious problem because the file system cannot be completely checked. You may need to obtain additional technical assistance.

`CAN NOT SEEK: BLK` *b*
`CONTINUE?`

`fsck`'s request for moving to a specified block number *b* in the file system failed.

Possible responses to the `CONTINUE` prompt are

`Y` Attempt to continue to run file system check. If the problem persists, run `fsck` a second time to recheck this file system. If the block *b* was part of the virtual memory buffer cache, `fsck` will terminate with the message "`Fatal I/O error.`"

`N` Terminate `fsck`.

`CAN NOT READ: BLK` *b*
`CONTINUE?`

`fsck`'s request for reading a specified block number *b* in the file system failed.

Possible responses to the `CONTINUE` prompt are

`Y` Attempt to continue to run file system check. A second run of `fsck` should be made to recheck this file system. If the block *b* was part of the virtual memory buffer cache, `fsck` will terminate with the message "`Fatal I/O error.`"

`N` Terminate `fsck`.

`CAN NOT WRITE: BLK` *b*
`CONTINUE?`

`fsck`'s request for writing a specified block number *b* in the file system failed. The disk is write protected.

Possible responses to the `CONTINUE` prompt are

`Y` Attempt to continue to run file system check. A second run of `fsck` should be made to recheck this file system. If the

block *b* was part of the virtual memory buffer cache, `fsck` will terminate with the message "`Fatal I/O error.`"

`N` Terminate `fsck`.

5.2 Phase 1: Check blocks and sizes

This phase of `fsck` execution checks the inode list. In this phase, `fsck` may discover error conditions that result from checking inode types, setting up the zero-link-count table, checking inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format.

Although each individual inode has a small chance to become inconsistent, there are so many of them that it's almost as likely that an inconsistency will occur in the inode list as in the superblock.

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

5.2.1 Inode type errors

Each inode contains a mode word that describes the type and state of the inode.

Inodes may be one of five types: regular, directory, special block, special character or fifo, according to the file involved. If an inode is not one of these types, it is an illegal type. Tell `fsck` to clear the inode.

`UNKNOWN FILE TYPE I=`*i*
`CLEAR?`

The mode word of the inode *i* indicates that the inode is not a special block or character inode, a regular inode, or a directory inode.

Possible responses to the `CLEAR` prompt are

`Y` Deallocate inode *i* by zeroing its contents. This will always invoke the `UNALLOCATED` error condition in Phase 2 for each directory entry pointing to this inode.

N Ignore this error condition.

5.2.2 Zero-link-count table errors

Each inode contains a stored count of the total number of directory entries linked to the inode. `fsck` verifies the link count of each inode by traversing down the total directory structure, starting from the root directory, and calculating an actual link count for each inode.

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

If the stored link count is nonzero and the actual link count is zero, `fsck` can link the disconnected file to the `lost+found` directory (at your direction). If the stored and actual link counts are nonzero and unequal, `fsck` can replace the stored link count by the actual link count.

```
LINK COUNT TABLE OVERFLOW
CONTINUE?
```

An internal table for `fsck` has no more room. This is a rare error. Contact your Apple representative for assistance.

Possible responses to the `CONTINUE` prompt are

Y Continue with program. This error condition will not allow a complete check of the file system. You should make a second run of `fsck` to recheck this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

N Terminate `fsck`.

5.2.3 Bad or duplicate blocks

Each inode contains a list of pointers to lists (indirect blocks) of all the blocks claimed by the inode.

`fsck` checks each block number claimed by an inode for a value lower than that of the first data block or greater than the last block in the file system. If the block number is outside this range, it is a bad block number. If an indirect block was not written to disk, an inode may

contain a lot of bad blocks. In this case, `fsck` will clear both inodes (at your direction).

`fsck` compares each block number claimed by an inode to a list of already allocated blocks. If a block number is already claimed by another inode, the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number. If there are any duplicate blocks, `fsck` will make a partial second pass of the inode list to find the inode of the duplicated block. This is necessary because without examining the files associated with these inodes for correct content, there is not enough information available to decide which inode is corrupted and should be cleared. Most of the time, the inode with the earliest modify time is incorrect and should be cleared. This condition can occur by using a file system with blocks claimed by both the free list and other parts of the file system.

A large number of duplicate blocks in an inode may be due to an indirect block not being written to disk. In this case, `fsck` will clear both inodes (at your direction).

b `BAD I=`*i*

> *i* stands for the actual number on your screen. Inode *i* contains block number *b* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system.
>
> This error condition may invoke the "`EXCESSIVE BAD BLKS`" error condition in Phase 1 if inode *i* has too many block numbers outside the file system range.
>
> This error condition will always invoke the `BAD/DUP` error condition in Phase 2 and Phase 4.

`EXCESSIVE BAD BLKS I=`*i*
`CONTINUE?`

> *i* stands for the actual number on your screen. There is more than a tolerable number of blocks (usually ten) with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system associated with inode

i.

Possible responses to the `CONTINUE` prompt are

`Y` Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to recheck this file system.

`N` Terminate `fsck`.

b `DUP I=`*i*

Inode *i* contains block number *b*, which is already claimed by another inode.

This error condition may invoke the "`EXCESSIVE DUP BLKS`" error condition in Phase 1 if inode *i* has too many block numbers claimed by other inodes.

This error condition will always invoke Phase 1B and the `BAD/DUP` error condition in Phase 2 and Phase 4.

`EXCESSIVE DUP BLKS I=`*i*
`CONTINUE?`

i stands for the actual number on your screen. There is more than a tolerable number of blocks (usually ten) claimed by other inodes.

Possible responses to the `CONTINUE` prompt are

`Y` Ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to recheck this file system.

`N` Terminate `fsck`.

`DUP TABLE OVERFLOW`
`CONTINUE?`

An internal table in `fsck`, containing duplicate block numbers, has no more room. Possible responses to the `CONTINUE` prompt are

`Y` Continue with file system check. This error condition will not allow a complete check of the file system, so a second run of `fsck` should be made to recheck this file system after recompiling `fsck`. If another duplicate block is found, this error condition will repeat.

`N` Terminate `fsck`.

5.2.4 Inode size errors

Each inode contains a 32-bit (4-byte) size field. This size indicates the number of characters in the file associated with the inode. This size can be checked for inconsistencies; for example, directory sizes that are not a multiple of 16 characters, or the number of blocks actually used do not match the number indicated by the inode size.

A directory inode within the file system has the directory bit on in the inode mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 bytes for the inode number and 14 bytes for the file or directory name).

`fsck` will warn of such directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

A rough check of an inode's size field consistency can be performed by computing from the size field the number of blocks that should be associated with the inode and comparing it to the actual number of blocks claimed by the inode.

`fsck` calculates the number of blocks by dividing the number of characters in an inode by the number of characters per block and rounding up. `fsck` adds one block for each indirect block associated with the inode. If the actual number of blocks does not match the computed number of blocks, `fsck` will warn of a possible file-size error. This is only a warning because the system does not fill in blocks in files created in random order.

> *Note:* These messages are only warnings. If you use the `-q` option on the `fsck` command line, these messages are not printed.

`POSSIBLE FILE SIZE ERROR I=`*i*

i stands for the actual number on your screen. The inode size does not match the actual number of blocks used by the inode. If this error occurs, write down the inode number. When `fsck` finishes, continue in single-user mode, mount the file system in which the error occurred and get its corresponding filename in the following way: Enter the command

```
ncheck -i i fs
```

where *i* is the number of the inode as provided by the "`POSSIBLE FILE SIZE ERROR`" message, and *fs* stands for the name of the file system in which the message occurred. `ncheck` will print the name of the file on the screen. Copy it into a temporary name and run `sync`. Examine the file, and if you want to retain it, remove the original and give the temporary file its original name. Note that just using `mv` would not do the job.

`DIRECTORY MISALIGNED I=`*i*

i stands for the actual number on your screen. The size of a directory inode is not a multiple of the size of a directory entry (usually 16). It should be cleared.

5.2.5 Inode format errors

Each inode contains a mode word that describes the type and state of the inode. Inodes may be found in one of three states: unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted inode. An inode can get in this state if bad data is written into the inode list, which can be caused by a hardware failure. The only possible corrective action is for `fsck` to clear the inode.

`PARTIALLY ALLOCATED INODE I=`*i*
`CLEAR?`

Inode *i* is neither allocated nor unallocated.

Possible responses to the `CLEAR` prompt are

`Y` Deallocate inode *i* by zeroing its contents.

N Ignore this error condition.

5.3 Phase 1B: Rescan for more duplicates

When a duplicate block is found in the file system, the file system is rescanned to find the inode that previously claimed that block. If the duplicate block is found, the following error condition occurs.

b DUP I=*i*

Inode *i* contains block number *b* that is already claimed by another inode.

This error condition will always invoke the BAD/DUP error condition in Phase 2.

Inodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

5.4 Phase 2: Check pathnames

This phase removes directory entries that point to inodes that had error conditions from Phase 1 and Phase 1B. In this phase, `fsck` may discover error conditions that result from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes.

5.4.1 Root Inode mode and status errors

```
ROOT INODE UNALLOCATED. TERMINATING
```

The root inode (always inode number 2) has no allocate mode bits. This error condition indicates a serious problem that may require additional technical assistance. The program will terminate.

```
ROOT INODE NOT DIRECTORY
FIX?
```

The root inode (usually inode number 2) is not directory inode type.

Possible responses to FIX prompt are

Y Replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a very large number of error conditions will be produced.

N Terminate `fsck`.

```
DUPS/BAD IN ROOT INODE
CONTINUE?
```

Phase 1 or Phase 1B has found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to the `CONTINUE` prompt are

Y Ignore `DUPS/BAD` error condition in root inode and attempt to continue to run the file system check. If root inode is not correct, then this may result in a large number of other error conditions.

N Terminate `fsck`.

5.4.2 Directory Inode pointers range errors

```
I OUT OF RANGE I=i NAME=f
REMOVE?
```

A directory entry *f* has an inode number *i* greater than the end of the inode list.

Possible responses to the `REMOVE` prompt are

Y Remove the directory entry *f*.

N Ignore this error condition.

5.4.3 Directory entries pointing to bad inodes

UNALLOCATED I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t* NAME=*f*
REMOVE?

A directory entry *f* has an inode *i* without allocate mode bits. The owner *o*, mode *m*, size *s*, modify time *t*, and filename *f* are printed.

If the file system is not mounted and the -n option wasn't specified, the entry will be removed automatically if the inode it points to is character size 0.

Possible responses to REMOVE prompt are

Y Remove the directory entry *f*.

N Ignore this error condition.

DUP/BAD I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t* DIR=*f*
REMOVE?

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with directory entry *f*, directory inode *i*. The owner *o*, mode *m*, size *s*, modify time *t*, and directory name *f* are printed.

Possible responses to REMOVE prompt are

Y Remove the directory entry f.

N Ignore this error condition.

DUP/BAD I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t* FILE=*f*
REMOVE?

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with directory entry *f*, inode *i*. The owner *o*, mode *m*, size *s*, modify time *t*, and filename *f* are printed.

Possible responses to the REMOVE prompt are

Y Remove the directory entry *f*.

N Ignore this error condition.

BAD BLK *b* IN DIR I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t*

This message only occurs when the -q option is used. It means

that a bad block was found in inode *i*.

`fsck` checks directory blocks for nonzero padded entries, inconsistent `.` and `..` entries, and embedded slashes in the name field. This error message indicates that, at a later time, you should either remove the directory inode if the entire block looks bad or change (or remove) those directory entries that look bad.

5.5 Phase 3: Check connectivity

This phase checks the directory connectivity seen in Phase 2. In this phase, `fsck` may discover error conditions that result from unreferenced directories and missing or full `lost+found` directories.

5.5.1 Unreferenced directories

`UNREF DIR I=`*i* `OWNER=`*o* `MODE=`*m* `SIZE=`*s* `MTIME=`*t* `RECONNECT?`

The directory inode *i* was not connected to a directory entry when the file system was traversed. The owner *o*, mode *m*, size *s*, and modify time *t* of directory inode *i* are printed. `fsck` will force the reconnection of a nonempty directory.

Possible responses to the `RECONNECT` prompt are

`Y` Reconnect the directory inode *i* to the file system in the directory for lost files (usually `lost+found`). This may invoke `lost+found` error conditions (described below) if there are problems connecting directory inode *i* to `lost+found`. This may also invoke the `CONNECTED` message (described below) if the link was successful.

`N` Ignore this error condition. This will always invoke the `UNREF` error condition (described below).

`SORRY. NO lost+found DIRECTORY`

There is no `lost+found` directory in the root directory of the file system. In this case, `fsck` ignores the request to link a directory in `lost+found`. This will always invoke the `UNREF` error condition (described below). If `lost+found` exists, check its access modes. See `fsck`(1M) and `mklost+found`(1M) in *A/UX System Administrator's Reference* for further details.

`SORRY. NO SPACE IN lost+found DIRECTORY`

There is no space in the `lost+found` directory to add another entry; `fsck` ignores the request to link a directory in `lost+found`. This will always invoke the `UNREF` error condition (described below). Clean out unnecessary entries in the `lost+found` directory or make it larger. See `fsck`(1M) in *A/UX System Administrator's Reference* for further details.

`DIR I=`*i* `CONNECTED. PARENT WAS I=`*j*

This is an advisory message indicating that a directory inode *i* was successfully connected to the `lost+found` directory. The parent inode *j* of the directory inode *i* is replaced by the inode number of the `lost+found` directory.

5.6 Phase 4: Check reference counts

This phase checks the directory connectivity seen in Phase 2 and Phase 3.

In this phase of `fsck`, messages are reported that result from unreferenced files; a missing or full `lost+found` directory; incorrect link counts for files, directories, or special files; unreferenced files and directories; bad and duplicate blocks in files and directories, and incorrect total free inode counts.

5.6.1 Unreferenced files

`UNREF FILE I=`*i* `OWNER=`*o* `MODE=`*m* `SIZE=`*s* `MTIME=`*t* `RECONNECT?`

Inode *i* was not connected to a directory entry when the file system was traversed. The owner *o*, mode *m*, size *s*, and modify time *t* of inode *i* are printed. If the `-n` option is not set and the file system is not mounted, empty files will not be reconnected and will be cleared automatically.

Possible responses to the `RECONNECT` prompt are

`Y` Reconnect inode *i* to the file system in the directory for lost files (usually `lost+found`). This may invoke `lost+found` error condition (described below) if there are problems connecting inode *i* to `lost+found`.

N Ignore this error condition. This will always invoke the CLEAR error condition (described below).

5.6.2 lost+found directory errors

```
SORRY. NO lost+found DIRECTORY
```

There is no lost+found directory in the root directory of the file system; fsck ignores the request to link a file in lost+found. This will always invoke CLEAR error condition (described below). Check access modes of lost+found. Also see mklost+found(1M) in *A/UX System Administrator's Reference*.

```
SORRY. NO SPACE IN lost+found DIRECTORY
```

There is no space to add another entry to the lost+found directory in the root directory of the file system; fsck ignores the request to link the file. This will always invoke the CLEAR error condition (described below). Check the size and contents of lost+found.

```
CLEAR?
```

The inode mentioned in the immediately previous error condition cannot be reconnected.

