IRIX System Administrator's Guide

IRIS-4D Series

SiliconGraphics
Computer Systems

Document number: 007-0603-030

IRIX System Administrator's Guide

Document Version 3.0

Document Number 007-0603-030

11/90

Technical Publications:
Marcia Allen
Robert Reimann
Harry Max
C J Silverio

Engineering:
Dave Anderson
Andrew Cherenson
Howard Cheng
Brendan Eich
Donl Mathis
Paul Mielke
Vernon Schryver
Chris Wagner

IRIX System Administrator's Guide
Document Version 3.0
Document Number 007-0603-030

Silicon Graphics, Inc.
Mountain View, California

Contents

List of Figures

List of Tables

1. Introduction

This guide explains how to use advanced IRIX system utilities to keep an IRIS-4D Series workstation or server running smoothly. See your *Owner's Guide* first for basic information about your system and instructions on using the System Manager to perform standard system administration. If you have a server, refer to the server owner's guide for basic information about your system but use this guide as a primary reference for administering such a system.

Some of the chapters in this guide contain background information and theory about the IRIX operating system. Others are oriented toward specific administrative tasks. The task-oriented chapters, such as ''Software Installation'', present step-by-step instructions, while the others attempt to give you a conceptual understanding of a topic.

Each system configuration is a little different, which often affects the specific commands you use to administer your system. This guide should help you choose and use the correct methods to keep your system up and running.

Overview of this Guide

Chapter 1, Introduction, provides an overview of this guide. It describes briefly how to use the *sysadm* commands, which conventions are used in this guide, and where to go for additional help.

Chapter 2, "The PROM Monitor", describes the boot environment of the workstation and each of the PROM monitor commands.

Chapter 3, "Software Installation", describes how to use *inst* to install software updates and products.

Chapter 4, "System Security", describes how to prevent unauthorized access to the system and its software.

Chapter 5, "User Services," deals with login administration, user environment, communication services, and user problem resolution.

Chapter 6, "Processor Operations," addresses the standard operations of your workstation. In addition, you will find a list of administrative files and directories.

Chapter 7, "Disk and Cartridge Tape Devices", covers what you need to know about the disk and cartridge tape devices on your workstation.

Chapter 8, "File System Administration," discusses how file systems are organized, how they work, and how to maintain them.

Chapter 9, "Printer Use and Administration," describes how to use and maintain the *lp* spooling system. Installing and remove certains types of printers are covered too.

Chapter 10, "Terminals, Modems, and Dumb Printers," explains how to deal with ttys and terminals including terminal options, modifying line characteristics, and determining line settings.

Chapter 11, "UUCP," explains how to use the UUCP basic networking utilities.

Chapter 12, "The *fx* Disk Utility," contains instructions for using the bad block handling utility, changing the size of the disk partitions, and exercising the disk.

Appendix A, "Device Files," lists the device files and directories on the IRIS-4D Series workstation and server.

Index, references important terms and concepts by subject.

Using the Sysadm Commands

The *sysadm* commands are menu-driven facilities that allow you to perform administrative tasks. The chart on pages 1-4 and 1-5 shows the organization of the *sysadm* menu package. The commands in the *sysadm* menu package fall into five major categories:

- File Management
- Machine Management
- System Setup
- Tty Management
- User Management

Log in as *root*, then type:

```
sysadm
```

You see this display:

```
                SYSTEM ADMINISTRATION

  1 filemgmt          file management menu
  2 machinemgmt       machine management menu
  3 syssetup          system setup menu
  4 ttymgmt           tty management menu
  5 usermgmt          user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP, q to QUIT:
```

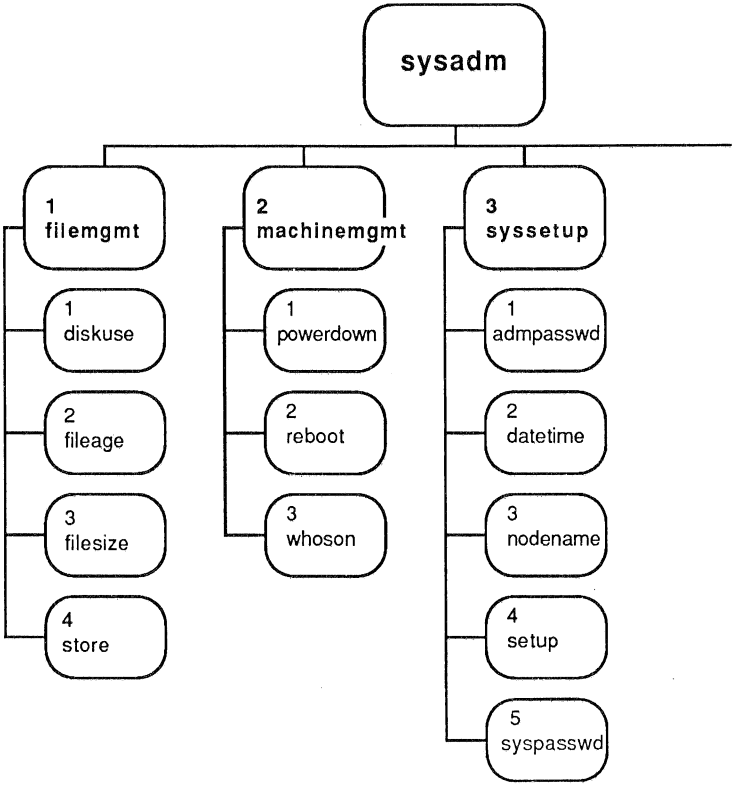
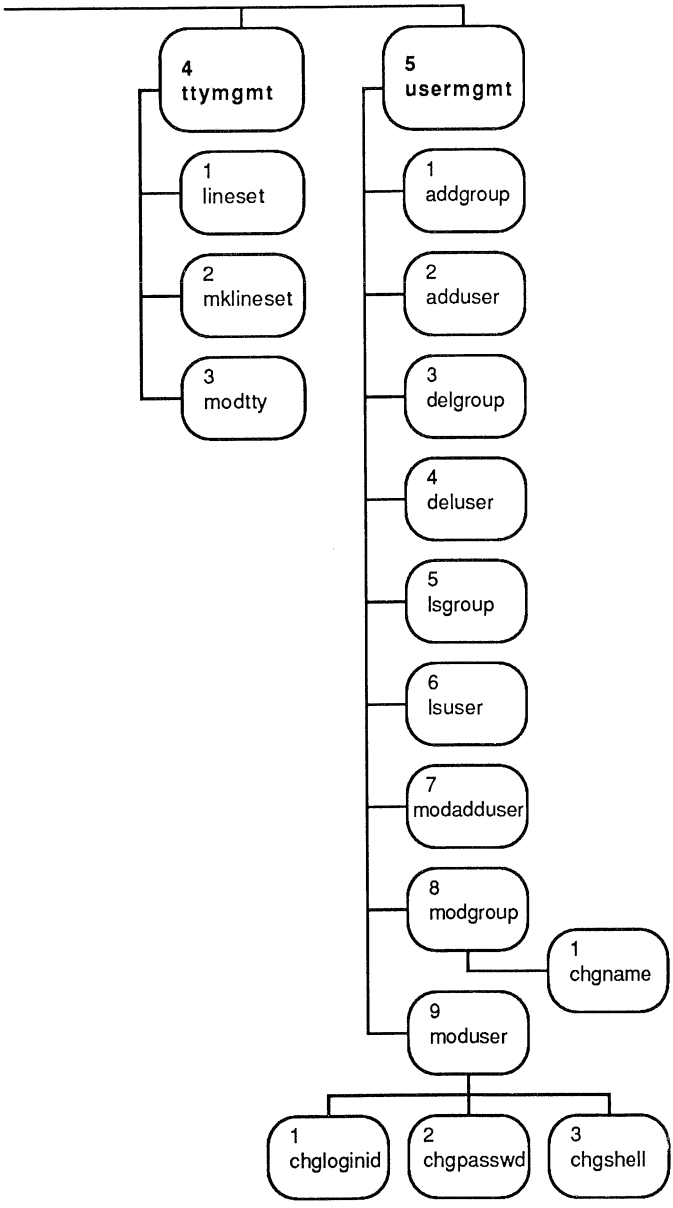