Possible responses to the CLEAR prompt are

Y Deallocate inode mentioned in the immediately previous error condition by zeroing its contents.

N Ignore this error condition.

5.6.3 Incorrect free inode counts

The superblock contains a count of the total number of free inodes within the file system. fsck compares this count to the number of inodes it found free within the file system. If the counts do not agree, fsck may replace the count in the superblock by the actual free inode count.

```
LINK COUNT FILE I=i OWNER=o MODE=m SIZE=s
MTIME=t COUNT=x SHOULD BE y
ADJUST?
```

The link count for inode *i*, which is a file, is *x* but should be *y*. The owner *o*, mode *m*, size *s*, and modify time *t* are printed.

Possible responses to the ADJUST prompt are

Y Replace the link count of file inode *i* with *y*.

N Ignore this error condition.

LINK COUNT DIR I=*i* OWNER=*o* MODE=*m* SIZE=*s*
MTIME=*t* COUNT=*x* SHOULD BE *y*
ADJUST?

The link count for inode *i*, which is a directory, is *x* but should be *y*. The owner *o*, mode *m*, size *s*, and modify time *t* of directory inode *i* are printed.

Possible responses to the ADJUST prompt are

Y Replace the link count of directory inode *i* with *y*.

N Ignore this error condition.

LINK COUNT F I=*i* OWNER=*o* MODE=*m* SIZE=*s*
TIME=*t* COUNT=*x* SHOULD BE *y*
ADJUST?

The link count for file *f* inode *i* is *x* but should be *y*. The filename *f*, owner *o*, mode *m*, size *s*, and modify time *t* are printed.

Possible responses to the ADJUST prompt are

Y Replace the link count of inode *i* with *y*.

N Ignore this error condition.

5.6.4 Unreferenced files/directories

UNREF FILE I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t*
CLEAR?

Inode *i*, which is a file, was not connected to a directory entry when the file system was traversed. The owner *o*, mode *m*, size *s*, and modify time *t* of inode *i* are printed. If you don't set the -n option and the file system is not mounted, empty files will be cleared automatically.

Possible responses to CLEAR prompt are

Y Deallocate inode *i* by zeroing its contents.

N Ignore this error condition.

UNREF DIR I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t*
CLEAR?

Inode *i*, which is a directory, was not connected to a directory entry when the file system was traversed. The owner *o*, mode *m*, size *s*, and modify time *t* of inode *i* are printed. If you don't set the -n option and the file system is not mounted, empty directories will be cleared automatically. Nonempty directories will not be cleared.

Possible responses to the CLEAR prompt are

YES Deallocate inode *i* by zeroing its contents.

NO Ignore this error condition.

5.6.5 Bad and duplicate blocks in files and directories

BAD/DUP FILE I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t*
CLEAR?

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with file inode *i*. The owner *o*, mode *m*, size *s*, and modify time *t* of inode *i* are printed.

Possible responses to the CLEAR prompt are

Y Deallocate inode *i* by zeroing its contents.

N Ignore this error condition.

BAD/DUP DIR I=*i* OWNER=*o* MODE=*m* SIZE=*s* MTIME=*t*
CLEAR?

Phase 1 or Phase 1B has found duplicate blocks or bad blocks associated with directory inode *i*. The owner *o*, mode *m*, size *s*, and modify time *t* of inode *i* are printed.

Possible responses to the CLEAR prompt are

`Y` Deallocate inode *i* by zeroing its contents.

`N` Ignore this error condition.

```
FREE INODE COUNT WRONG IN SUPERBLK
FIX?
```

The actual count of the free inodes doesn't match the count in the superblock of the file system. If you specify the `-q` option, the count will be fixed automatically in the superblock.

Possible responses to the `FIX` prompt are

`Y` Replace the count in superblock by the actual count.

`N` Ignore this error condition.

5.7 Phase 5: Check free list

The free list starts in the superblock and continues through the free list link blocks of the file system.

Each free list link block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The free list check begins in the superblock. `fsck` checks the list count for a value of less than 0 or greater than 50. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free list link block pointer is nonzero, `fsck` reads in the next free list link block and repeats the process.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the free list, `fsck` may rebuild the list, excluding all blocks in the list of allocated blocks.

The superblock also contains a count of the total number of free blocks within the file system. `fsck` compares this count to the number of

blocks it found free within the file system. If the counts do not agree, `fsck` may replace the count in the superblock by the actual free-block count.

Phase 5 of the `fsck` program lists error conditions resulting from bad blocks in the free list, bad free-block count, duplicate blocks in the free list, unused blocks from the file system not in the free list, and the total free-block count incorrect.

`EXCESSIVE BAD BLKS IN FREE LIST`
`CONTINUE?`

The free list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system.

Possible responses to the `CONTINUE` prompt are

`Y` Ignore the rest of the free list and continue execution of `fsck`. This error condition will always invoke the "`BAD BLKS IN FREE LIST`" error condition in Phase 5.

`N` Terminate `fsck`.

`EXCESSIVE DUP BLKS IN FREE LIST`
`CONTINUE?`

The free list contains more than a tolerable number (usually 10) of blocks claimed by inodes or earlier parts of the free list.

Possible responses to the `CONTINUE` prompt are

`Y` Ignore the rest of the free list and continue execution of `fsck`. This error condition will always invoke the "`DUP BLKS IN FREE LIST`" error condition in Phase 5.

`N` Terminate `fsck`.

`BAD FREEBLK COUNT`

The count of free blocks in a free list link block is greater than 50 or less than 0. This error condition will always invoke the "`BAD FREE LIST`" condition in Phase 5.

x BAD BLKS IN FREE LIST

x blocks in the free list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

x DUP BLKS IN FREE LIST

x blocks claimed by inodes or earlier parts of the free list were found in the free list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

x BLK(S) MISSING

x blocks unused by the file system were not found in the free list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

FREE BLK COUNT WRONG IN SUPERBLK
FIX?

The actual count of free blocks does not match the count in the superblock of the file system.

Possible responses to the FIX prompt are

Y Replace the count in the superblock by the actual count.

N Ignore this error condition.

BAD FREE LIST
SALVAGE?

Phase 5 has found bad blocks in the free list, duplicate blocks in the free list, or blocks missing from the file system. If the –q option is specified, the free list will be salvaged automatically.

Possible responses to the SALVAGE prompt are

Y Replace the actual free list with a new free list. The new free list will be ordered to reduce time spent by the disk waiting for the disk to rotate into position.

N Ignore this error condition.

5.8 Phase 6: Salvage free list

This phase concerns itself with the free list reconstruction. This part lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

`Default free list spacing assumed`

This is an advisory message indicating the blocks-to-skip value is greater than the blocks-per-cylinder value, the blocks-to-skip value is less than 1, the blocks-per-cylinder value is less than 1, or the blocks-per-cylinder value is greater than 500. The default values of 9 blocks-to-skip and 400 blocks-per-cylinder are used. See `fsck`(1M) in *A/UX System Administrator's Reference* for further details.

5.9 Cleanup

Once a file system has been checked, a few cleanup functions are performed. This part lists advisory messages about the file system and modify status of the file system.

x `files` *y* `blocks` *z* `free`

This is an advisory message indicating that the file system checked contained *x* files using *y* blocks leaving *z* blocks free in the file system.

`***** Fixed root, rebooting Unix! *****`

This message indicates that `fsck` has modified the root file system to fix inconsistencies which it found. `fsck` forces a reboot because it can only fix the file system image on the disk but not in memory. Since A/UX periodically issues a `sync`(1) call to update the the file system from memory, the forced reboot of the system prevents the file system repair from being undone by the automatic `sync` call.

`***** FILE SYSTEM WAS MODIFIED *****`

This advisory message indicates that `fsck` has modified a non-root file system to fix the inconsistencies which it found. A/UX will continue to run.

Chapter 7
System Accounting Package

Contents

Chapter 7

System Accounting Package

1. Introduction

Two packages provided with your A/UX system allow you to keep track of the details of system operation and usage:

- The **system accounting package** collects information and generates reports on buffer activity, CPU utilization in general, device activity, and so on. This chapter discusses the system accounting package.
- The **system activity package** permits you to keep track of all low-level activity in your system. For information on the system activity package, see Chapter 8, "System Activity Package."

The system accounting package collects detailed information on system usage and allows you to generate a comprehensive system usage report on a daily and monthly basis. (Note that the report provides information on overall system usage, not on individual users.) This "report" function, for the most part, is automatic and is described in "Routine Accounting Procedures." You can also produce customized reports using accounting commands; these are described in "Special Accounting Procedures."

2. Routine accounting procedures

The system uses these procedures to generate information describing what a user is doing and how often the user invokes a specific command. It also generates information on overall system usage, frequency of usage, and system resource allocation.

To turn on system accounting and automate its operation,

1. Log in as `root`.
2. Make sure the following lines are in the `/etc/rc` file and are not commented out (if they are preceded by a number sign (#),

remove it):

```
/bin/su adm -c /usr/lib/acct/startup
echo process accounting started
```

The first line is a command that activates the accounting system when the system is brought up. The second line prints "`process accounting started`" when the accounting system starts up.

3. Make sure that the following lines in the `/usr/spool/cron/crontabs/adm` file are not commented out (if they are preceded by a number sign (#), remove it):

```
0 4 * * 1-6 /usr/lib/acct/runacct
                    2> /usr/adm/acct/nite/fd2log
0 2 * * 4 /usr/lib/acct/dodisk
5 * * * * /usr/lib/acct/ckpacct
15 5 1 * * /usr/lib/acct/monacct
0 * * * 0,6 /usr/lib/sa/sa1
0 18-7 * * 1-5 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
0 20 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e
                        18:00 -i 3600 -uybd
```

Note: The first two lines and the last two lines in the above example *must* appear as one line in the `adm` file. These lines have been split in two only for formatting reasons.

The `adm` file instructs `cron` to automatically run the daily accounting. The line:

```
15 5 1 * * /usr/lib/acct/monacct
```

uses the `monacct` file to clean up all daily reports and daily total accounting files. The `adm` file also deposits one monthly total report and one monthly total accounting file in the `fiscal` directory after `runacct` has completed the last day's entries. By default, `monacct` uses the current month's date as the suffix

for the filenames.

4. Make sure that the following line is in the `/usr/adm/.profile` file:

   ```
   PATH=/usr/lib/acct:/bin:/usr/bin
   ```

   This ensures that the operating system has access to all the necessary files and scripts to process the accounting system automatically.

2.1 The `cron` file

Although you don't need to understand how the `cron` file works to use the accounting package, knowing how the entries are written and what they do makes it easy to modify entries if you need to. This section is a quick introduction to `cron`.

`cron` is a program that performs specified functions at specified times. These functions and times are specified in the file `/usr/spool/cron/crontabs/adm`, which is read and acted upon by `cron`. Each line of text in this file is a small script that consists of six fields that together specify a process to be executed at a certain time. The fields are separated by spaces or tabs. The first five fields specify a time. They specify, in order,

minute	Time in minutes (0–59)
hour	Time in hours (0–23)
day-of-month	Day of the month (1–31)
month-of-year	Month of the year (1–12)
day-of-week	Day of the week (0–6, with 0 as Sunday)

Each of these may be either an asterisk (which means ignore all cases) or a list of elements separated by commas (which specify specific cases to be activated). An element may be one number or two numbers separated by a hyphen (meaning an inclusive range).

You specify days in two fields: *day-of-month* and *day-of-week*. If you specify both, both are used. To specify days using only one field, set the other to `*`.

The sixth field is the process to be executed. A percent sign in this field (unless masked by \\) is translated as the signal for a new line. The shell executes only the first line (up to the % or the end of the line).

In this example, the line

```
15 5 1 * * /usr/lib/acct/monacct
```

translates to the following: at the 15th minute of the 5th hour on the first day of each month, run the `monacct` program, which is located in the `/usr/lib/acct` directory. The asterisks in the *month-of-year* and the *day-of-week* fields tell the system to ignore these fields.

You can change the field entries in the file `/usr/spool/cron/crontabs/adm` to run the accounting procedures more frequently or even print a weekly report instead of a monthly one.

2.2 Updating holidays

`cron` knows when to run the requested procedures because it is governed by the system's internal clock. The file `/usr/lib/acct/holidays` contains the prime/nonprime table, which gives the start of prime time and the start of nonprime time for your operating system, using a 24-hour system (0100 to 2400 hours). Note that the hour 2400 automatically converts to 0000. Information on changing the prime/nonprime table to reflect individual preferences is given below. For example, during normal business operation, prime time is considered to be from 8:30 a.m. to 5:30 p.m. It also contains the holiday schedule for the year, which you can adjust to include additional holidays.

The format of the `holidays` file includes

Comment lines
: Comment lines can appear anywhere in the file but they *must* be preceded by an asterisk so the system won't read them.

Year designation line
: The year designation line *must* be the first data line in the file and can appear only once in the file. It consists of three fields of four digits each:

yyyy hhmm hhmm (number of spaces irrelevant)

- The first field is the current year. Be sure to set it correctly, because an incorrect year will affect the holiday entries.
- The second field is the start of prime time, in 24-hour time. Prime time refers to the peak activity period for your system. In most businesses, for example, prime time would start at 8:30 a.m. (0830 in a 24-hour system) to coincide with the business day.
- The third field is the start of nonprime time, in 24-hour time. This field represents the end of peak activity for the operating system, usually 5:30 p.m. (or 1730).
- Your system may be set to start prime time at 0800 and nonprime time at 1700 (5:00 p.m). If your company begins work at 8:00 p.m., for example, you may want to change the start of prime time to 2000 to reflect this. You would also change the nonprime time to reflect the close of business, say 2:00 a.m., as 0200.

Note: The hour 2400 automatically converts to 0000.

Company holiday lines

You can make entries for national and local holidays on the line following the year designation line. The format is

day-of-year month day description-of-holiday

The day of the year is a number from 1 to 366 indicating the day for the corresponding holiday. The other three fields are not used by A/UX and are provided for commentary only.

Note: Remember to separate all entries with tabs. If you include spaces, the program will not run.

2.3 Daily operation

The lines of text you enter in the `cron` file cause the system to automatically run the `startup`, `ckpacct`, `turnacct`, `dodisk`, `runacct`, and `monacct` procedures. Each of these procedures is described in this section. These are only some of the procedures

available. Additional procedures are also described, and you can add them to the /usr/spool/cron/crontabs/adm file to be processed automatically. For example, acctcom is discussed in "Special Accounting Procedures."

When you bring the system into multiuser mode, /usr/lib/acct/startup is executed. This does three things:

- The acctwtmp program records a "boot" in the /etc/wtmp file. It uses your system name as the login name in the file.
- Turnacct begins the process accounting. The accton program is executed and the collected data is stored in the /usr/adm/pacct file. This file is later read by the reporting function and summarized in the daily and monthly reports.
- The remove shell procedure is executed. This cleans up the saved pacct and wtmp files that runacct creates.

2.3.1 The ckpacct procedure

After process accounting has begun, cron executes the ckpacct procedure every hour. This procedure checks the size of the pacct file. ckpacct begins the process to create multiple pacct files when the file grows to 1000 blocks. It executes turnacct switch, which turns the process accounting off, moves the current /usr/adm/pacct data to /usr/adm/pacct/*incr* (where *incr* is a number that starts with 1 and increases by one for each additional pacct file turnaccct creates), and then turns the process accounting back on. This keeps the pacct files to a reasonable size. If you ever need to restart runacct, the smaller file size makes the job easier.

2.3.2 The dodisk procedure

cron invokes the dodisk procedure to perform the disk accounting functions. The command is structured as follows:

```
/usr/lib/acct/dodisk [-o][file1...]
```

If you don't specify any options (the default), it will do disk accounting on the files in /etc/checklist, which is a list of all file systems in the disk partition.

If you use the `-o` flag, a slower version of disk accounting by login directory will be done.

file1 specifies the file system name(s) where the disk accounting will be done. If you use files, disk accounting will be done on these file systems only. If you use the `-o` flag, files should be the names of the directories on which the file systems are mounted. If you omit the `-o` flag, files should be the special filenames of mountable file systems. See Chapter 6, "Checking the A/UX File System: `fsck`."

2.3.3 The `chargefee` procedure

You can invoke the `chargefee` shell procedure to charge a number of units to a login name. The command syntax is

```
/usr/lib/acct/chargefee login-name number
```

For example, you charge login name `john` for two units (for $2.00) for his system usage. This information is then written to `/usr/adm/fee` and merged with the other accounting records during the night. This information will then appear in the `FEE` column in the daily report generated by `runacct`.

2.3.4 The `runacct` procedure

`runacct` is the main daily accounting shell procedure. This section covers the `runacct` command itself, the error messages it generates, and how to recover if the procedure fails.

The `runacct` command has the following form:

```
/usr/lib/acct/runacct [mmdd][mmdd state]
```

You can use the options to restart `runacct` after a failure. They are explained later in the section.

The entries made to the `/usr/spool/cron/crontabs/adm` file initiate the `runacct` procedure during nonprime-time hours. `runacct` automatically processes the connect, fee, disk, and process accounting files and prepares daily and cumulative summary files, which are then read by `prdaily` or used for billing purposes. When you run `monacct`, these daily reports are summarized into a monthly report.

To read the daily report, type

```
/usr/lib/acct/prdaily | more
```

The `prdaily` reporting function is covered in more detail later in this chapter. The above command prints a report that was generated by the `runacct` procedure. The report consists of a header and five parts.

The header displays the dates of the current reporting period, for example:

```
Oct 26 10:04 1987  DAILY REPORT FOR A/UX Page 1

from Tue Oct 20 04:00:13 1987
to   Tue Oct 20 04:00:13 1987
```

Following the date is a listing of the `/etc/wtmp` entries generated by the `acctwtmp` program. This listing includes any reboots, shutdowns, power fail recoveries, date changes, and so on during the reporting period:

```
2   date changes
```

Commands used to produce the report are displayed:

```
1    runacct
1    acctcon1
```

The first part of the report describes the connect accounting information on terminal usage, the number of sessions (logins, logouts), and the amount of time each terminal was used. The fields in this part of the report are as follows:

`TOTAL DURATION`
: The amount of time the system was in multiuser mode.

`LINE`
: The terminal line or access port used.

`MINUTES`
: The amount of time the line was in use during the reporting period covered in the report.

PERCENT
The number of MINUTES divided by TOTAL DURATION.

SESS, # ON
These columns give the number of logins on the line during the reporting period. This report is helpful in finding which lines have been logged on, but not off.

OFF
Not only the number of logoffs, but also the number of interrupts on the line. If the # OFF exceeds the # ON by a large factor, it is possible that there is a bad connection, or the multiplexer, modem, or cable is going bad. You should monitor the /etc/wtmp file. If it grows rapidly, execute acctcon1 to see which TTY line is the noisiest. A high number of interrupts can adversely affect the system.

The next part of the report provides a breakdown of system resource utilization by user. It does not give specific information on what each user was doing, but provides information on the CPU and connect time for each user. To charge users for system usage, see the information in the FEE column and the "The chargefee Procedure" in this chapter.

UID The user ID (uid) for each login. For more information on uid, see Chapter 3, "User Administration."

LOGIN NAME
The actual user login name. This column helps you differentiate the activities of different users if there are more than one per uid. The next column gives CPU usage. It is broken down into two amounts: prime-time usage and nonprime-time usage.

KCORE-MINS
A cumulative measure of the amount of memory a process uses. It is measured in kilobytes per minute and is broken down into prime-time and nonprime-time usage.

CONNECT (MINS)
The amount of time a user was logged in. It is measured in "real time" and is broken down into prime-time and nonprime-time

usage. If this column is high, and the # OF PROCS column is low, the user probably logs in first thing in the morning and then hardly uses the terminal.

DISK BLOCKS
: An accounting of the disk usage by user as it is calculated by the acctdusg program. The # OF PROCS column gives the number of processes used. A very high number might indicate a shell process gone wild. The # OF SESS tells you how many times each user logged in.

DISK SAMPLES
: The number of times the acctdusg ran in order to obtain the information in the DISK BLOCKS column.

FEE
: A record of disk charges to each user ID. This information is written to /usr/adm/fee and merged with the other accounting records during the night. It will then appear in the FEE column in the daily report.

The next two parts of the report, the DAILY COMMAND SUMMARY and MONTHLY COMMAND SUMMARY, summarize command usage over the report period. The report period is specified using the *number* argument to the monacct command; if not specified it defaults to the current month. These parts are identical in format. The daily report summarizes the command usage for the report period, and the monthly report summarizes the command usage from the beginning of the month through the current period. Entries may appear in the monthly command name column that do not appear in the daily report. These are commands that were used sometime during the current month but not during the current reporting period. You can fine-tune the system by making commonly used commands more accessible.

The columns and their output are defined as

DAILY, MONTHLY, and TOTAL KCOREMIN
: Both the DAILY and MONTHLY command summaries are sorted by the TOTAL KCOREMIN column. This provides the total amount of memory a process uses per minute, measured in kilobytes.

COMMAND NAME

Self-explanatory. All shell commands, however, are lumped under the entry `sh`. For example, the command `cd` won't appear in the report; it is included in the `sh` column.

It is important to note entries such as `a.out`, `core`, or mysterious command names. `a.out` and `core` indicate errors in compiled programs. If a compiled program is not named, it will appear in the report under the default name `a.out`. `core` indicates errors in the execution of a compiled program. You can use `acctcom` to tell you who used a suspicious command, perhaps an alias, or who has been exercising superuser privileges. See "Special Accounting Procedures" later in this chapter for more information.

NUMBER CMNDS

Total number of times a command was executed.

TOTAL CPU-MIN

Total processing time a command was used.

TOTAL REAL-MIN

Amount of time it takes to process the command in real time. It includes the real-time usage of background processes as well.

MEAN SIZE-K

Calculated as TOTAL KCOREMIN over NUMBER CMNDS.

MEAN CPU-MIN

Calculated as NUMBER CMNDS over the TOTAL CPU-MIN used to execute the commands.

HOG FACTOR

The ratio of system availability to system utilization. It is calculated by dividing the total CPU time by the elapsed time. It provides a relative measure of the CPU time the process used during its execution.

CHARS TRNSFD

Gives the total number of characters pushed around by the read and write system calls. The number of characters is calculated, command by command. It can be a negative number; the reads

may outnumber the writes, for example.

`BLOCKS READ`
: A total count of the physical block reads and writes that a process performs.

The last part of the accounting report is a compilation of all the logins on the system and the last date they logged in. The report looks like this:

```
Sep 29 04:05 1986  LAST LOGIN Page 1

86-09-25 apple
86-09-27 alice
00-00-00 phil
00-00-00 sys
86-09-29 john
```

The first column gives the date of the last login in *yy-mm-dd* format. The second column gives the login name itself. As you can see from the example, the date information can be blank. If the system is shut down for any reason, the last login date for all users who have not logged on since the shutdown will read `00-00-00`. You can use this part of the report to determine which users are no longer active. These users (excluding those who may have logged on prior to a crash, but not since) may be candidates for removal.

2.3.5 The `prdaily` procedure

`prdaily` is the script that generates a printout of the `runacct` process. The report resides in `/usr/adm/acct/sum/rprt`*mmdd*. The command syntax is

```
/usr/lib/acct/prdaily [-l][-c][mmdd]
```

The notation *mmdd* indicates the month and day of the report. You can generate previous daily reports using this option, specifying the month and day of the data you wish to see. Note that the report generated on `0611`, for example, is actually the report on usage for `0610`. Remember, the daily information is no longer available after `monacct` is run. The `-l` flag prints a report of exceptional usage by login identification for a date specified with *mmdd*. The `-c` flag prints a report of exceptional resource usage by command. You can use it on

the current day's accounting data only.

2.4 Restarting runacct

runacct has been designed to try to recognize possible errors and print warnings before terminating the process. During processing, messages are written in the /usr/adm/acct/nite/active file to inform the operator of successful completion of the various phases of the procedure.

Diagnostics are written into the fd2log (all runacct files are located in the /usr/adm/acct/nite directory unless otherwise specified). runacct will inform you if lock or lock1 exists. To prevent generation of more than one report for a day, the lastday file keeps a record of the month and day the program was last run.

runacct is careful not to damage active accounting or summary files. It records its progress by writing messages into the file /usr/adm/acct/nite/active. When it detects an error, it writes a message to the console, sends mail to root and adm, and then terminates.

runacct uses a series of lock files to prevent reinvocation of the accounting process until the errors have been corrected. It uses the files lock and lock1 to prevent simultaneous invocation; the file lastdate prevents more than one invocation per day.

2.4.1 In case runacct fails

If you must restart runacct after a failure, you can begin by following these steps.

1. Check for diagnostic error messages in the active*mmdd* file located in the /usr/adm/acct/nite directory. If this file contains error messages and the lock files exist, check the fd2log for unusual messages.
2. Fix up any corrupted data files such as pacct or wtmp.
3. Remove the lock, lock1, and lastdate files if they are present.

If runacct cannot complete the procedure for any reason (lock file encountered, error encountered, or the like), a message is written to the

console (with copies sent via mail to `root` and `adm`), locks are removed, diagnostic files are saved, and the process is terminated. If you review the messages in the active file and at the console or in mail, you can determine at which point the process was stopped and why. You can then restart the process at the appropriate location and let it finish.

To make it easier to recover from errors, `runacct` is broken down into separate, restartable states. Under ordinary circumstances, the state's name is written into `statefile` as each state is completed. `runacct` then checks `statefile` to see what has been done and to determine what state to process next. States are executed in the following order:

`SETUP`
: Moves active accounting files into working files.

`WTMPFIX`
: Verifies the integrity of the `wtmp` file and corrects date changes if necessary.

`CONNECT1`
: Produces connect session records in `ctmp.h` format.

`CONNECT2`
: Converts `ctmp.h` format files into `tacct.h` format.

`PROCESS`
: Converts process accounting records into `tacct.h` format.

`MERGE`
: Merges the connect and process accounting records.

`FEES`
: Converts the output of `chargefee` into `tacct.h` format and merges it with the connect and process accounting records.

`DISK`
: Merges the disk accounting records with connect, process, and fee accounting records.

`MERGETACCT`
: Merges the daily total accounting records in `daytacct` with the

summary total accounting records in `/usr/adm/acct/sum/tacct`.

`CMS`
: Produces the command summaries.

`USEREXIT`
: You can include customized accounting procedures here.

`CLEANUP`
: Cleans up the temporary files and exits.

`runacct` will begin processing with the next state in `statefile`. If you want to begin processing at another state, include the desired state on the command line to designate where processing should begin. You must also include the argument *mmdd*, specifying the month and day for which `runacct` will rerun the accounting process.

For example,

```
nohup runacct 0601 2>>/usr/adm/acct/nite/fd2log&
```

restarts `runacct` on June 1 at the next state in `statefile`. If you enter

```
nohup runacct 0601 MERGE 2 >>/usr/adm/acct/nite/fd2log&
```

`runacct` restarts on June 1 at the Merge state.

Normally it is not a good idea to restart `runacct` in the Setup state. Run `SETUP` manually and restart via

```
runacct mmdd WTMPFIX
```

If `runacct` failed in the Process state, be sure to remove the last `ptacct` file because it will not be complete.

2.4.2 Error messages

If `runacct` is terminated, error messages are written into the `active`*mmdd* file in the `/usr/adm/acct/nite` directory. If this file and the lock files exist, check `fd2log` for unusual messages.

The following are some common error messages and possible solutions. The list is by no means complete.

`ERROR: locks found, run aborted`
The files `lock` and `lock1` were found. You must remove them to restart `runacct`.

`ERROR: acctg already run for` *`date`*`: check /usr/adm/acct/nite/lastdate`

Today's date is the same as the last entry in `lastdate`; remove the last entry in `lastdate`.

`ERROR: turnacct switch returned rc= ?`
Check the integrity of `turnacct` and `accton`.

Note: The `accton` program must be owned by `root` and have the setuid bit set.

`ERROR: Spacct?.`*`mmdd`*` already exists`
File setups probably have already been run. Check the status of files and run setups manually.

`ERROR: /usr/adm/acct/nite/wtmp.`*`mmdd`*` already exists. Run setups manually.`
File setups have probably already been run. Check the status of files and run setups manually.

`ERROR: wtmpfix errors see /usr/adm/act/nite/wtmperror`

A corrupted `wtmp` file exists. Use `fwtmp` to correct it.

`ERROR: connect acctg failed: check /usr/adm/acct/nite/log`

The `acctcon1` program encountered a bad `wtmp` file. Use `fwtmp` to correct it.

`ERROR: Invalid state, check /usr/adm/acct/nite/active`

The `statefile` is probably corrupted. Check it and read the active file before restarting.

2.4.3 Fixing corrupted files

When it is necessary to restart `runacct`, some of the files may need to be recreated before proceeding. You can ignore some, and you can restore others from backups. Some files, however, *must* be fixed.

2.4.4 Fixing `wtmp` errors

If the date is changed during multiuser mode, a set of date change records is written into `/etc/wtmp`. The `wtmpfix` program is designed to modify the time stamps in the `wtmp` files when this happens. If there has been a combination of date changes and reboots, the `wtmpfix` program might not work, causing `acctcon1` to fail.