Figure 1-1. Hierarchical Structure of the *sysadm* Menu Package


4
ttymgmt
1
lineset
2
mklineset
3
modtty
5
usermgmt
1
addgroup
2
adduser
3
delgroup
4
deluser
5
lsgroup
6
lsuser
7
modadduser
8
modgroup
1
chgname
9
moduser
1
chgloginid
2
chgpasswd
3
chgshell

Now that you have entered the *sysadm* menu package, you can identify menu entries by number as well as by name. The up arrow key (^) takes you from the second level to the top level in the system administration menu package. The question mark (?) is the help command. A number followed by a question mark (?) produces information about the corresponding command. `q` quits the *sysadm* menu package and returns you to the system prompt.

From the system prompt, you can enter any of the menus directly. For example, to see the menu for File Management, type:

```
sysadm filemgmt
```

From the system prompt, you can also enter specific functions directly.

Conventions

IRIX manual pages, or *man pages*, describe commands, subroutines, and other elements that make up the IRIX operating system. Generally, the man pages are available on-line. To view an on-line man page, type **man** followed by the name of the manual page. For example, to see the on-line man page for *diff*, type:

```
man diff
```

Each element is described in a separate "man" page. The man pages are divided into eight sections, as shown in Table 1-1 below.

In referring to man pages, this document follows a standard IRIX convention: the name of the man page is followed by its section number in parentheses. For example, *cc*(1) refers to the *cc* man page in Section 1. The man pages are divided into six volumes.

The table below shows the volumes and the sections that they contain.

Name of Manual	Section Numbers
IRIS-4D User's Reference Manual	
Volume I	1 (Commands) A-L
Volume II	1 (Commands) M-Z 6 (Demos and Games)
IRIS-4D Programmer's Reference Manual	
Volume I	2 (System Calls)
Volume II	3 (Subroutines)
Volume III	4 (File Formats), 5 (Miscellaneous) Permuted Index
IRIS-4D System Administrator's Reference Manual	1M (Maintenance), 7 (Special Files)

Table 1-1. Outline of Man Page Organization

In addition to the convention for referring to man pages, this guide follows these conventions:

- In command syntax descriptions and examples, square brackets surrounding an argument indicate that the argument is optional. Variable parameters are in *italics*. You replace these variables with the appropriate string or value.
- In text descriptions, filenames, IRIX commands, and PROM monitor commands are in *italics*. IRIS Graphics Library routines and literals (things you type exactly as they are shown) are in `typewriter font`. Labels on the hardware are in Helvetica.
- **`Heavy typewriter font`** shows possible user responses.

Product Support

Silicon Graphics, Inc., provides a comprehensive product support and maintenance program for hardware and software products. For further information, contact your service organization.

2. The PROM monitor

This chapter describes the PROM monitor, which controls the boot environment for the IRIS-4D Series workstations or servers. With the PROM monitor, you can boot and operate the CPU under controlled conditions, run the CPU in PROM monitor mode, and load programs (for example, the operating system kernel, /unix). Note that there are numerous minor differences between machines, and you should refer to your owner's guide for information specific to your machine.

2.1 Front Panel Switches

The IRIS 4D 100, 200, and 300 series systems include a bank of eight switches on the front panel. These switches are intended for use in controlling certain debugging operations. On these systems, you must make sure that these switches all point away from the word OPEN before you set any of the PROM variables. If you change the switch position, you must also reset the machine and reboot.

2.2 Determining the PROM in your IRIS

Your IRIS is equipped with one of two possible generations of PROMs. All generations have the same underlying functionality. To identify which PROM your machine uses, go to the PROM level, by powering up, or shutting down.

If you see a numbered menu, choose option 4, "Enter Command Monitor". If your IRIS displays this prompt:

```
>>
```

then you can use the PROM monitor commands described in this chapter directly.

2.3 Summary of PROM Monitor Commands

Table 2-1 summarizes the PROM monitor commands and gives each command's syntax.

Command	Description	Syntax
auto	Autoboots (no arguments)	auto
boot	Boots with arguments	boot [–f file][–n][args]
disable	Disables console; console can be gfx(0), tty(0), or tty(1)	disable console_device
enable	Enables console console can be gfx(0), tty(0), or tty(1)	enable console_device
help or ?	Prints a PROM monitor command summary	help [command] ? [command]
init	Reinitializes the PROM monitor	init
printenv	Displays the current environment variables	printenv [env_var_list]
setenv	Sets environment variables	setenv env_var string
unsetenv	Unsets an environment variable	unsetenv env_var

Table 2-1. PROM Monitor Command Summary

In addition, some PROM monitors use these commands:

Command	Description	Syntax
go	Transfers program execution to PC or to entry point of last booted program if PC omitted	go <*pc*>
hinv	Displays some system hardware configuration	hinv
version	Displays PROM monitor version	version

Table 2-2. Additional PROM Monitor Commands

2.4 Getting Help

The question mark (?) command displays a short description of a specified command. If you do not specify a command, the ? command displays a summary of all PROM monitor commands. To get help, type either `help` or a question mark (?).

`help` [*command*]
`?` [*command*]

2.5 Using PROM Monitor Commands

The following sections cover these subjects:

- The command syntax notation that this chapter uses
- The function of the commands listed in Tables 2-1 and 2-2

2.5.1 Using the Command Line Editor

You can edit on the command line by using the commands shown in Table 2-3.