If this happens, you should make the following adjustment:

```
cd /usr/adm/acct/nite
fwtmp < wtmp.mmdd > xwtmp
ed xwtmp
   delete corrupted records or
   delete all records from beginning up to date change
fwtmp -ic < xwtmp > wtmp.mmdd
```

If you can't fix the `wtmp` file, create a null `wtmp` file, which will prevent connect time from being charged incorrectly. `actprc1` will not be able to determine which login used a specific process; it will charge the process to the first login in that user's password file.

2.4.5 Fixing `tacct` errors

If you are using the accounting system to charge users for system usage, you must maintain the integrity of the `tacct` file in the `/usr/adm/acct/sum` directory.

If `tacct` records have negative numbers, duplicate user IDs, or a user ID of 65,535, the file may be corrupted. First, check `sum/tacctprev` with `prtacct`. If it looks all right, patch up the latest `sum/tacct.`*mmdd*, then recreate `sum/tacct`.

A sample patchup follows:

```
cd /usr/adm/acct/sum
acctmerg -v < tacct.mmdd > xtacct
ed xtacct
    remove the bad records
    write duplicate uid records to another file
acctmerg -i < xtacct > tacct.mmdd
acctmerg tacctprev < tacct.mmdd > tacct
```

Note: You can recreate `sum/tacct` by merging all the `tacct.`*mmdd* files (the `monacct` procedure does this).

2.4.6 The `monacct` procedure

The monthly accounting summary is another automated procedure in the accounting package. You should invoke `monacct` once each month or once each accounting period. The line in the `cron` file

```
15 5 1 * * /usr/lib/acct/monacct
```

causes `monacct` to be invoked once per month (see "The `cron` File" earlier in this chapter.)

When run from the command line, the form of the `monacct` command is

```
/usr/lib/acct/monacct [number]
```

where *number* indicates which month or period it is. You can specify the week (01–52), the month (01–12), or the fiscal period, such as quarters (01–04). If you don't give any argument, `monacct` defaults to the current month. `monacct` creates summary files in `/usr/adm/acct/fiscal` and restarts summary files in `/usr/adm/acct/sum`.

3. Special accounting procedures: `acctcom`

Besides the user information you can get from the automated procedures, there is additional information available on users. For example, the automated procedure does not tell you who is doing what, or when. One way to gather this information is by implementing additional accounting commands supplied with your system.

One such command is the `acctcom` command, described in the next section.

3.1 The `acctcom` command

The `acctcom` command is the most useful accounting command supplied with your system. It reports what processes are associated with a particular terminal, user, or group of users.

The command syntax of `acctcom` is

```
acctcom [options][file]
```

`acctcom` reads a specified file, the standard input, or `/usr/adm/pacct`, and writes the selected records to the standard output. Each record represents the execution of one process.

If you don't specify a file, and standard input is associated with a terminal, `/usr/adm/pacct` is read. If this isn't the case, the standard input is read. If *file* arguments are given, they are read in the order given. Each individual file is read in chronological order by process completion time. The `/usr/adm/pacct` file is usually the current file to be examined. A busy system, however, may have several `pacct` files to be processed.

The output generated by this command is similar in format to the report generated by `runacct`. It shows the following:

`COMMAND NAME`
: The command name is preceded by a number sign # if it was executed with superuser privileges. By using the `-n` option of the command, you can find out which users (selected with the `-u` option) are executing the commands.

`USER NAME`
: The login name of the user.

`TTY NAME`
: The terminal device associated with the process. If a process is not associated with a known terminal, a ? appears in this column.

`START TIME`
: The time the process began.

`END TIME`
The time the process terminated.

`REAL (SEC)`
The elapsed real time the process took to complete.

`CPU (SEC)`
The elapsed CPU time the process took to complete.

`MEAN SIZE (K)`
The average amount of memory (in kilobytes) used by a process over the number of invocations of the process.

`STAT`
The system exit status.

`HOG FACTOR`
Ratio of system availability to system utilization. It is calculated as total CPU time over elapsed time.

`KCORE MIN`
The amount of kilobyte segments of memory used by a process.

`CPU FACTOR`
A measurement of user time over system time plus user time.

`CHARS TRNSFD`
The number of characters transferred by the `read` and `write` commands.

`BLOCKS READ`
A total count of the physical block reads and writes that a process performed.

The `acctcom` command has several options. The options are given below with a brief explanation of what each does. Try a few out and see which ones are best suited to your purposes. Pay particular attention to the `-u` option, which describes user usage of the system. For a full listing of all the options, see `acctcom`(1M) in *A/UX System Administrator's Reference*.

`-a` The main `acctcom` option. In addition to the column headings listed above, it prints some average statistics about the processes

selected.

`-p` Prints the statistical information that appears at the end of the `-a` report without printing the report itself. The report output appears similar to this:

```
cmds=2218 Real=147.68 CPU=5.64 USER=4.33 SYS=1.30
CHAR=37042.86 BLK=43.71    USR/TOT=0.77 HOG=0.04
```

`-f` Prints a report with columns showing the number of `fork/exec` flags and the system exit status.

`-h` Displays the fraction of total available CPU time consumed by the process during its execution in a column titled `HOG FACTOR`.

`-l` Shows only those processes associated with a particular line. The option requires the argument *line*, to specify which line you wish to observe.

`-u` Gives you a report of all system usage by a particular login name. The option requires the argument *user*, which specifies the login name about which you wish to generate a report. You can use it in conjunction with the `-s`, `-e`, `-S`, and `-E` options to limit the search to a specific time period.

`-g` Similar to the `-u` option. Instead of printing system usage by user, however, it prints usage by the group. It requires the argument *group*, which may be either the group name or group ID.

`-s, -S, -e, -E`
Limit the reported information to processes that occur by a specified time. These options can be used with the other options to specify a range of time in which the activities are to be reported on. They require the argument

hr[: *min*[: *sec*]]

`-s` selects processes existing at or after *time*.

`-S` selects processes starting at or after *time*.

`-e` selects processes existing at or before *time*.

`-E` selects processes ending at or before *time*.

Chapter 8
System Activity Package

Contents

Chapter 8

System Activity Package

1. Introduction

Two packages provided with your A/UX system allow you to keep track of the details of system operation and usage:

- The **system accounting package** collects information and generates reports on buffer activity, CPU utilization in general, device activity, and so on. For information on the system accounting package, see Chapter 7, "System Accounting Package."
- The **system activity package** permits you to keep track of all low-level activity in your system. This chapter discusses the system activity package.

The system activity package provides features for collecting data about the low-level functioning of your system. You can use commands to get information on low-level system activity and to generate reports that summarize this activity. This is accomplished by using various counters to monitor kernel activity. The counters are sampled regularly and the data saved in binary format. This data is then used to generate ASCII reports. The information can help you to determine if and where the system may need fine-tuning. You can view the output from these commands immediately or redirect it to a file for future use.

The system activity commands rely on a series of **activity counters** to gather and sample data and generate a report on system activity. These counters will be described in conjunction with the commands that use them to generate reports.

2. The system activity counters

A series of system activity counters must be working to determine what processes are being run at any given time. These counters, which are located in the operating system kernel, record various activities at

selected times. For example, you can set them to record all processes used at 8:00 a.m. Monday and store the information in a file for review later.

There are several types of counters. Each counter generates various pieces of information, but the same counter may be used by different commands to provide different information. The **CPU counter,** for instance, reports the prime- and nonprime-time usage in minutes in the accounting report (see Chapter 7, "System Accounting Package"), while in the activity report the same counter reports the state(s) the CPU is in: idle, user, kernel, or wait.

In this chapter, each type of counter is explained as it first occurs in relation to the system command that invokes it. Most are explained in relation to the `sar` command, as this is the most useful system activity command. Particular emphasis is placed on those counters that you can use to troubleshoot or fine-tune your system.

If you would like more detailed information about the counters, keep the following in mind:

- The data structure for most counters is defined in the `sysinfo` structure in `/usr/include/sys/sysinfo.h`.
- The system table overflow counters are kept in the `_syserr` structure.
- The device activity counters come from the `device status tables`.

3. The system activity data collector

The system activity data collector (`sadc`) is the automated data collection feature supplied with your system. Two shell scripts, `sa1` and `sa2`, assemble the data for the reporting functions.

3.1 The `sadc` command

This command is used to collect system activity information at regular intervals. It is an executable program that reads the system counters located in `/dev/kmem`, and records them in binary format in a file for later sampling by the system activity reporting functions.

The command `sadc` can be used alone or with arguments. It has the format

```
/usr/lib/sa/sadc [t n] [file]
```

When used without arguments, `sadc` sets the startup time. It creates a special record that tells the system to reset the system counters to zero. The same thing occurs when the system is rebooted. When the arguments *t* and *n* are used, `sadc` samples the counters *n* times every *t* seconds and writes the data in binary format to the named *file* or, by default, to the standard output, `/usr/adm/sa/sa`*dd*, where *dd* stands for a given day.

3.2 The `sa1` and `sa2` commands

`sa1` and `sa2` supply the default parameters for the operation of `sadc` and `sar`.

`sa1` invokes `sadc` to enter the information gathered by the system counters at the intervals specified in the `/usr/lib/cron/crontab` file into the daily data file `/usr/adm/sa/sa`*dd*. This information is in binary format.

`sa2` is a variation of the `sar` command. It reads the data file created by `sa1` and uses it to generate a report in ASCII format. This report is stored in the `/usr/adm/sa` directory in the files named `sar`*dd*. Again, *dd* stands for the day of the report. You can use the `sar` options with the `sa2` command. For a complete explanation of these options, see " The `sar` Command Options" later in this chapter.

4. Setting up the system activity functions

It is a good idea to monitor and record system activity routinely in a standard way for historical analysis. Each of the previous commands initiates activity observations. For these observations to occur, you must first initiate the data collection functions. By automating these functions, you can generate regular system activity reports.

To have your system automatically sample the activity counters and store the information in a file for later reporting, you should add the following lines to the `/usr/spool/cron/crontabs/adm` file.

```
0 * * * 0,6 /usr/lib/sa/sa1
0 18-7 * * 1-5 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
```

This produces records every 20 minutes during working hours and hourly otherwise.

You can generate hourly records during working hours by substituting the following for the second line above.

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i3600 -A
```

These entries will cause the collection functions to operate. They will not, however, generate reports automatically. It is still the responsibility of the system administrator to check the desired reporting function regularly and to generate the report for current and later review.

5. The system activity report commands

The three commands `sar`, `sag`, and `timex` generate system activity reports.

You can use these commands to observe system activity during

- normal operations
- a controlled stand-alone test of a large system
- an uncontrolled run of a program to observe the operating environment

The `sar` command generates system activity reports in real time and saves the output in a file.

The `sag` command displays system activity in graph form.

The `timex` command is a modified `time` command that reports how long a given command takes to execute, and the user time and system time spent in execution of the command.

These commands and their options and applications are discussed in detail below.

5.1 The `sar` command

`sar` is the system activity reporter command. You can use the `sar` command alone or with a series of options. If you type

```
sar
```

a report on today's CPU activity scrolls across the screen. The output should look like this:

```
00:00:03   %usr   %sys   %wio   %idle
01:00:03     1      1      0      98
02:00:03     1      1      0      98
14:39:12    41     12      4      42
14:59:12     3      5      3      89
15:20:04    14     12      4      70
15:40:04    11      8      3      78

Average      5      5      1      89
```

If you type

```
sar > sar.output
```

the output from `sar` is stored in the file `sar.output` and doesn't scroll across your screen.

By typing `sar` with the options discussed in the next section, you can sample various counters to view activity at specific times and intervals. By redirecting the output to a file, you can save the information for later review.

The full `sar` command syntax is written in one of these two ways:

`sar [-u] [-b] [-y] [-c] [-w] [-a] [-q] [-v] [-m] [-A] [-o`*file*`] t [`*n*`]`

`sar [-u] [-b] [-y] [-c] [-w] [-a] [-q] [-v] [-m] [-A] [-s`*time*`] [-e`*time*`] [-i`*sec*`] [-f`*file*`]`

The first word is the command itself; the letters within the brackets are flag options that sample different counters. The remaining options allow you to sample certain counters at specified times and save the result for later viewing.

In the first command syntax, `sar` extracts the data from a previously recorded file (which you specify with the `-f` *file* option) or, by default, the standard system activity daily data file `/usr/adm/sa/sa`*dd*, where *dd* stands for a given day. This file appears unreadable when you view it because it is stored in internal format that the system reporting functions read to prepare reports. The reports themselves give you the information in a readable format.

You can specify the starting and ending times of the report by invoking the `-s` *time* and `-e` *time* options, using the format *hh*[: *mm*[: *ss*]].

The `-i` option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported. For example,

```
sar -s8:00 -e18:01 -i3600
```

will sample activity at intervals of 3600 seconds, that is, every hour starting at 8:00 a.m. and ending at 6:01 p.m.

In the second command syntax above, `sar` invokes the data collection program `sadc` to sample the system activity counters every *t* seconds for *n* intervals and generates a system activity report.

If you specify the `-o` option, `sar` saves the output in binary format into *file*.

If you don't supply any frequency arguments, `sar` generates system activity reports for the time interval specified in the existing data file. On this system, that time period is from 8:00 to 18:00 and the data is sampled every hour (see "Setting Up the System Activity Functions"). Unless you redirect the output to a file, it scrolls across the screen.

> *Note:* All reports you generate with the options listed print a time stamp for each entry. It appears in the first column of each report section in the format *hh* : *mm* : *ss*.

When you use the `sar` command without options, it generates a report on CPU activity only. This is the same report you receive if you type

```
sar -u
```

The system activity package automatically generates a report containing all of the options listed below. It is stored in the `/usr/adm/sa` directory in the files listed as `sar`*dd*. These files are the binary representations of the `sar` reports. You can get the same report as `sar`*dd* by typing

```
sar -A
```

This generates the same information as

```
sar -udqbwcayvm
```

but takes fewer keystrokes.

5.2 The `sar` command options

The options for the `sar` command are listed below.

`-u` Report CPU utilization. This is the default option, the option the system selects if none is specified. If you type

```
sar -u
```

the following report is displayed:

```
07:00:02   %usr   %sys   %wio   %idle
08:00:04     0      1      0      99
09:00:02    10      7      3      80
11:00:07     8      8      4      81
12:00:05    15     10      4      71
13:00:03    12     10      2      76
15:00:08    31     16      4      48
16:00:06    39     15      4      43
18:00:02     1      1      0      97
19:00:02     0      1      0      99

Average     12      8      2      78
```

The column headings display the following:

`%usr` amount of time running in user mode
`%sys` amount of time running in system mode
`%wio` idle time with process waiting for block I/O

`%idle` idle time
`Average` daily averages

All of the information under these headings is sampled each hour to produce the report. The headings correspond to the four CPU counters: user, kernel, wait for I/O completion, and idle. These counters are increased by one (incremented) each time the clock calls for an interrupt (a count on the system), which occurs 60 times per second.

`-b` Report buffer activity. The `-b` option has access to three sets of read and write counters:

- `lread` and `lwrite` (logical `read`, logical `write`)
- `bread` and `bwrite` (block `read`, block `write`)
- `phread` and `phwrite` (physical `read`, physical `write`)

If you type