Command	Description
<ctrl-h>, <del>, or <backspace>	Deletes previous character
<ctrl-u>	Deletes entire line; question mark (?) prompts for corrected line
<ctrl-c>	If a command is doing input or output, kills current command

Table 2-3. The PROM Monitor's Command Line Editor

2.5.2 Syntax of PROM Monitor Commands

The PROM monitor command syntax is designed to resemble the syntax by commands under the IRIX operating system. This chapter uses IRIX notation for command descriptions:

- Boldface words are literals. Type them as they are shown.
- Square brackets ([]) surrounding an argument means that the argument is optional.
- Vertical lines (|) separating arguments mean that you can specify only one optional argument within a set of brackets.
- *file* means that you must specify a file name. A file name includes device specification as described in Section 2.5.3, Syntax of File Names.

2.5.3 Syntax of File Names

When you specify file names for PROM monitor commands, use this syntax:

```
device([cntrlr,[unit[,partition]]])file
```

- *device* specifies a device driver name known to the PROM.
- *cntrlr* specifies a controller number for devices that may have multiple controllers.
- *unit* specifies a unit number on the specified controller.
- *partition* specifies a partition number within a unit.
- *file* specifies a pathname for the file to be accessed.

If you do not specify *cntrlr*, *unit*, and *partition*, they default to zero. The notation shows that you can specify only a *cntrlr*, a *cntrlr* and *unit*, or all three variables. The commas are significant as place markers.

The PROM monitor defines the devices shown in Table 2-4.

Device Name	Description
dkip	the ESDI disk controller (ips in IRIX)
dksc	the SCSI disk controller (dks in IRIX)
tpsc	the SCSI tape controller (tps in IRIX)
xyl	the SMD disk controller (xyl in IRIX)
ipi	the IPI disk controller (ipi in IRIX)
tty	CPU board duart
tty(0)	the local console
tty(1)	the remote console
gfx	the graphics console
console	the "pseudo console" which may be one of gfx(0), tty(0), or tty(1). See Section 2.6.1, Enabling a Console.
bootp	Ethernet controller using BOOTP and TFTP protocols
tpqic	the quarter-inch QIC02 tape drive

Table 2-4. Device Names for PROM Monitor Commands

The PROM device notation is different from IRIX device notation. Certain environment variables (such as *root* and *swap*) are passed to higher level programs, and often require IRIX notation for the */dev* device name. For example, an ESDI disk partition most commonly used for swap, in PROM notation is:

```
dkip(0,0,1)
```

In IRIX notation, the same disk is:

```
ips0d0s1
```

2.6 Running the PROM Monitor

This section describes the commands that you use to run the PROM monitor. The PROM monitor accepts the commands listed in Table 2-1.

2.6.1 Enabling a Console

The PROM monitor can support a local console and a remote console (alone or simultaneously) through a serial port that provides modem support. The *enable* command enables a device that you want to use as a console. The PROM monitor accepts commands from the specified console and displays output to that console.

`enable` *`console_device`*

console_device can be gfx(0) for the graphics console, tty(0) for a terminal on port 1, or tty(1) for a terminal on port 2.

2.6.2 Reinitializing the Processor from PROM Memory

The *init* command reinitializes the processor from PROM memory. *init* returns you to the monitor program.

2.6.3 The PROM Monitor's Environment

The PROM monitor maintains an environment, which is a list of variable names and corresponding values (the values are actually text strings). These *environment variables* contain information that the PROM monitor either uses itself or passes to booted programs. The system stores some environment variables, which are important and unlikely to change frequently, in non-volatile RAM. If you turn off power to the machine or press the **Reset** button, the system remembers these variables. When you change the setting of these variables using the *setenv* command, the PROM code automatically stores the new values in non-volatile RAM. Table 2-5 shows a list of the environment variables that the system stores in non-volatile RAM.

Several environment variables also exist that affect IRIX's operation. These are not stored in non-volatile RAM, but they do affect the operation of the PROM and of IRIX. See Table 2-6.

Variable	Description
netaddr	Specifies the local network address for booting across the Ethernet. See the *bootp* protocol.
dbaud	Specifies the diagnostics console baud rate. You can change the baud rate by setting this variable (acceptable rates include 75, 110, 134, 150, 300, 600, 1200, 2400, 4800, 9600, and 19200), or you can change baud rates by pressing the <break> key. IRIS will use the dbaud rate for the diagnostics console during the entire system startup. **Note:** Pressing the <break> key changes the baud rate only temporarily; the baud rate reverts to the value specified in `dbaud` or `rbaud` when you press the reset switch or issue an *init* command.
rbaud	Specifies the remote console baud rate. The list of acceptable baud rates is the same as for `dbaud`, above.
bootfile	Specifies the name of the file to use for autobooting, normally a stand-alone shell (*sash*).
bootmode	Specifies the type of boot. The options have these meanings: • **c** – performs a complete cold autoboot, using the file pointed to by the *bootfile* variable to boot the kernel; boots sash, then boots kernel; runs power-on diagnostics. • **m** – (default) goes straight to the PROM monitor; clears memory; runs power-on diagnostics. • **d** – go straight to the PROM monitor; do not clear memory; do not run power-on diagnostics (on IRIS-4D 100, 200 and 300 series systems, this has the same effect as *bootmode m*).

Table 2-5. PROM Monitor Environment Variables Stored in Non-volatile RAM

Variable	Description
console	Specifies which console to use. The options have these meanings: • **G** – graphics console with the Silicon Graphics, Inc., logo in the upper left corner • **g** – (default) graphics console without the logo • **d** – diagnostics terminal attached to the port labeled **1** Caution: disables the graphics terminal, and requires an ASCII terminal on port 1.
root	Specifies (in IRIX notation, such as `ips0d0s0`) the disk that contains the root (/) file system.
keybd	Specifies the type of keyboard used. The default is "df"; it should not be more than two characters. This variable provides a hook to override the normal system mechanism for determining the kind of keyboard installed in the system.

Table 2-6. PROM Monitor Environment Variables Stored in Non-volatile RAM

Variable	Description
showconfig	Prints extra information as IRIX boots. If set through *setenv*, its value must be `istrue`.
initstate	Passed to IRIX, where it overrides the `initdefault` line in */etc/inittab*. Permitted values are `s` and the numbers 0-6. See *init*(1M).
swap	Specifies in IRIX notation the swap partition to use. If not set, it defaults to the partion configured into the operating system, which is normally partition 1 on the drive specified by the *root* environment variable.
path	If set, specifies a list of device prefixes that tell the PROM monitor where to look for a file, if no device is specified.
verbose	If set, tells the system to display detailed error messages.

Table 2-7. Environment Variables that Affect IRIX

When you boot a program from the PROM monitor, it passes the current settings of all the environment variables to the booted program.

Displaying the Current Environment Variables

The *printenv* command displays the PROM monitor's current environment variables.

```
printenv [env_var_list]
```

To change (reset) the variables, see the next section.

Changing Environment Variables

The *setenv* command changes the values of existing environment variables or creates new environment variables.

```
setenv  env_var string
```

env_var is the variable you're setting, and *string* is the value you assign to that variable. To see the current monitor settings, use *printenv*.

When you use *setenv* to change the value of one of the stored environment variables in Table 2-6, the system automatically saves the new value in non-volatile RAM. You do not need to re-enter the change the next time the machine is turned off and then on again.

Setting the Keyboard Variable

If the *keybd* variable is set to anything but the default *df*, the appropriate keyboard translation table is loaded from the volume header of the hard disk. If the keyboard translation table is missing or unable to load, then the default translation table stored in the PROMs is used. The *keybd* variable can be set to any value, but the keyboard translation table should be loaded from the volume header on the hard disk.

Note: This variable overrides the normal system mechanism for determining the kind of keyboard installed in the system. You should not change this variable unless you are performing keyboard diagnostics.

The following *keybd* variables are suggested for international keyboards:

Variable	Description
be	Belgian
da	Danish
de	German
df	the default
fr	French
it	Italian
no	Norwegian
sf	Swiss-French
sd	Swiss-German
es	Spanish
sv	Swedish
uk	United Kingdom
us	United States (available on all models)

Table 2-8. *keybd* Variables for International Keyboards

Removing Environment Variables

The *unsetenv* command removes the definition of an environment variable.

```
unsetenv  env_var
```

env_var is the variable whose definition you are removing (see *setenv*, above). Note that variables stored in non-volatile RAM cannot be unset.

2.7 Booting a Program from the PROM Monitor

This section describes each PROM monitor boot command and shows you how to use it. When you reboot or press the **Reset** button, you start up the PROM monitor.

Caution: Do not press the **Reset** button under normal circumstances, that is, when the workstation is running IRIX.

2.7.1 Booting a Default File

The *auto* command reboots the operating system. It uses the default boot file as though you were powering up the CPU. At the PROM monitor prompt (>>), type:

```
auto
```

or `1` at the numbered menu.

The PROM's environment variable *bootfile* specifies the default boot file. In addition, you must set the environment variable *root* to the disk partition that IRIX uses as its root file system. The *auto* command assumes that the desired image of IRIX resides on the partition specified by *root* of the drive specified in the environment variable *bootfile*.

The *bootfile* name can contain no more than 14 characters. To select a different boot file, see the subsection entitled Changing Environment Variables in Section 2.6.3, "The PROM Monitor's Environment."

2.7.2 Booting a Specific Program

The *boot* command starts the system when you want to use a specific boot program and give optional arguments to that program. The syntax of the *boot* command is:

```
boot  [-f  program]  [-n]  [args]
```

- **–f** specifies the program you want to boot. The program name must contain fewer than 20 characters. If you do not specify this option, the environment variable *bootfile* specifies the default program. *boot* normally loads *sash*.

  When you specify a program, you can include a device specification. If you don't name a device, the PROM monitor uses the device specifications in the environment variable *path*. The PROM monitor tries in turn each device that you specify in *path*, until it finds the program you request, or until it has tried all the devices listed in *path*.

- **–n** means no go: it loads the specified program, but does not transfer control to it. Instead, **–n** returns you to the PROM monitor command environment.

- *args* are variables that the PROM monitor passes to the program you're booting. For an *arg* that starts with a hyphen (-), you must prepend an additional hyphen so that the PROM monitor doesn't think that the argument is intended for itself. The PROM monitor removes the extra hyphen before it passes the argument to the booted program. For more information, see the next section.

For example, to boot the disk formatter/exerciser program (*fx*) from the cartridge tape drive, use this command:

```
boot -f tpsc(,7,)fx
```

Without any arguments, *boot* loads the program specified in *bootfile*.

2.7.3 Booting the Standalone Shell

The PROM monitor has been designed to keep it independent of operating systems and as small as possible. Therefore, the PROM monitor cannot directly boot files residing in IRIX or other operating system file trees. However, the PROM monitor does provide a two-level boot mechanism that lets it load an intermediary program which does understand file systems; this intermediary program can then find and load the desired boot file.

This intermediary boot program is called the *standalone shell*, and is referred to as *sash*. *sash* is a reconfigured and expanded version of the PROM monitor program that includes the modules needed to handle operating system file structures. It also has enhanced knowledge about devices.

After the system software is installed, a copy of *sash* is located in the volume header of the first disk. The header contains a very simple file structure that the PROM monitor understands. You can also boot *sash* from tape or across the network.

When the *boot* command invokes *sash* with arguments, *sash* is loaded. *sash* then loads the file specified by the first argument and passes any subsequent arguments to that file. You can also boot *sash* without arguments. In this case, *sash* operates in interactive command mode. You see the sash prompt:

```
sash:
```

For example, to boot a kernel called *dkip()unix*, use this syntax:

```
boot -f dkip()unix
```

To use the multi-level boot feature, set the PROM environment variable *bootfile* to refer to a copy of *sash*. In normal configurations, setting *bootfile* to *dkip(0,0,8)sash* tells the PROM monitor to load *sash* from the ESDI disk controller 0, disk unit 0, partition 8 (the volume header). Use this syntax:

```
setenv bootfile "dkip(0,0,8)sash"   for ESDI drives
setenv bootfile "dksc(0,1,8)sash"   for SCSI drives
setenv bootfile "xyl(0,0,8)sash"    for SMD drives
setenv bootfile "ipi(0,0,8)sash"    for IPI drives
```

Then issue a boot command, as in this example for an ESDI drive:

```
boot dkip()unix initstate=s
```

The following then occur:

- *boot* loads *dkip(0,0,8)sash*, as specified by *bootfile*, since the boot command doesn't contain a **–f** argument. (A **–f** argument would override the default specified by *bootfile*.)
- *sash* gets two arguments: *dkip()unix* and **initstate=s**, which brings the IRIS up in single-user mode. (Note that the PROM monitor removes the leading hyphen (–) from any argument, so if you use the next layer of software, and need an argument with a leading hyphen, you should put two hyphens in front of it.)
- *sash* loads the file specified by the first argument (*dkip()unix*) and passes the next argument to that file.

Note: Do not issue the *auto* command from *sash* with the bootfile set as shown above. If you do, the system tries to boot *sash* over itself and will exit with an error.