```
sar -b
```

a report is generated that is organized in columns as follows:

`bread/s, brwrit/s`
Samples the `bread` and `brwrite` counters giving the transfers per second of data between the system buffers and the disk.

`lread/s, lwrit/s`
Samples the `lread` and `lwrite` counters, giving the number of times the system buffers are accessed.

`phread/s, phwrit/s)`
Samples the `phread` and `phwrite` counters, giving the number of data transfers via raw devices.

`%rcache, %wcache`
Gives the ratio of buffer reads (`bread`) to logical reads (`lread`) and buffer writes (`bwrit`) to logical writes (`lwrit`) in what is called the **cache hit ratio.** The cache hit ratio reports if files are being accessed from the buffers or if the information has to be retrieved from the disk itself. A low ratio could point out an area where more efficient buffering could increase system response time.

`-d` Reports device activity. When you select this option, you can view information on block devices such as the disk or tape drives.

If you type

```
sar -d
```

a report is generated that is organized in columns as follows:

`Device`
: The device being sampled.

`%busy`
: Samples the device counters and prints out the percentage of time the device was engaged in transferring data.

`avque`
: Displays the average number of requests waiting to be processed.

`r+w/s`
: Displays the number of data transfers to or from the disk or tape drive.

`blks/s`
: Displays the number of bytes transferred and counted in block-sized units.

`avwait`
: Displays the number of milliseconds that transfer requests have to wait in the queue before being processed.

`avserv`
: Displays the average time it takes to service the request.

These columns sample a combination of I/O activity counters and character counters. The `%busy` column, for example, samples the `io_ops` counters and the `io_act` counters. The `blks/s` column uses the `io_bcnt` counters. In addition to the above combinations, the columns `avserv` and `avwait` use the `io_act` and `io_resp` counters. These counters are explained in greater detail with some suggestions for fine-tuning later in this section.

Each disk or tape device has four counters to record activity. The activity information is kept in the device status table.

Whenever an I/O request occurs, `io_ops` (I/O operations) is incremented. It keeps track of block I/O, swap I/O, and physical I/O.

Transfers between the device (particular disk or tape drive) and memory are recorded in 512-byte blocks by `io_bcnt` (I/O block counters). The `io_act` and `io_resp` are particularly useful I/O counters. By time ticks, they measure the time it takes a device to receive, process, and transmit a request, summed over all I/O requests for the device. `io_act` measures the active time, which is the time the device is actively engaged in seeking, rotating, and transferring data (all measured by different counters and combined into one active count).

`io_resp` measures the total elapsed time between the time the request is received in the queue and the time it is completed. By looking at the ratio between active time and response time, you can determine if the disk and tape devices are being put to their best use. For example, if one device is being used at such a rate as to increase the response time significantly, perhaps some of the more frequently used information could be moved to a less heavily used device, speeding up reaction time on the system. Also, if the active time counter is high for a device, compared with the number of requests on the system, perhaps you could load the file systems on the device differently for more effective access.

`-w` Reports swapping and switching activity. This option uses the `swapin` and `swapout` counters. (The column headings that appear in the report, `swpin` and `swpot`, are abbreviations.) These counters are incremented each time the system receives a request initiating a transfer to or from the swap device. The **swap device** is a disk partition used as a "holding area" for processes that are not currently running. When main memory is full, processes that are not currently running are swapped out (transferred) to the swap device. When the CPU is ready to work on a process that is in the swap device, the process is swapped back into memory.

Thus, when main memory is crowded the typical sequence of activities is

1. Load the program into memory.
2. Work on the process.
3. Swap the process out.
4. Swap the process back into memory.
5. Repeat steps 2 through 4 until the process is completed and terminates.

The amount of data swapped in and out is measured in blocks and counted by the `bswapin` and `bswapout` counters. The data collected by these counters is displayed in the column headings `bswin` and `bswot`.

The figures for `swpin` are usually higher than `swpot`. This peculiar asymmetry arises from programs with the "sticky bit" set. Once loaded, these programs reside on the swap device. Thus, the typical sequence of activities for sticky bit programs is

1. Swap the program in from the swap device.
2. Work on the process.
3. Swap out the process.
4. Swap the process back into memory.
5. Repeat steps 2 through 4 until the process is completed and terminates.

The fact that these programs are initially swapped in gives rise to the `swpin/swpot` asymmetry on the report. Thus, `swpot` is the more accurate measure of actual swapping.

If you type

```
sar -w
```

a report is generated that is organized in columns as follows:

`swpin/s`
: Reports the number of transfers from the swap area on disk to main memory.

`swpot/s`
: Reports the number of transfers from memory to the swap area.

`bswin/s, bswot/s`
Reports the number of bytes transferred.

`pswch/s`
Reports the number of process switches that have occurred.

`-y` Reports TTY device activity. There are six counters monitoring terminal device activities. Three of these monitor hardware interrupts:

`rcvint` receiver interrupts
`xmint` transmitter interrupts
`mdmint` modem interrupts

The other three monitor terminal queue activity:

`rawch` Counts the number of characters in the raw (unprocessed) queue.
`canch` Counts the number of characters in the canonical (processed) queue.
`outch` Counts the number of characters in the output queue.

To check the rate at which TTY devices are processing characters and to generate a TTY activity report, type

```
sar -y
```

In the output generated by this, the columns and their output are defined as follows:

`rawch/s, canch/s`
Displays the input character rate in raw (unprocessed) characters (`rawch/s`) and in canonical (partially processed) characters (`canch/s`).

`outch/s`
Reports the output characters.

`rcvin/s, xmtin/s, mdmin/s`
Reports the number of receiver (`rcvin/s`), transmitter (`xmtin/s`), and modem (`mdmin/s`) interrupts that occurred on the system.

`-c` Reports system calls. This option accesses the `pswitch` and `syscall` counters. These counters are related to the

management of multiprogramming. This occurs when one process, the **parent process,** calls (forks and execs) another program, the **child process.** While the parent waits, the child performs its task, terminates, and wakes up the parent process, which continues.

`syscall` is incremented every time a system call occurs. Certain system calls — the `read`, `write`, `fork`, and `exec` calls — are counted individually in `sysread`, `syswrite`, `sysfork`, and `sysexec`.

`pswitch` counts the number of times the switcher was invoked. The switcher is invoked when the program running could not complete its intended process.

The following situations cause `pswitch` to be incremented:

- a system call that resulted in a roadblock
- an interrupt that caused the awakening of a higher priority process
- a 1-second clock interrupt

To check the number of system calls, type

```
sar -c
```

The first five columns of the output generated by this give information on the number of system calls made. The first column (`scall/s`) gives the total number of system calls; the next four columns give the number of specific system calls for read (`sread/s`), write (`swrite/s`), fork (`fork/s`), and execute (`exec/s`) system calls, respectively. The last two columns give the number of characters transferred by the read (`rchar/s`) and write (`wchar/s`) system calls, respectively.

`-a` Reports on use of file access system routines. The `-a` option of `sar` checks the following:

- the number of times a file's i-node number is requested
- the number of file path searches
- the number of directory blocks read by the system

Typing

```
sar -a
```

produces a report that samples the file access counters and gives the number of times each of the file access routines is performed.

The report output is described below.

`iget/s`
: Measures the number of requests for the i-node number that corresponds to a particular file. `iget` is the routine used to locate the i-node entry (i-number) of a file. See Chapter 6, "Checking the A/UX File System: `fsck`" for an explanation of i-numbers and i-nodes. It first searches the in-core (main memory) i-node table. If the i-node entry is not in the table, `iget` will get the i-node from the file system where the file resides and make an entry in the i-node table.

`namei/s`
: Measures the number of requests for a file system path search. `namei` is the routine that performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special path. `namei` (as well as other file access routines) calls `iget` to find the inumber of the file it is searching for. Therefore, counter `iget` is always greater than counter `namei`.

`dirblk/s`
: Measures and records the number of directory block read requests issued by the system. Dividing the directory blocks read by the number of `namei` calls results in an estimate of the average path length of files. A long path length may indicate a significant number of subdirectories. Rearranging the file structure to move the more commonly accessed files higher up the path would correct this problem.

Each time one of these routines is called, the respective counter is incremented.

`-q` Reports on queue activity. At every 1-second interval, the clock routine examines the process table to see whether any processes are queued and ready. If so, the counter `runocc` is incremented and the number of processes waiting is added to the `runque` counter. While this is happening, the clock routine also checks the process status of the swapper. If the swap queue is occupied, the counter `swapocc` (swap occupied) is incremented and the number of processes waiting in the queue is added to the `swapque` counter.

Typing

```
sar -q
```

produces the average time a process is being queued before being acted upon. The report output lists the average queue length while the queue is occupied (the columns ending in `-sz`) and the percentage of time the queue is occupied (the columns beginning with `%`). It is broken down into two main parts: the run queue (the `runq` columns), which lists the processes in memory and runnable; and the swap queue (the `swp` columns), which lists the processes swapped out but ready to run.

`-v` Reports the status of text, process, i-node, and file tables. When an overflow occurs in any of the i-node, file, text, or process tables, the corresponding counter (`inodeovf`, `fileovf`, `textovf`, or `procovf`) is incremented. These indicate resource problems with tables or the size of memory.

You can use the `-v` option of the `sar` command to report the size of tables and any tables indicating an overflow.

Typing

```
sar -v
```

produces a report in which the first four columns report the number of entries over the size of the entries for the text, process, i-node, and file tables, respectively. The measurement is taken once at the sampling point. The last four columns give the number of overflows that occur between the sampling points.

`-m` Reports message and semaphore activities. This option of the `sar` command reports which processes requested the operating system to send information directly to another process.

Typing

```
sar -m
```

generates a report on message and semaphore activity in which the message and semaphore columns (`msg/s` and `sema/s`) count the most basic input/output operations of the system. These "primitives" (for example, `read` and `write`) are called upon by other programs, which use them as building blocks to complete their processes. The message primitives keep track of interprocess communications; that is, the number of times one process asks the operating system to send information directly to another process. The semaphore primitives synchronize the actions of various processes and facilitate the use of shared resources.

5.3 The `sag` command

`sag` is the system activity graph command. It displays the system activity data that was created by a previous run of the `sar` command and stored in binary format.

You can plot the graph using any single column or combination of columns. It can prepare cross-plots or time plots.

A graphics package that can invoke the `graphics` and `tplot` commands must reside on the system for you to print a system activity graph. See `sag`(1G) in *A/UX Command Reference.*

5.4 The `timex` command

The `timex` command is an extension of the `time` command; see `time`(1) and `timex`(1) in *A/UX Command Reference.* It times a command and reports process data and system activity. The given command is executed and the elapsed time, user time, and system time spent in execution of the command are reported in seconds.

Normally, you use the `timex` command to measure a single command. If you want to measure multiple commands, combine the commands in an executable file and time the file. You can also do this by typing

```
timex sh -c "cmd1; cmd2;...;"
```

This allows `timex` to extract the user and system times consumed by all the commands as if they were one single command. See `sh`(1) in *A/UX Command Reference.*

Because process records associated with a command are selected from the accounting file `/usr/adm/pacct`, background processes that have the same user ID, terminal ID, and execution time window are included in the totals given.

You can specify options to list or summarize process accounting data for the command and its children and to report the total system activity during the execution interval. If you don't specify any options, `timex` behaves the same as the `time` command.

The syntax for the `timex` command is