To be able to use the *auto* command from *sash*, you should set *bootfile* to refer to the kernel, for example, *dkip()unix*. Even better, return to the PROM level to use the *auto* command.

2.7.4 Booting Across the Network

At the heart of the operation of diskless workstations is the *bootp* protocol. One of the devices that the PROM monitor can use for booting is the Ethernet. Silicon Graphics, Inc., provides a TCP/IP boot protocol that lets you boot files that reside on another host in the network, if the other host supports the booting protocol. The network booting protocol is the bootp protocol. It is a datagram protocol thats use the User Datagram Protocol (UDP) of TCP/IP to transfer files across the Ethernet.

BOOTP is a DARPA standard protocol supported on the following IRIS workstations:

- IRIS 2400T, 2500T, 3020, and 3030 systems, running GL2-W3.6 or later software
- All IRIS-4D Series workstations

To boot across the network, you must first determine the Internet address of the machine you want to boot. The Internet address is a number assigned by the network administrator of the network to which the system is attached. The format of the number is four decimal numbers between 0 and 255, separated by periods; for example:

```
192.20.0.2
```

Use the *setenv* command to set the *netaddr* environment variable to this address; for example:

```
setenv netaddr 192.20.0.2
```

Booting Across the Network with BOOTP

Once you have set the *netaddr* environment variable, you can use BOOTP to refer to a remote file by using a file name of the form:

```
bootp()[hostname:]path
```

- *hostname* is the name of the host where the file resides. The specified host must run the BOOTP server daemon, *bootp*. If you omit *hostname*, BOOTP broadcasts to get the file from any of the hosts on the same network as the machine making the request. The first host that answers fills the request. Only hosts that support BOOTP can respond to the request. It is safe to omit the hostname only when you know that the path is unique to a particular host, or when you know that all the copies of the file are interchangeable.

hostname can be the name of a host on a different Ethernet from the machine that you are booting, if a *gateway* on the local Ethernet provides a route to the remote host. The gateway must be an IRIS workstation running a *bootp* server that you have configured to do cross-network forwarding.

For more information about booting through gateways, see *bootp*(1M). For more information about the */usr/etc/inetd.conf* configuration file, see *inetd*(1M).

To configure the gateway to permit cross-network forwarding, follow these steps:

1. Log in as *root* or become the superuser by issuing the *su* command.
2. Edit the file */usr/etc/inetd.conf* on the gateway machine. This file configures the *bootp* server, which is started by the *inetd*(1M) daemon.
3. Change the *bootp* description so that *inetd* invokes *bootp* with the –f flag. Find this line:

   ```
   bootp  dgram  udp  wait  root  /usr/etc/bootp  bootp
   ```

   Add the –f flag to the final "bootp" on the line:

   ```
   bootp  dgram  udp  wait  root  /usr/etc/bootp  bootp  -f
   ```

4. Change the *tftp* configuration line in one of the following ways:

   Remove the *–s* flag from the argument list for *tftpd*:

   ```
   tftp   dgram  udp   wait  guest   /usr/etc/tftpd  tftpd -s
   ```

   This allows *tftpd* access to all publicly readable directories. If you are concerned about a possible security compromise, you can instead explicitly list the directories to which *tftpd* needs access. In this case, you need to add */usr/etc*:

   ```
   tftp   dgram  udp   wait  guest   /usr/etc/tftpd  tftpd -s /usr/etc
   ```

   See *tftpd*(1M) and *tftp*(1C) for more information.

5. Signal *inetd* to re-read its configuration file.

```
killall 1 inetd
```

- *path* is the pathname of a file on the remote host.
  For example, this command:

```
boot -f bootp()sgi:/usr/local/boot/unix
```

boots the file */usr/local/boot/unix* from the remote host *sgi*. The command:

```
boot -f bootp()/usr/joe/help
```

boots the file */usr/joe/help* from any host on the network responding to the *bootp* broadcast request that has a file of that name.

2.7.5 Booting from a Resource List

To tell the PROM monitor to load standalone commands from various resources, or devices, set the *path* environment variable (see the subsection entitled Changing Environment Variables in Section 2.6.3, "The PROM Monitor's Environment.") Set the *path* variable as follows:

```
setenv path device alternate_path
```

For example:

```
>> setenv path "dkip(0,0,8) bootp()/altdir/"
>> altbootfile
```

This causes the PROM monitor to boot the file *dkip(0,0,8)altbootfile*. If that file fails, the PROM monitor boots *bootp()/altdir/altbootfile*. If that file also fails, the PROM monitor prints the message "command not found". Note that pathnames are separated with spaces.

Note: If the device specification is contained within a command or by *bootfile*, the PROM monitor ignores *path*. Only *bootp* or volume headers are understood by the PROM.

3. Installing Software

This chapter explains how to install software on your IRIS-4D Series workstation or server. It covers:

- Things to consider before you begin
- Setting up the installation tools
- Installing your software
- Finishing up after you install new software
- Troubleshooting installation problems

3.1 READ THIS Before You Begin

This section contains important information and explains several things you should consider before you start to install software on your IRIS-4D Series workstation or server. Reading this section before you begin will help you avoid problems and confusion as you install your software.

To install new software, you must shut down your system (if it's running) and install special software installation tools. These tools include *inst* and a temporary operating system called the *miniroot*. Carefully read this section before you begin the software installation procedure, because it covers some topics that need to be considered *before* you shut down your system.

Important Before you begin to install your new software, make a complete backup of your system. See your owner's guide for instructions on making backups.

Network Addresses

If you plan to install software from a remote tape drive or a directory on a remote system, *inst* might prompt you for the IP network addresses of your system and, possibly, the name of the remote system you use for installation. *inst* looks to your */etc/hosts* file for these addresses. However, if the file doesn't exist, or if you are referencing a remote system not noted in your system's */etc/hosts* file, *inst* prompts you for the IP network addresses of the systems in question.

Note: If you want to install software remotely over a network through a gateway system, the systems must be able to communicate across the network. Check the file */usr/etc/inetd.conf* on the gateway system to make sure that the last column of the *bootp* entry is:

```
bootp -f
```

See "The *distcp* Command" in these release notes, your Owner's Guide, and the *Network Communications Guide* for more information about using the network.

Once your system is shut down to the PROM level to install software, you do not readily have access to these addresses. Therefore, before you begin to install software, find and record the IP network addresses of the workstations or servers directly involved with your installation.

You can get these addresses by looking at any */etc/hosts* file on your network. You are looking for four numbers separated by periods. For example, a typical network address might look like this:

```
192.26.61.13
```

Either of the following commands should provide you with what you need:

grep *system_name* **/etc/hosts**

or if your system is running Yellow Pages:

ypmatch *system_name* **hosts**

Replace *system_name* in the above command with the name of each system for which you need an IP network address. You should see something similar to this:

```
192.26.61.13     system_name.sgi.com  system_name
```

Record the network addresses in case you need them.

Installing Earlier System Software Releases

Once you've installed the 4D1-3.3 system software, you cannot re-install a previous system software release (or portions of a previous release) without first completely removing IRIX 4D1-3.3. If, for some reason, you must re-install a previous system software release, make a complete backup of all user data, use the *inst* clean option to erase all 4D1-3.3 system and user files, and then restart the installation procedure from scratch. When you have finished re-installing the previous release, use the backup tapes you made to put the user data back on the system. Read the following section for important information on the compatibility of installation tools across different system software releases.

Note: The format of the on-line installation history format is slightly different in Release 4D1-3.3 from what it was in previous releases. When you use the 4D1-3.3 installation tools, the on-line installation history will contain this new format. Once this is done, you cannot use older versions of the installation tools because they will not understand the new format. If it is necessary to install older software once your system has been updated to 4D1-3.3 or newer, use the new installation tools to do so.

Determining Which Installation Mode to Use

There are two modes for installing software: *automatic* and *manual.* The easiest way to install software is to use the *automatic* mode. If it fails (for lack of disk space, for example) or if you want more control over your installation than the *automatic* mode provides, you should use the *manual* mode.

Software is divided into groups of related files called *subsystems*. The *manual* installation mode lets you preview a list of the available subsystems, the subsystems selected for installation, and the ones that have not been selected at all. The *manual* mode also provides other functions that you might need during installation. Use the *manual* mode if you need to:

- specify (or respecify) the installation source. This is a place, such as a tape drive (device) or directory, in which components (such as software images and product descriptors) of a product reside. A product descriptor file is a list of the components in a product that *inst* uses to install that product.
- verify *inst*'s default subsystem selections
- override the default subsystem selections
- use less disk space than is required for installing the currently selected set of subsystems
- make new file systems, control file system mounting or unmounting, or perform additional network initialization tasks
- delete unneccessary subsystems

Special Characters Used in Installation

The software products you receive from Silicon Graphics, Inc., are divided into discrete components, which you must deal with before, during, and after installation.

A broad, internal division of a product, such as on-line manual pages, is called an *image*. An image is divided into *subsystems*, which are made up of related files. Each subsystem has a name that follows this format:

```
product.image.subsystem
```

For example, *ftn.man.fedgetut* is the name of the FORTRAN *edge* manual pages. It is a subsystem of the software image of the FORTRAN product.

Using the *manual* mode, you can specify on which subsystems you wish to operate. Often, you will need to specify more than one subsystem at once.

You can specify a subsystem name individually or use a "pattern" to specify many at one time. A pattern is a subsystem name that contains shell-style metacharacters. These metacharacters allow you to match zero or more regular characters in the available subsystem name(s).

The metacharacters are:

?	matches one character
*	matches any combination of characters
[ ]	matches any enclosed characters or range of characters separated by a dash

Missing components in subsystem names are treated by the system as though asterisks (*) are in their places. The following list contains two hypothetical products, to help you see how special characters and missing components affect subsystem selection. Both products, *abc* and *xyx*, contain two images: one for manual pages (on–line documentation) and one for software.

In this example, each image is divided into five subsystems:

```
abc.man.prgs      xyx.man.prgs
abc.man.games     xyx.man.games
abc.man.demos     xyx.man.demos
abc.man.tools1    xyx.man.tools1
abc.man.tools2    xyx.man.tools2
abc.sw.prgs       xyx.sw.prgs
abc.sw.games      xyx.sw.games
abc.sw.demos      xyx.sw.demos
abc.sw.tools1     xyx.sw.tools1
abc.sw.tools2     xyx.sw.tools2
```

Here are examples of some names and patterns, and the subsystems they identify:

Name	Identified Subsystems
abc	all subsystems in the *abc* product
abc.man	all subsystems in the manual page image of *abc*
**.man*	all subsystems in all manual page images
**.*.demos*	all demo subsystems in all products
.sw.[0-9]	all subsystems in the software images of all products that end with a digit

See Section 3.4.2, "The Subsystem Selection Menu," for more information on selecting subsystems.

3.2 The Installation Procedure

This section describes how to prepare your system for installation and use *inst* to install new software.

3.2.1 Setting Up the Installation Tools

To install software on an IRIS-4D Series workstation or server, you first need to set up the installation tools. The main component of this tool set is the *miniroot*. The *miniroot* is a small, temporary operating system that prevents *inst* from overwriting system files that would be active if the IRIX operating system were running.

When you start setting up the installation tools, you will have to bring the system down to the PROM level. Once the *miniroot* is set up, it automatically invokes *inst*, at which point you can install new software. When you've finished installing your software and reboot your system, IRIX overwrites the *miniroot* (which is stored in *swap*), and your newly installed software is ready for use.

In the event of a power failure during this procedure, rebooting the system returns you to the *miniroot* and *inst* by default. To quit out of *inst* after a power failure, you may select the *quit* option from any of the installation menus and your original system configuration should be restored. If the system does not recover properly, see "Error Recovery," later in this chapter.

Warning: Some systems include a bank of eight switches on the front panel for use in certain debugging operations. If you have such a system, make sure that these switches all point away from the word OPEN before you bring your system down to the PROM level. If you change the switch position, you must reset the system and then reboot.

1. If your IRIS-4D Series workstation or server is not already running, turn it on and bring it to PROM menu. See your owner's guide if you need help with this. If you see this message:

```
Press any key to restart.
```

Press any key on the keyboard. If you see a message similar to this one:

```
Starting the system, press <Esc> to stop.
```

Press **<Esc>**.

2. If your system is already running, log in as *root* and then use *who*(1) to determine whether anybody else is logged into the system. If so, post a system-wide message with *wall*(1) asking them to log off while you install new software. Give people a few minutes to finish what they are doing and to log off.

If you have not already done so, make a complete backup of the system. See your owner's guide or other sections of this manual for more information about making backups.

Type this command to shut down the system:

```
/etc/halt
```

3. If your system displays this:

```
     Starting up the system...
To perform system maintenance instead, press <Esc>.
```

Press **<Esc>**.

Depending on the type of IRIS-4D Series workstation or server you have, you see either the System Maintenance Menu or the PROM monitor prompt, which looks like this:

```
>>
```

If you encounter any of the following conditions, skip ahead to Section 3.2.3, "Using the PROM Monitor" to set up the *miniroot* and continue with the installation procedure.

- You see the PROM monitor prompt >>
- You want to install software from a remote directory. For information on setting up a remote installation directory, read "The *distcp* Command" in Section 3.5.
- Your system has more than one tape drive (you need to specify which tape drive to use for installation)

If none of these conditions are true, continue on to step 4.