```
timex [-pos] command
```

The `timex` command options are as follows:

`-p` Lists the process accounting records for the specified command. This option has six suboptions:

- `f` Prints the `fork/exec` flag and system exit status.
- `h` Reports the fraction of total available CPU time the process consumes during its execution and suppresses reporting of the mean memory size.
- `k` Reports the total kcore-minutes and suppresses reporting of memory size.
- `m` Reports the mean core size.
- `r` Reports the fractional representation of CPU factor (user time) over system time plus user time.
- `t` Reports separate system and user CPU times.

`-o` Reports the total number of blocks read or written and total characters transferred by the command selected and all its children.

`-s` Reports the total system activity during the execution interval of the command, not just the activity resulting from the command specified. All the data items listed in `sar` are reported.

Chapter 9
The UUCP System

Contents

Tables

Chapter 9

The UUCP System

1. Introduction

One of the most important features of A/UX is the ability to transfer information between A/UX systems (and other systems derived from UNIX). The standard A/UX communication package, called UUCP (for "UNIX to UNIX copy"), is the most common mechanism for performing this function. The UUCP package permits mail, command execution, and file transfers between A/UX computers. These functions are useful whenever related pieces of information and/or users are located on separate computers. For example, mail can be used to communicate with other companies, file transfer can be used to directly copy programs from one computer to another, and remote command execution can be used to print documents on a remote computer that has an attached laser printer.

When two or more computers can communicate, the computers and the communication channels between the computers (for instance, phone lines) are considered to be parts of a computer network. Each computer is a node in the network; the communication channels are network links. Communication between nodes can take place using one of two primary methods: interactive or automated. **Interactive communication** requires a user to log into the remote system and issue commands to initiate command execution and file transfer. **Automated communication** requires some method by which the local computer can communicate with the remote computer(s) without user intervention.

This chapter describes the many commands, scripts, and spooling directories involved in UUCP, and then introduces a step-by-step procedure for setting up UUCP to handle interactive logins and file transfer between UNIX nodes. Next, the procedure for setting up mail to a remote UNIX node is described. The chapter concludes with hints

for modifying, expanding, and tightening the security of UUCP on your system.

2. The components of UUCP

This section discusses the user files and directories that are used when running UUCP and the system files and directories that are used in the administration of UUCP.

The files can be grouped into the following categories:

- executable: binary executable files used by UUCP
- script: shell scripts used as commands and for system maintenance
- A/UX system: A/UX system files also used by UUCP
- `uucp` system: files used by UUCP for system configuration
- statistical: files used by UUCP for storage of statistical information
- sequence: files used for storing sequential number information
- log: files used for logging information on requests and connections
- audit: files used for storing debugging information
- spool: files used to spool requests
- lock: files used to lock UUCP system files and prevent simultaneous updating or accessing
- status: files used to store status of connections to specific systems
- temporary: files used for temporary storage of information
- backup: files used for keeping backups of recent log files

After discussing the directories involved, this section discusses each of these categories and the files that constitute them.

2.1 Directories

The following directories are used exclusively by `uucp`:

`/usr/lib/uucp`
: Used for storage of UUCP system files that are not temporary in nature. This includes script files run by `cron` and executable binary files that are forked by UUCP processes.

`/usr/spool/uucp`
: Used for storage of UUCP files that are temporary in nature. This includes log files and spool files.

`/usr/spool/uucp/.XQTDIR`
: Used by `uuxqt` for temporary storage of files used in remote command execution by `uux`.

`/usr/spool/uucppublic`
: The directory reserved for public usage when sending files from one system to another. The directory has read, write, and execute permission for every user of the system (that is why it is said to be for public usage) and all files that are to be sent to the computer should be sent to this directory. Some subdirectories are used for specific purposes.

`/usr/spool/uucppublic/`*user*
: Where *user* is a specific user name, used by the system to place files that could not otherwise be transferred because of permission problems.

`/usr/spool/uucppublic/receive`
: Along with all of its subdirectories, used exclusively by the programs `uuto` and `uupick`.

`/usr/spool/uucppublic/receive/`*user*
: Where *user* is a known user on the system, used to build directories for files from other systems sent by `uuto`.

`/usr/spool/uucppublic/receive/`*user*`/`*system*
: Where *user* is a known user on the computer and *system* is a remote computer, used to store files sent from *system* to *user* with `uuto`. This is where `uupick` will find files.

2.2 Executable files

The following files are the binary executable files used by UUCP for common usage and administration:

`/usr/bin/uucp`
: The user command that copies files from system to system. Forwarding may be allowed through intermediate nodes. See `uucp`(1C) in *A/UX Command Reference.*

`/usr/bin/uulog`
: May be used as a user command or a system maintenance command. As a user command, it presents logged information about `uucp` or `uux` requests by either system name or user name. As a maintenance command, it appends any temporary log files to the permanent log file. See `uucp`(1C) in *A/UX Command Reference*.

`/usr/bin/uuname`
: A user command whose output is either the local node name or a list of the node names of all computers with which the system can communicate. See `uucp`(1C) in *A/UX Command Reference.*

`/usr/bin/uustat`
: May be used to display the status of `uucp` jobs or connections and can be used to cancel certain jobs. See `uustat`(1C) in *A/UX Command Reference*.

`/usr/bin/uusub`
: May be used as a user command or a system maintenance command. As a user command, it displays the statistics on `uucp` connections or traffic. As a maintenance command, it flushes the statistics files, adds new systems for statistics gathering, or calls other systems. See `uusub`(1M) in *A/UX System Administrator's Reference.*

`/usr/bin/uux`
: A user command that executes commands on a remote system and takes care of transferring all files necessary for the command. See `uux`(1C) in *A/UX Command Reference.*

`/usr/lib/uucp/uucico`
: Forked by `uucp` or `uux` to call other systems and do all the work. It may be called directly from shell scripts started by `cron` to scan for work needing to be done. It is also executed directly when a remote system logs in.

`/usr/lib/uucp/uuclean`
: A system maintenance command usually called from shell scripts started by `cron` to remove old files from `uucp` directories. See `uuclean`(1M) in *A/UX System Administrator's Reference*.

`/usr/lib/uucp/uuxqt`
: Forked by `uux` or `uucico` to execute the scripts that `uux` sets up to perform remote command execution.

2.3 Script files

The following are script files used as normal commands and files used for system maintenance:

`/usr/bin/uupick`
: A user shell script used to pick up files that have been sent to a user from another system via `uuto`. The appropriate public subdirectories are searched and the user may accept or reject files for copying to another directory. See `uuto`(1C) in *A/UX Command Reference*.

`/usr/bin/uuto`
: A user shell script that sends files or entire directories via `uucp` to a user on another system. In the case of directories, entire subtrees are sent by calling `uucp` repeatedly. See `uuto`(1C) in *A/UX Command Reference*.

`/usr/lib/uucp/uushell`
: A login shell script used to set the time zone environment variable before executing `uucico`. This will cause log file entries to show the correct time for remotely initiated requests and connections. It should be the login shell for all `uucp` connections.

`/usr/lib/uucp/uudemon.day`
: A maintenance shell script started by `cron` every night to

perform UUCP maintenance. It is normally used to remove old files from UUCP directories and trim status and statistics files, that is, to eliminate old entries to prevent the files from growing too much.

`/usr/lib/uucp/uudemon.hr`
A maintenance shell script started by `cron` every hour to perform UUCP maintenance. It is normally used to append temporary log files to permanent log files and to check for spooled work.

`/usr/lib/uucp/uudemon.wk`
A maintenance shell script started by `cron` every week to perform `uucp` maintenance. It is normally used to store week-old log files and remove two-week-old log files.

The descriptions of the `uudemon` scripts are intended to demonstrate their typical uses. You decide when each of these scripts is actually run via its respective `crontab` entry. You can also easily change what these scripts actually do by editing the script files.

2.4 A/UX system files

The following system files are used by UUCP:

`/etc/group`
Keeps track of group IDs for user names. `uucp` uses these user names to allow remote systems to log in. See Chapter 3, "User Administration."

`/etc/inittab`
Generates `gettys` for ports. These `gettys` must be enabled for dialin ports and disabled for dialout ports. See "Dialin and Dialout Ports" later in this chapter.

`/etc/passwd`
Keeps track of user names and passwords. `uucp` uses these to allow remote systems to log in and transfer files. Both the home directory and the login shell are specified for each user name. See Chapter 3, "User Administration."

`/usr/spool/cron/crontabs/uucp`
Tells the `cron` process when to schedule the execution of

programs. `uucp` uses this to schedule hourly, daily, and weekly shell scripts to perform maintenance. See Chapter 7, "System Accounting Package."

2.5 UUCP system files

UUCP uses the following system files for configuration:

`/usr/lib/uucp/ADMIN`
: Stores additional information about each system that appears in the `L.sys` file. You can display this description along with the system names using the command `uuname -v`.

`/usr/lib/uucp/FWDFILE`
: Stores a list of system names to which files may be forwarded. These are a subset of the `L.sys` system names and do not restrict the final destination, just the next system in the path. Each line in this file looks like

  system[, *user*, *user*]

  where *system* is the name of a system to which a communication can be forwarded, and the optional list of *user*s separated by commas represents the login names of users in the system to which the communication can be forwarded.

`/usr/lib/uucp/L-devices`
: Stores the names of the ports that are connected to modems or directly to other systems. The file contains the attributes, such as baud rate, for each of these ports. Each line in this file looks like

  type line device speed [*protocol*]

  where *type* can be either `DIR`, indicating that the line is directly connected to another system (this includes modem connections), or `ACU`, indicating that the line uses an automatic calling unit; *line* is the device name for the line (for instance, `tty0` if the modem is connected to port `/dev/tty0`); *device* is the device name of the `ACU` if one is specified in the *type* field (otherwise, a placeholder `[0]` must be used in this field); *speed* is the line speed for the connection, as measured in baud; and *protocol* is an optional field that needs to be filled only if the connection is for a protocol other than the default protocol.

A typical entry in the `L-devices` file looks like

```
DIR tty0 0 1200
```

`/usr/lib/uucp/L-dialcodes`

Stores dial-code abbreviations for `L.sys`. Each abbreviation is associated with a dial sequence of numbers (typically an area code). Each line in this file looks like

abbreviation dialing-sequence

where *abbreviation* stands for an arbitrary abbreviation of an area code or telephone exchange number, and *dialing-sequence* stands for the telephone number that should be dialed after the abbreviation.

A typical entry in this file looks like

```
sfba 415
```

where `sfba` stands for San Francisco Bay Area and `415` is the area code.

`/usr/lib/uucp/L.cmds`

Stores a list of command names that `uux` is permitted to execute. If a command name is not listed in this file, it cannot be used for remote execution. This includes `rmail`. Each line in this file looks like

cmd

where *cmd* is the name of any command you want `uux` to be able to execute in your system.

`/usr/lib/uucp/L.sys`

Stores information on connecting to remote systems. Each line represents a way to connect to another system. If more than one line exists for a particular system, each line is tried sequentially until a connection is made. Each line in this file looks like

system time device class phone login

where *system* is the name of the remote system; *time* is the time(s) at which that system can be called; *device* is either the

name of the port (if it is a DIR connection) or the keyword ACU; *class* is the speed of the connection in baud; *phone* is the phone number to be called, expressed either in an all-numeric sequence or in an alphabetic abbreviation, as specified in a corresponding line in the L-dialcodes file; and *login* is an *expect-send-expect* sequence as specified in "Automatic File Transfer."

A typical entry in this file looks like

```
doosy Any tty0 1200 tty0 "" ATDTsfba5551212^M
ogin:-@-ogin:-EOT-ogin:-BREAK-ogin:u_mickey ssword:mouse
```

/usr/lib/uucp/ORIGFILE

Stores a list of system names and users from which files may be forwarded. Each entry refers to the system that was the originator of the request and not the most recent system in the path. Each line in this file looks like

system[,*user*,*user*]

where *system* is the name of a system whose communication the system is willing to forward, and the optional list of *users* separated by commas represents the login names of users in that system whose communication can be forwarded. Note that *system* represents the name of the system that originally was the source of the communication, not the name of the last system that forwarded the communication.

/usr/lib/uucp/USERFILE

Stores access permissions. These include pathname prefixes that are accessible by local users and remote systems. Remote system login names must appear here with an optional call-back facility. Each line in this file looks like

[*login*],*system* [c] *path* [*path*]

where *login* is the login name for a user or a remote computer; *system* is the system name for a remote computer; c is an optional flag that, as a security measure, requires that the remote computer be called back before any further communication takes place; and *path* is a pathname prefix constraining file access to only those files preceded by the prefix. If the *login* field is empty

(a null login), any user can access the specified path.

As an example, the lines

```
root, /
, /usr/spool/uucppublic
```

permit any user at any remote computer to transfer any files from `/usr/spool/uucppublic`, but only a user with the login name of `root` can transfer any file, because all filenames ultimately begin with `/`.

2.6 Statistical files

UUCP uses the following files to store status and statistical information:

`/usr/lib/uucp/L_stat`
: Stores the latest connection status of each remote system. You can display this information using the `uustat` command.

`/usr/lib/uucp/L_sub`
: Stores connection statistics for each remote system. You can display this information using the `uusub` command.

`/usr/lib/uucp/R_stat`
: Stores the status of each `uucp` request. You can display this information using the `uustat` command.

`/usr/lib/uucp/R_sub`
: Stores traffic statistics for each remote system. You can display this information using the `uusub` command.

2.7 Sequence files

UUCP uses the following files for storing sequence information:

`/usr/lib/uucp/SEQF`
: Stores the sequence number, which is incremented by one for each `uucp` request. This is a four-digit number used in generating the names for the spooled files.

`/usr/lib/uucp/SQFILE`
: Stores a conversation count for remote systems. These systems will be a subset of the systems in `L.sys`. The remote system

must also have an entry for your system in its SQFILE. If a connection is made but the counts in the two files do not agree, the login attempt fails.

/usr/lib/uucp/SQTMP
Temporarily stores the SQFILE used by uucico.

2.8 Log files

UUCP uses the following files for logging information on UUCP requests and connections:

/usr/spool/uucp/ERRLOG
Logs uucp errors generated when uucico fails on a file transfer. If the -x option is used with uucico, the errors do not appear in this file.

/usr/spool/uucp/LOGDEL
Used by uuclean to log files that have been removed.

/usr/spool/uucp/LOGFILE
Logs the status of calls and uucp requests. During execution of uucp requests, log information is normally appended to the file. If more than one uucp process is active at a time, the information is logged to temporary files. You can use the uulog command to display portions of this file as well as append temporary versions to it.

/usr/spool/uucp/SYSLOG
Logs the number of bytes sent and received during each connection and the duration of the connection in seconds.

2.9 Audit files

UUCP uses the following files to store debugging information:

/usr/spool/uucp/AUDIT
Stores debugging information from the remote uucico process. It is created every time uucico is run as a login shell.

/usr/spool/uucp/AUDIT.*system*
Where *system* is the name of a remote computer, stores debugging information when the remote computer calls with uucico and the -x option.

2.10 Spool files

UUCP uses the following files to spool requests for work to be done:

`/usr/spool/uucp/C.`*systemxxdddd*
: Stores information for `uucico` on which system to connect and the source and destination filename to transfer. It is created whenever `uucp` or `uux` is executed.

`/usr/spool/uucp/D.`*systemxxdddd*
: Stores data files for transfer. This file is created when you use the `-c` option with `uucp`.

`/usr/spool/uucp/X.`*systemxxdddd*
: Stores information used to execute remote commands. This file is created prior to running `uuxqt`, which reads it.

In these filenames *system* is the name of the remote computer, *xx* are two ASCII characters representing the priority of the work, and *dddd* is the sequence number from `SEQF`.

2.11 Lock files

UUCP uses the following files as lock files to prevent simultaneous update of UUCP system files:

`/usr/spool/uucp/LCK..`*system*
: Where *system* is the name of a remote computer, prevents more than one simultaneous connection to that computer. Notice the two dots (`..`) in the name. These are an integral part of the filename.

`/usr/spool/uucp/LCK..`*ttynn*
: Where *ttynn* is the name of a port used for dialout or direct connection, prevents more than one `uucico` process from using the port at the same time. Notice the two dots (`..`) in the name. These are an integral part of the filename.

`/usr/spool/uucp/LCK.LOG`
: Prevents simultaneous update of the log files.

`/usr/spool/uucp/LCK.LSTAT`
: Prevents simultaneous update of `L_stat`.

`/usr/spool/uucp/LCK.LSUB`
Prevents simultaneous update of `L_sub`.

`/usr/spool/uucp/LCK.RSTAT`
Prevents simultaneous update of `R_stat`.

`/usr/spool/uucp/LCK.RSUB`
Prevents simultaneous update of `R_sub`.

`/usr/spool/uucp/LCK.SQ`
Prevents simultaneous update of `SQFILE`.

`/usr/spool/uucp/LCK.SEQL`
Prevents simultaneous update of `SEQF`.

`/usr/spool/uucp/LCK.XQT`
Prevents simultaneous remote command execution by `uuxqt` (see "Executable Files").

2.12 Status files

UUCP uses the following file to store the status of connections to specific systems:

`/usr/spool/uucp/STST.`*system*
Where *system* is the name of a remote computer, used to store the status of a connection that fails.

2.13 Temporary files

UUCP uses the following files for temporary storage:

`/usr/spool/uucp/LTMP.`*pid*
Used to temporarily store `LOGFILE` entries when more than one `uucp` process is executing. *pid* is the process ID. The `uulog` command is used to append these files to `LOGFILE`.

`/usr/spool/uucp/TM.`*pid*`.`*ddd*
Temporarily stores these data files until the transfer is complete. *pid* is the process ID and *ddd* is the number of this file in the current transfer. If the transfer is successful, the file is moved to the correct destination; otherwise it is removed.

2.14 Backup files

UUCP maintenance scripts create the following files to keep backups of recent log files:

`/usr/spool/uucp/Log-WEEK`
: Stores the `LOGFILE` entries for the current week to date. The shell script `uudemon.day` appends the current `LOGFILE` to `Log-WEEK` every night.

`/usr/spool/uucp/o.Log-WEEK`
: Stores the `LOGFILE` entries for the entire previous week. Between `Log-WEEK` and `o.Log-WEEK` the system will have a backup of one to two weeks of `LOGFILE` entries.

`/usr/spool/uucp/o.SYSLOG`
: Stores the `SYSLOG` entries for the entire previous week.

3. Setting up the `L-devices` and `L.sys` files

There are many files that you will become familiar with as you work with the UUCP system. Two are of particular importance because they affect both interactive and automated file transfers.

3.1 The `/usr/lib/uucp/L-devices` file

The `L-devices` file contains information about what devices the computer can use to communicate with the outside world (modem, automatic calling unit, and so on). If you already have a modem on your system, the file should contain information about what port it is attached to. If you don't have a modem attached, you will have to attach one to communicate with other computers. See Chapter 5, "Managing Peripheral Devices" and "Dialin and Dialout Ports" later in this chapter for a review of how to do this. Each line in the `L-devices` file has the form

type line device speed [*protocol*]

where

type can be either `DIR`, indicating that the line is directly connected to another system (this includes modem connections), or `ACU`, indicating that the line uses an automatic calling unit;

line is the device name for the line (for instance `tty0` if the modem is connected to port `/dev/tty0`);

device is the device name of the ACU if one is specified in the *type* field (otherwise, a placeholder `[0]` must be used in this field);

speed is the line speed for the connection, as measured in baud; and

protocol is an optional field that needs to be filled only if the connection is for a protocol other than the default protocol.

Take a look at this entry from an `L-devices` file:

```
DIR tty0 0 1200
```

`tty0` is the name of the port to which the modem is attached. If your modem is attached to another port, you should substitute its name for `tty0`.

`1200` is the speed, in baud, at which the modem will communicate. If you have a 300/1200 selectable modem, you should have two lines in your `L-devices` file that look like

```
DIR tty0 0 1200
DIR tty0 0 300
```

3.2 The `/usr/lib/uucp/L.sys` file

The `L.sys` file contains information that your system uses to call other computers.

> *Note:* The `L.sys` file contains information vital to the security of your neighboring UUCP sites. Keep its permissions closed to reading by unauthorized parties.

The following is an example `L.sys` file entry:

```
doosy Any tty0 200 tty01 "" ATDT5551212^M\
ogin:-@-ogin:-EOT-ogin:-BREAK-ogin:u_mickey ssword:mouse
```

`doosy` is the name of the system this entry allows you to call. Every system has a name that identifies it.

`Any` indicates that your system can call `doosy` at any time. You will find out how to restrict these times later. Once again `tty0` is the port to which the modem is attached and `1200` is the speed at which the modem will communicate. The second `tty0` is a placeholder for a telephone number.

The rest of the line has the form

expect-send-expect-send

where *expect* is what your computer expects the other computer to transmit to it, and *send* is what your computer will transmit to the remote site. The `""` in the example above is a null string that will expect nothing. `ATDT5551212^M` is sent to the modem telling it to dial 555-1212. This assumes that you are using a Hayes-compatible modem. For any other type, you would substitute whatever command would cause the modem to dial the number (this information can be found in the modem owner's manual).

Note: `^M` is CONTROL-m. To enter it into a file when using the `vi` editor, press CONTROL-v. Then press CONTROL-m. This tells the computer to send a carriage return. This is not the same as typing `^m` (caret-m). It is not the same as typing UP ARROW-m either.

The second *expect* field is then broken up into

expect-send-expect-send-expect-send-expect

In other words, the simplest form of *expect-send-expect-send* for the rest of the line would be `ogin: u_mickey ssword: mouse`. This tells `uucp` to expect a prompt ending with `ogin:`. When it gets the prompt it sends `u_mickey`. Then it expects `ssword` and replies with `mouse`. The strings `ogin:` and `ssword:` are used because most computers send a string ending in either `Login:` or `login:` to each port. After you enter a login, most computers respond with either `Password` or `password`.

Unfortunately things are not so simple. This garbled-looking expect string:

```
ogin:-@-ogin:-EOT-ogin:-BREAK-ogin:
```

is of this form

expect-send-expect-send-expect-send-expect

What it means is expect "`ogin:`"; if that does not happen, wait (that's what the `@` means) and expect "`ogin:`". If that does not happen, send `EOT`, and so on.

Why the login `u_mickey`? The `u_` is an ad hoc convention for `uucp` logins, but it doesn't need to be adhered to. This login name can be any name that you and the remote system's administrator agree on.

While A/UX supports dialing on assorted modems via the `/etc/remote` and `/etc/phones` files, (see `remote`(4) and `phones`(4) in *A/UX Programmer's Reference*), you may wish to dial an unsupported modem.

The following `L.sys` entry shows how this may be accomplished, using the Apple Personal Modem (APM) as an example:

```
foo Any tty0 1200 tty0 "" atdt1234567\r 1200 \r
     ogin:-BREAK-ogin: Umine ssword: passwd4foo
```

Note: The above lines have been wrapped onto two lines. This must appear on a single line in the A/UX `L.sys` file.

This says that machine `foo` may be called using tty0 at 1200 bps, at any time. `uucp` is told to expect and send the following strings:

Table 9-1. Expect-send strings for the Apple Personal Modem

expect	*send*	Comments
""		Don't wait for anything.
	atdt1234567\r	Send APM command line.
1200		Wait for "CONNECT 1200".
	\r	Send a Carriage Return.
ogin:-BREAK-ogin:		Wait for "... login".
		Send a break, if necessary.
	Ubar	Foo knows us as "Ubar".
ssword:		Wait for "password:".
	passwd4foo	Send the password...

4. Interactive file transfer

At this point you can already start transferring files to and from another system, provided that the other system is set up so that you can log into it.

To dial out, you must make sure that the modem port does not have a `getty` running on it, that is, that it is not working as an incoming modem. For that, you must establish your modem either as an outgoing modem only or as both a dialout and a dialin modem. For an explanation of how to do this, refer to Chapter 5, "Managing Peripheral Devices." To call up the other computer, you can use the `cu` command (see Chapter 5, "Using `cu`," in *A/UX Communications User's Guide*). You can, for instance, enter

```
cu -l line dir
```

where *line* is the name of the port to which the modem is attached and `dir` ensures that `cu` uses that line. In this case, `cu` will look in the `L-devices` file for the appropriate speed at which to communicate, and once it finds it, it will inform you by printing the message

```
Connected
```

on the screen. This means that you are connected to the modem, and you can now instruct the modem to dial out, for instance,

```
ATDT5551212
```

if your modem is Hayes-compatible. If the computer at the other end of the line is available and a connection with it can be established, the familiar login prompt will appear on your screen, and you will be able to log into the other computer.

You can also call up the other system, specifying the speed you want the connection to be at. For this, enter

```
cu -lline -sspeed
```

Once again, `cu` will look in the `L-devices` file, this time to check that the requested speed for the requested line is available. If it is, the

```
Connected
```

message appears on your screen as before, and you can log in on the other computer.

Finally, you can also use `cu` without specifying either *line* or *speed*, but using *system-name* as found in the `L.sys` file. For instance,

```
cu doosy
```

calls the system `doosy` and goes through the login procedure as specified in the `L.sys` file.

Once you are logged in, you can transfer files, one by one, from your local machine to your remote machine by entering

```
~%put infile [outfile]
```

where *infile* is the name of the file you are transferring and *outfile* is the name you want it to have in the remote computer. If you do not specify *outfile*, the file transferred will have the same name as in the local computer.

You can also transfer files from the remote computer to your local one by entering

```
~%take infile [outfile]
```

This works exactly as `put`, but in the reverse direction.

Although `cu` is the simplest method for file transfer between two computers, it has the disadvantage that it does not provide any kind of

error checking.

5. Automatic file transfer

The UUCP system also provides for an automatic file transfer capability between two computers. Other files, apart from the `L-devices` and `L.sys` files, have to be adjusted to permit this to operate properly. Not only must the remote computer be set up to recognize your computer, its login name, and its password, but in order to receive files from the remote computer, the local one has to be able to recognize it too.

5.1 Preparing the systems

5.1.1 The system node name

You must decide on a node name for your system. This is the name other people will put in their `L.sys` file to allow them to call your computer. If you want to change the node name from the current host name for your system, use a text editor to open the `/etc/HOSTNAME` file and change the first field in that file to the new name you want for your computer. When you reboot your system, the new node name will be in effect.

5.1.2 Dialin and dialout ports

The `/etc/inittab` file needs to be modified to enable `gettys` for dialin ports and disable them for dialout ports. Depending on your exact needs, you might need a range of differing `/etc/inittab` lines. Here is a sampling which should cover most normal cases:

- Dialing out only, at 9600 bps.

```
do:2:off:/etc/getty tty0 at_9600     # Port tty0
```

- Incoming calls only, at 1200 bps.

```
du:2:respawn:/etc/getty tty0 tt_1200  # Port tty0
```

- Both directions, using an Apple Personal Modem.

```
do:2:wait:/etc/setup_apm tty0          # Set up port
du:2:respawn:/etc/getty tty0 mo_1200  # Port tty0
```

This last case deals with the fact that the Apple Personal Modem (APM) powers up with answering disabled. The following script

programs the APM to answer calls:

```
:
: setup_apm - Set up Apple Personal Modem
:
# Usage: setup_apm tty[01]

rm -f /usr/spool/uucp/LCK..$1  # remove any lock file

stty -n /dev/$1 modem            # in case this is tty1

for i in z 's0=1'; do            # send commands to APM
   echo "at$i\r\n~.\n" |
   cu -s1200 -l$1 > /dev/null 2>&1
done
```

Note: The `stty` line deals with the fact that TTY1 is not normally set up for use as a modem line. This will be needed when using `mo_####` codes, even if an APM is not involved. Without this `stty`, `getty` cycling has been observed to occur. For more information on `/etc/gettydefs` see Chapter 5, "Managing Peripheral Devices."

5.1.3 Generic `uucp` logins

Your system comes configured with two generic `uucp` logins:

```
uucp::5:5:UUCP admin:/usr/spool/uucppublic:
        /usr/lib/uucp/uushell
nuucp::5:5:UUCP admin:/usr/lib/uucp:
```

Note that the first sample line above has been wrapped onto two lines with the second line indented; this must appear as one long line in the `/etc/passwd` file.

Both of these `uucp` logins should be assigned passwords. Note that both generic logins have the same UID and GID. See "The `nuucp` Login Environment" later in this chapter for more information.

The nuucp login is for administrative use; when you are working on uucp you should log in as nuucp. Your home directory will be /usr/lib/uucp and you will have the same permissions as the uucp login. See "The nuucp Login Environment" later in this chapter for information on using this login.

The other generic uucp login is uucp. The startup program for this login is /usr/lib/uucp/uushell, a script that establishes the time zone (TZ) variable and calls the uucico program. uushell should be the startup program for any uucp login except the administrative login nuucp.

5.1.4 System-specific uucp logins

It is not a good idea to allow other systems to log in as nuucp. Instead, you can setup a unique entry in /etc/passwd and USERFILE for each remote computer that will be calling your system. This allows you to control the access from each of these computers independently. For instance, if the password for one of these is inadvertently disclosed, you can change the password for that system's login and not have to inform the administrators of every computer with which your system connects. Likewise, if one computer starts behaving strangely and calls you at the wrong times or ties up the phone line, you can temporarily change the password to stop the problem until you contact the administrator of the problem system.

The conventional name for a system-specific uucp account is U*xxx*, where *xxx* is the calling machine. For example, a typical set of entries in /etc/passwd might be:

```
uucp:SkQs1q/3e1NMo:5:5:UUCP:
        /usr/spool/uucppublic:/usr/lib/uucp/uushell
nuucp:GpBTy2Ls/upXk:5:5:UUCP admin:/usr/lib/uucp:
Ufoo:avxoVAFxOzBTU:uid:5:UUCP for Foo:
        /usr/spool/uucppublic:/usr/lib/uucp/uushell
Ubar:4vpPMIds1ZpHk:uid:5:UUCP for Bar:
        /usr/spool/uucppublic:/usr/lib/uucp/uushell
```

Note: Three of the sample lines above have been wrapped onto two lines with the second line indented. Each of these lines

must appear on one long line in /etc/passwd.

Note that each system-specific uucp account should have a unique UID and the same GID as the generic uucp login.

If you have separate entries in /etc/passwd, you must have separate entries in USERFILE. When a remote system logs in, USERFILE is searched for the user name from /etc/passwd that was used to log in. The system name in the same entry must either match the system name of the remote computer or be null. See "Controlling Logins and File System Access" later in this chapter.

5.2 Receiving mail and files

You now want to make sure that users on the doosy system can send mail or files to users on your system. It is assumed that you have at least one more modem port that is a dialin port, or that your one modem can serve as both dialout and dialin modem. All you have to do is add an entry in the /etc/passwd file on your own system for the system doosy. The entry should look like this:

```
Udoosy::uid:5:UUCP for Doosy:/usr/spool/uucppublic:
        /usr/lib/uucp/uushell
```

Note: This sample line has been wrapped onto two lines with the second line indented. It must appear on one long line in /etc/passwd.

This means that when a machine logs on as Udoosy, it will not get a normal shell but start up the uucp process directly with the program uushell, which in turn calls /usr/lib/uucp/uucico (UNIX to UNIX copy in copy out). Make sure that the user ID (*uid* in the example) is unique in the system and that the group ID is the same as the number in /etc/group in the entry for uucp (5 in the standard A/UX distribution). For more information on UIDs and GIDs, see Chapter 3, "User Administration."

Next you must assign a password for the user Udoosy. While logged in as root, type

```
passwd Udoosy
```

(You will be asked to enter the password twice, as usual for password changes.)

To give remote users access to parts of your file system, you must modify `/usr/lib/uucp/USERFILE`. A conservative security measure is to force all files to be copied in and out of `/usr/spool/uucppublic`. Note that the permissions on `/usr/spool/uucppublic` must be left open to all users (`777`), for example

```
drwxrwxrwx  2 uucp  uucp  944 Sep 24 08:49 uucppublic
```

Adding the following line to this file will allow all users on the system `doosy` to send you files on `/usr/spool/uucppublic` and use the mail facility:

```
Udoosy,doosy  /usr/spool/uucppublic
```

You must follow add a specific entry to `/usr/lib/uucp/USERFILE` for each system you may want to have `uucp` access to `/usr/spool/uucppublic` on your system. To be able to send mail and files to the remote computer, the same procedure must be followed on that system to allow your system access permission. See "Controlling Logins and File System Access" later in this chapter for allowing access to specific users or to other directories on your system.

5.3 Sending mail or files

To dial out, you must make sure that the modem port does not have a `getty` running on it, that is, that it is not working as an incoming modem. For that, you must establish your modem either as an outgoing modem only or as both a dialout and a dialin modem. For an explanation of how to do this, refer to Chapter 5, "Managing Peripheral Devices."

Once the modem is able to dial out, you should be able to send mail or to a user on the `doosy` system using the syntax

```
mail doosy!user
```

(When you are using the C shell, a backslash (\) must precede the

exclamation mark.)

When sending files using the C shell, the notation ~uucp may be used for as shorthand for /usr/spool/uucppublic. For example, to send a file named my.file to the uucppublic directory on a system named doosy, a user would enter the following commands from the C shell:

```
cp ./my.file /usr/spool/uucppublic
cd /usr/spool/uucppublic
uucp my.file doosy\!~uucp
```

See "Controlling Logins and File System Access" later in this chapter for allowing access to specific users or to other directories on your system.

5.4 Cleaning up

If you have mail going from your computer to doosy and from doosy to your computer, you must make sure that uucp cleans up after itself; that is, that log files do not grow to enormous lengths, and that if there is work spooled but for any reason not executed within a certain time (such as a few days or a week), it will be purged from the spooler. Three shell scripts are provided to do this work:

```
/usr/lib/uucp/uudemon.hr
/usr/lib/uucp/uudemon.day
/usr/lib/uucp/uudemon.wk
```

The /etc/cron utility runs these scripts on a regular basis; see cron(1M) in *A/UX System Administrator's Reference*. cron starts when the system boots; see Chapter 7, "System Accounting Package". It checks the file /usr/spool/cron/crontabs/uucp (which you must create) once every minute to see if it has changed; see crontab(1) in the *A/UX Command Reference*. Create /usr/spool/cron/crontabs/uucp as follows:

```
56 **** /bin/su uucp -c
     "/usr/lib/uucp/uudemon.hr > /dev/null"
0 4 *** /bin/su uucp -c
     "/usr/lib/uucp/uudemon.day > /dev/null"
30 5 ** 1 /bin/su uucp -c
     "/usr/lib/uucp/uudemon.wk > /dev/null"
```

Note: In this example, output lines have been wrapped onto two lines with the second line indented. These lines will each appear as one long line on the screen.

If you no longer want to run `uucp`, this file must be removed or renamed.

At this point, if all has gone well, you have a functioning remote mail system that cleans up after itself. Not only does `mail` work but `uucp` and `uux` should also work, because both these utilities use less of the `uucp` system than `mail`. To confirm that everything is working, not just `uucp` and `uux`, establish a link with an actual computer. When this is done, send mail to a user on the other system requesting that person to send you a reply. If you receive mail from the person on the remote system, everything is working properly.

If you do not receive a reply within a reasonable period of time, there are a few things you should do to find out where the problem may be. First test that the modem connections between the two sites are working correctly. For this, use `cu` to call up the other system, and ask the other person to call up your system with `cu`. Then, if the `cu` connection is working properly, check whether the other person received your mail message or not and whether a response message was actually sent back to you. Finally, you may have to go back through all the steps described above and check that everything was done properly, both at your end and at the other computer's end.

6. Security and other tips

This section looks at the security features of `uucp` as well as more administrative procedures and what can be done to debug an improperly functioning `uucp` link.

6.1 Controlling logins and file system access

If you have separate entries in `/etc/passwd` for each remote computer that will be calling your system (see "System-Specific `uucp` Logins"), you must also have separate entries in `/usr/spool/uucp/USERFILE`. When a remote system logs in, `USERFILE` is searched for the user name from `/etc/passwd` that was used to log in. The system name in the same entry must either match the system name of the remote computer or be null. Null system names are not recommended.

A very important feature of `USERFILE` is its ability to restrict file system access. For each line in the file you may specify a list of pathnames separated by commas. Whenever the remote system requests file access, the pathname of the file is compared with this list. For the access to be allowed, one of the pathnames in the list must be a prefix of the complete filename. The following is an example of how you would allow more file system access to the remote system `doosy` by adding to its line in `USERFILE`:

```
Udoosy,doosy   /usr/spool/uucppublic,/usr/doosy
```

To give other users access to parts of your file system, you must modify `/usr/lib/uucp/USERFILE`. The following line in this file will allow all local users to send files and use the mail facility:

```
/usr/spool/uucppublic
```

Notice the comma in the above line; this is a required part of the syntax. The general format for entries in the `USERFILE` file is

[*system*],[*login*] *directory*

where *system* and *login* specify access permission to the named *system* and *login* names. When no system or user is mentioned, access is wide open, but the comma that separates them in the general format must remain in place.

As an added security feature of `USERFILE`, you can use the `c` flag to force a call back to the remote computer instead of allowing it to log in to your computer. To use this feature, insert the letter `c` between the first and second entry on the line, for example,

```
Udoosy,doosy  c  /usr/spool/uucppublic,/usr/doosy
```

6.2 Conversation count checking

The next thing you can do to improve security is to require a conversation count check every time a system calls. A conversation count check means that both systems must record the number of times they communicate with each other and check whether these counts coincide with each other, as explained below. The file used for this purpose is `/usr/lib/uucp/SQFILE`. This file has a line for each system for which you will require the check. To initiate the checking, all you have to do is enter the remote system name into the file as a separate line. The administrator at the other computer will have to add your computer's name to the `SQFILE` on their system. After the first call, the line will look like this:

```
doosy 1 10/15-10:15
```

where `doosy` is the name of the other computer, `1` is the conversation count, `10/15` is the date, and `10:15` is the time of the conversation.

From that point on, the conversation count will be incremented on these corresponding lines every time these two computers are connected via `uucp`. If a call between the two systems is ever made in which these counts do not match, the conversation will fail. Thus, to impersonate another computer you would not only have to know the login name and password, but also the conversation count. If a call attempt ever fails because this file is corrupted on one of the two systems, all you have to do is reinitialize the lines (on both systems) to contain the system names only.

6.3 Controlling file forwarding

The `uucp` utility can forward files through intermediate nodes to get them to another system. If you are going to allow forwarding through your system (that is, make your system a node in the forwarding chain), and you want to have some control over this, you have to make entries in `/usr/lib/uucp/FWDFILE` and `/usr/lib/uucp/ORIGFILE`. The first file, `FWDFILE`, contains a list of systems to which you are willing to forward files via `uucp`. For instance, if you are willing to forward files to the computer `doosy` from other systems, just add a new line with the name `doosy` to the

file. The second file, ORIGFILE, contains a list of systems and users from which you are willing to forward files. Thus, if you are willing to forward files originating with the doosy system by users mark and marian, you would add the following line to the file:

```
doosy,mark,marian
```

If these files do not exist, no restriction applies to forwarding on your system. This means that if you do not have a FWDFILE, you will allow forwarding to all systems to which you connect. Similarly, if you do not have an ORIGFILE, you will allow forwarding to originate on any system. To disable forwarding to any other system, create a FWDFILE with no contents (no systems permitted to be forwarded to). Similarly, you can create an ORIGFILE without any contents to prevent any other computers from using your system to forward files.

6.4 Controlling remote command execution

Another security feature of uucp is its capability to restrict the execution of remote commands. For uucp to execute remote commands, the names of these commands must be listed in the file /usr/lib/uucp/L.cmds, one command name per line. If you want maximum security, you should include a single line in this file with the command rmail, and no other line, so no other remote commands can be executed. Without the rmail line, local users will not be able to get mail from remote systems.

6.5 File permissions

The last issue of security is not a feature of uucp itself but involves the file permissions for the uucp directories, executable files, and administrative files.

In general, the public should be denied read permission for most administrative files, especially /usr/lib/uucp/L.sys. This in turn requires that the owner of uucp's executable files be the same as the owner of the administrative files, and that these be given setuid permission in order to access them. See Chapter 3, "User Administration." Thus, it is recommended that all these files be owned by uucp and that all the binary executables have the setuid bit set, as shown in the recommendations in this section.

The following modes are recommended for directories:

```
755          /usr/lib/uucp
755          /usr/spool/uucp
777          /usr/spool/uucp/.XQTDIR
777          /usr/spool/uucppublic
777          /usr/spool/uucppublic/receive
```

The following modes are recommended for executable files (notice the setuid permissions):

```
4111         /bin/uucp
4111         /bin/uulog
4111         /bin/uuname
4111         /bin/uustat
4111         /bin/uusub
4111         /bin/uux
4111         /usr/lib/uucp/uucico
4111         /usr/lib/uucp/uuclean
4111         /usr/lib/uucp/uuxqt
```

The following modes are recommended for script files:

```
755          /usr/bin/uupick
755          /usr/bin/uuto
400          /usr/lib/uucp/uudemon.day
400          /usr/lib/uucp/uudemon.hr
400          /usr/lib/uucp/uudemon.wk
755          /usr/lib/uucp/uushell
```

The following modes are recommended for uucp system files:

```
444          /usr/lib/uucp/ADMIN
444          /usr/lib/uucp/FWDFILE
444          /usr/lib/uucp/L-devices
444          /usr/lib/uucp/L-dialcodes
444          /usr/lib/uucp/L.cmds
400          /usr/lib/uucp/L.sys
444          /usr/lib/uucp/ORIGFILE
400          /usr/lib/uucp/SQFILE
400          /usr/lib/uucp/USERFILE
```

6.6 The nuucp login environment

A user login for nuucp is set up in the /etc/passwd file in the A/UX standard distribution. You should always log in as nuucp to do UUCP administrative work, because working as the superuser can be dangerous and is not needed when dealing with UUCP system files.

The administrative login entry in /etc/passwd is set up as follows:

```
nuucp::5:5:UUCP admin:/usr/lib/uucp:
```

Note that the directory /usr/lib/uucp has been chosen as the home directory and that the default /bin/sh is the startup shell. If the startup shell chosen were /bin/csh or /bin/ksh, this would appear as the last field on the line. Note also that the nuucp login has the same UID and GID as the uucp login. Because the uucp login occurs first in this file, all files created by the nuucp administrative login will be owned by uucp. This is recommended.

You should assign a password to the nuucp login, and you may also create a .profile file in nuucp's home directory with some helpful shell procedures as follows:

```
cdlib   () { cd /usr/lib/uucp; }
cdpub   () { cd /usr/spool/uucppublic; }
cdspl   () { cd /usr/spool/uucp; }
poll    () { /usr/lib/uucp/uucico -r1 -s$1 &; }
pollx   () { /usr/lib/uucp/uucico -r1 -s$1 -x4 &; }
rmstat  () { rm -f /usr/spool/uucp/ST*; }
taillog () { tail -f /usr/spool/uucp/LOGFILE; }
```

As you get to know UUCP, you will find out how helpful these procedures can be.

The file `/usr/lib/uucp/ADMIN` consists of a list of systems and their descriptions separated by a tab. The entry for the system `doosy` might look like this:

```
doosy          1234 University Avenue,
Sometown, CA
```

Now when you run `uuname` with the `-v` option the description will also be displayed for `doosy`.

Three `uucp` commands are for administrative as well as general use:

```
/bin/uulog
/bin/uustat
/bin/uusub
```

You can use the `uulog` command to display selected portions of `/usr/spool/uucp/LOGFILE`. You may request lines for either a specific system name or a user. For example, to check what has happened with the system `doosy`, issue the command

```
uulog -sdoosy
```

You can use the `uustat` command to find the job number of a `uucp` request and also to cancel a `uucp` request. It is recommended that you use this method to terminate `uucp` jobs if at all possible. For example, if you start a `uucp` file transfer and after you have issued the command you discover that you have requested the wrong file, you first run the following command to get the job number:

```
uustat -sdoosy
```

The status of requests is displayed in reverse chronological order, so your request is near the top. If the job number is 1234, you run the following command to cancel the request:

```
uustat -k1234
```

You can use the `uusub` command to collect and display statistics for the systems with which your system communicates. Before statistics can be collected for a system, `uusub` must first recognize it by adding it to its list. The command to do this is

```
uusub -adoosy
```

You can also use `uusub` to connect to a system. This is usually scheduled by `cron`. If you want to call the `doosy` system every hour on the half hour, the `/usr/spool/cron/crontabs/uucp` entry should look like this:

```
30 **** /bin/su uucp -c "/bin/uusub -cdoosy > /dev/null"
```

See Chapter 7, "System Accounting Package," for a thorough discussion of `cron` entries format.

6.7 Suggested links

The last issue concerning administration is purely conventional. Instead of using names like `tty0` in the files `L.sys` and `L-devices`, it is easier to make links in the `/dev` directory to provide alternate names for ports. The naming convention is used for ports that either have direct links to other computers or have modems attached. If you have `tty0` connected to a modem and `tty5` connected directly to `doosy`, make the following links:

```
ln /dev/tty0 /dev/acu.hayes
ln /dev/tty5 /dev/dir.doosy
```

Then use `acu.hayes` instead of `tty0` in the administrative files and `dir.doosy` instead of `tty5`. This also makes it easier to use `cu` because you don't have to remember the number of the port. See Chapter 5, "Using `cu`" in *A/UX Communications User's Guide*. If the connection is changed to another port, all you have to do is remove the link to the old port and link the name to the new port. None of the `uucp` administrative files need be modified.

6.8 Troubleshooting uucp

There are several things you can do if `uucp` is not working properly on your system.

If you have no information, such as status files or log files, and you have to figure out why `uucp` is not working and fix it, the first thing to do is to make sure the port, modem, and phone line are working. You can use the `cu` utility to determine this. See Chapter 5, "Using `cu`" in *A/UX Communications User's Guide*. The command to check `tty0` would be as follows:

```
cu -ltty0 dir
```

If you cannot even get the "`Connected`" message from `cu`, the port probably has the wrong permissions. To correct this, enter the following command:

```
chmod 666 /dev/tty0
```

At this point you should be able to communicate with the modem and have it dial up the other system. When you get the login and password prompts, use the same login and password that are in `L.sys`. The remote system should respond with something like this:

```
Shere=doosy
```

This means you have reached the `uucico` shell on the remote system and you can do no more using `cu`. Otherwise, contact the administrator of the remote system and explain that your `uucp` login is not working.

Once you have established that the port, modem, and phone line are working, test the connection with the following command:

```
/usr/lib/uucp/uucico -r1 -sdoosy -x4 &
```

Note: You should always run this command in the background so you can kill it if it gets hung. Do not use the `-9` option of `kill` because `uucp` will not catch the signal and clean up after itself. Instead, use `kill` with no options.

To save the debug output in a file, use the `script` program:

```
script /usr/lib/uucp/uucico -r1 -sdoosy -x4 &
```

By comparing the debugging output of this command with the *expect-send* sequence in the `L.sys` file, you can usually tell if something is wrong. For instance, if an entry in the `L.sys` file specifies that the password is `foo` but the password that appears in the debugging output is `fee`, you can then modify `L.sys` and keep testing until it works. By running the `uucico` command with the `-x9` options, you may generate a lot more debug output.

A typical problem is finding strange times in the log files. This generally happens because the time zone environment variable is not set correctly. You can avoid this by making sure that `/usr/lib/uucp/uushell` is the startup script for all `uucp` logins. The contents of this script are

```
exec env TZ=PST8PDT /usr/lib/uucp/uucico
```

Make sure that the setting for `TZ` corresponds to your own time zone.

Most problems with `uucp` occur because of incorrect permissions on files. You should always check this if `uucp` starts working improperly. See "File Permissions," for specific recommendations about the appropriate permissions for the files used by the UUCP system, and Chapter 3, "User Administration," for a general overview of file permissions.