4. When you see the System Maintenance menu or, if you want to install software from the tape drive on a remote system, select number 2, "Install System Software".

5. *inst* checks to see if your system has a tape drive. If it is able to verify that you have one, you are are ready to install your software. Go to step number 6 and continue from there.

   If you were planning to use your system's tape drive but *inst* failed to verify its existence, you see a message similar to this:

   ```
   Are you using a remote tape?
   ```

   If you see this message, you may:

   - Type **n** if you plan to install software from your system's tape drive. You see:

     ```
     Enter the name of the machine...
     ```

     Enter the name of your system and go to step number 6.

   - Type **y** if you want to use the tape drive on a remote system. You see:

     ```
     Enter the name of the machine...
     ```

     Enter the name of the remote system and go to step number 8.

- Type **n** if you want to install software from a directory on a remote system. You see:

  ```
  Enter the name of the machine...
  ```

  Enter the name of the remote system and the distribution directory in this format:

  system_name:/*dir_path*/sa

  Replace *system_name* with the name of the computer from which you plan to copy your software. Replace *dir_path* with the path name of the directory where the software image resides. Press **<enter>**.

6. You see a message similar to this:

   ```
   Insert the installation tape, then press <enter>:
   ```

   Insert the Execution Only Environment 1 tape (EOE1), lock it in place, and then press **<enter>**.

7. You should see this message followed by several lines of dots that print across your screen as the installation software is loaded onto the system:

   ```
   Copying installation program to disk.
   ```

   After several minutes, the dots stop printing across the screen and you see the *inst* menu, which indicates that *inst* is running and that you may install your software. You see:

   ```
   Ready to install software.

    Choose an item, then press <enter>:

      1. Automatically install software
      2. Use manual installation features

      3. Help
      4. Quit
   ```

   Go to Section 3.2.2, "Using *inst*" for further instructions.

   Note: If the system has trouble reading the tape, see Section 3.1, "Network Addresses".

8. Continue here if you are troubleshooting problems with tape drive recognition on a remote system.

   Before you type anything, make sure the installation tape is locked in the remote system's tape drive. On the remote system, type:

   ```
   mt rewind
   ```

9. On your system, type the name of the remote system, then press **<enter>**. You see:

   ```
   Copying installation program to disk...
   ```

   After a few minutes, the dots stop printing across the screen and you see the *inst* menu, which indicates that *inst* is running and that you may install your software. You see:

   ```
   Ready to install software.

    Choose an item, then press <enter>:

      1. Automatically install software
      2. Use manual installation features

      3. Help
      4. Quit
   ```

   Go to Section 3.2.2, "Using *inst*" for further instructions. If there is a problem at this point, see the section on "Remote Installation Failure" later in this chapter.

3.2.2 Using *inst*

Once *inst* is automatically invoked, installing software is quite straightforward. This section explains how to use *inst* to install new software on your IRIS-4D Series workstation or server.

1. *inst* displays the main installation menu on your screen.
2. If you are installing from a tape cartridge, insert the tape in the tape drive and lock it in place.
3. Choose installation mode from the menu. Remember, you can install most software automatically. This is the simplest way to install software.

   If *automatic* installation fails (because there is not enough space on the hard disk, for example), or you want interactive control over what you install on your hard disk, use the *manual* mode. See Section 3.1, "Before You Begin", to determine which installation mode you should use.

   Note: You can switch installation modes while you are using *inst*. This is useful when you want to install software from multiple sources.

   Enter the number, the name, or an abbreviation of the menu item you choose and then press `<enter>`.

   You can ask *inst* for *help* at any point during the installation process, except when software is actually being copied to the hard disk. Section 3.4, "The Installation Menus", provides detailed information on the *help*.

   inst runs a series of pre-installation checks to determine which subsystems to install by default. If you are installing a product for the very first time, the default subsystems are predefined. If you are updating a product, the default subsystems are those that replace previously installed, corresponding subsystems.

   New software directly replaces corresponding subsystems already on the disk though they might have different names from their older counterparts. *manual* mode allows you to redefine which subsystems are to be installed.

The pre-installation check tests to:

- ensure that mandatory subsystems are selected
- determine whether prerequisite subsystems are already installed
- determine whether enough space exists in the file system to install all of the selected subsystems

Note: If any of these pre-installation checks fail, you see a warning message and the installation stops. You can use the *manual* mode to resolve such problems and then continue installing your software.

If you are attempting to install from a remote tape or remote directory, *inst* might prompt you for network addresses. (See "Network Addresses" in Section 3.1 for help with this.)

4. If you selected the *manual* mode, skip ahead to step 7 to continue with the installation.

   If you chose to install your software automatically, the software is copied from the source to the hard disk. *inst* notifies you as old versions of subsystems are being removed and new versions are installed.

   If *inst* has trouble getting software across the network, refer to "Network Addresses" or "Remote Installation Failure" (whichever is applicable to your situation).

   At the end of the software installation, *inst* checks for compatibility between the new software being installed and existing software already on the system. If any incompatibilities are found, *inst* notifies you by printing out the names of the incompatible subsystems. These incompatibilities must be resolved before *inst* will let you *quit*. For information on resolving incompatibilities, type:

   ```
   help incompatible
   ```

   When the installation process is complete, you see:

   ```
   Done.
   Is there more software to install?
   ```

5. Indicate whether to install additional software.

   If you don't have any more software to install, type **no** and press **<enter>**.

   If you have more software to install, type **yes** and then press **<enter>**. You see:

```
Insert the next tape, then press <enter>:
```

   Insert the tape and press **<enter>**. You see the following menu again:

```
Choose an item, then press <enter>:

    1. Automatically install software
    2. Use manual installation features

    3. Help
    4. Quit
```

   Repeat the installation procedure until you are finished installing your software.

   Note When you use *quit* to exit, *inst* checks for compatibility with existing software on the system. If any of the products are incompatible with the new software, *inst* notifies you of the incompatibility. Before you are allowed to quit *inst*, you must resolve the incompatibilities.

6. Restart the system.

   When all of your software is installed, you see:

```
Please wait...
Ready to restart the system.  Restart? [y, n]
```

   Type **y**, then press **<enter>**. The system reboots, and your new software is available for use.

   Next, you see a message reminding you to use *versions changed* to see which configuration files changed during installation. Go to Section 3.3, "Finishing Up the Installation", for further instructions.

7. Continue here if you are installing your software using the *manual* mode.

   The general procedure for using the *manual* mode is outlined below, but can vary depending on your particular needs. You must be familiar with the various menus to successfully perform an installation using the *manual* mode. See Section 3.4.1, ''Manual Installation Menus'' for a complete explanation of each menu item function.

8. Use *admin* to perform any necessary file system operations, such as making new file systems, mounting, and unmounting.

9. Use *admin* to perform any nonstandard network initialization that might be required.

10. Use *from* to specify or respecify the installation source if necessary.

11. Use *list* to examine a list of the subsystems to see which are available and which are selected by default.

12. Choose one of the *standard*, *all*, or *select* items. The *standard* and *all* items initiate installation. The *select* item invokes the Subsystem Selection menu. See Section 3.4.2, ''The Subsystem Selection Menu'', to learn more about how to use the *select* option.

13. Repeat steps 8 through 12 above for each additional software product that you are installing.

14. Choose *go*, if you used *select*) to copy the software to your disk. This instructs *inst* to install the selected software on your system.

15. When you are finished installing software, use *quit* to stop the installation tool. At the end of the software installation, *inst* checks for compatibility between the new software being installed and existing software already on the system. If any incompatibilities are found, *inst* notifies you by printing out the names of the incompatible subsystems. These incompatibilities must be resolved before *inst* will let you *quit*. For information on resolving incompatibilities, type:

```
help incompatible
```

16. Restart the system.

    When all of your software is installed, you see:

```
Please wait...
Ready to restart the system.  Restart? [y, n]
```

    Type **y**, then press **<enter>**. The system reboots, and your new software is available for use.

    You see a message reminding you to use *versions changed* to see which configuration files changed during installation.

3.2.3 Using the PROM Monitor

If your system displays the System Maintenance menu, choose number 5, "Enter Command Monitor" to get the PROM monitor prompt >>.

Follow the instructions that apply to your software installation in one of the following subsections:

- From a Local Tape Drive
- From a Remote Tape Drive
- From a Remote Distribution Directory

From a Local Tape Drive

To set up the *miniroot* from a local tape drive, follow these steps:

1. Put the installation tape in the tape drive.
2. Set the environment variable *tapedevice* to the name of your tape device by typing in the following command at the >> prompt and pressing **<enter>**. Replace *device_name* in the command below with the name of the device that corresponds to your tape drive. See Table 3–1 for standard tape drive device names.

```
setenv tapedevice device_name
```

You can use the *hinv* command to determine the SCSI number for your drive:

```
hinv
```

Table 3–1 lists the device names for QIC and SCSI tape drives. *n* is the controller number and *ID* is the SCSI ID number.

Type of Drive	Device Name
QIC tape drives	tpqic(*n*,*ID*)
SCSI tape drives	tpsc(*n*,*ID*)

Table 3-1. Tape Drive Device Names

3. Boot the standalone shell, *sash*, from the tape device. Replace *cpu* in the command below with the appropriate CPU type from Table 3–2.

```
boot -f ${tapedevice}(sash.cpu) --m
```

Model Number	CPU Type
4D/60	R2300
4D/50, 60T, 70, and 80	IP4
4D/85	IP4
4D/120 (always SCSI)	IP5
4D/20, 25	IP6
4D/220, 240, 280	IP7
4D/210	IP9

Table 3-2. CPU Types

If you encounter errors in booting *sash*, see the section "Trouble Reading a Local Tape," later in this chapter.

This command copies the *miniroot* to your hard disk and boots your system from it. After about 10 minutes, you see this message:

```
Ready to install software.

  Choose an item, then press <enter>:
     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit
```

Now you can install your software. See Section 3.2.2, "Using *inst*", for further instructions.

From a Remote Tape Drive

A remote system must be running system software release 4D1-3.2 or later to load the installation tools from a remote tape drive. Follow these steps:

1. If the *netaddr* variable on your system is not set, or is incorrect, you need to set it. (Refer to "Network Addresses" in Section 3.1 for help doing this):

   setenv netaddr *network_address*

2. Make sure the installation tape is locked securely in the tape drive on the remote system.

3. Set the *tapedevice* environment variable as shown in the example below. Replace *remote_system* in the command line below with the name of the remote computer.

   setenv tapedevice bootp()*remote_system***:/dev/nrtape**

4. Boot the standalone shell, *sash*, from the tape on the remote computer. Replace *cpu* with the appropriate CPU type from Table 3–2 in the previous section.

   boot -f ${tapedevice}(sash.*cpu***) --m**

   You see messages similar to these:

   ```
   Obtaining /dev/tape from server remote.
   Copying installation program to disk
   ```

   followed by a series of dots.

   If you have problems booting *sash*, see "Remote Installation Failure," later in this chapter.

After about ten minutes, you see this message:

```
Ready to install software.

Choose an item, then press <enter>:

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit
```

Now you can install your software. See Section 3.2.2, "Using *inst*", for further instructions.

From a Remote Distribution Directory

To load the *miniroot* from a directory located on a remote computer, follow these steps:

1. Use *distcp* to prepare a directory on the remote computer from which you plan to install your software. See "The *distcp* Command" in Section 3.5 for more details.
2. If the *netaddr* variable on your system is not set (or is incorrect), you need to set it. (Refer to "Network Addresses" in Section 3.1 for help doing this). To set the *netaddr* variable, enter:

   setenv netaddr *network_address*

3. Set the *notape* environment variable as shown below:

   setenv notape 1

4. Set the *tapedevice* environment variables as shown below. Replace *distdir* with the complete path of the directory from which you want to install software:

   setenv tapedevice bootp()*remote_system*:*/distdir*/**sa**

5. Boot the standalone shell, *sash*, from the tape on the remote computer. Replace *cpu* with the appropriate CPU type from Table 3–2.

```
boot -f ${tapedevice}(sash.cpu) --m
```

You see messages similar to these:

```
Obtaining /distdir/sa from server remote.
Copying installation program to disk
```

followed by a series of dots.

If there are problems at this point, see "Remote Installation Failure," later in this chapter.

After about 10 minutes, you see this message:

```
Ready to install software.

  Choose an item, then press <enter>:

     1. Automatically install software
     2. Use manual installation features

     3. Help
     4. Quit
```

Go to Section 3.2.2, "Using *inst*" for further instructions.

3.3 Finishing Up the Installation

Adding and updating software often affects configuration files. The type and number of configuration files affected depend upon the product that you are installing.

When you finish installing software on your system, you might see a message reminding you to update your configuration files and remove obsolete ones.

The following list describes how *inst* handles configuration files that have been modified by the user prior to installation:

- If a configuration file on your system should not be updated and cannot be improved upon, *inst* changes nothing.
- If the product you are installing contains a new configuration file with optional improvements, *inst* leaves your files alone and installs the new file for comparison, with *.N* appended to the file name.
- If the product you are installing contains a new configuration file with mandatory improvements, *inst* installs the new file and renames your old one for comparison, with *.O* appended to the file name.
- Obsolete configuration files are renamed with a *.X* suffix. These files are *not* in the installation history of the product, but can be located with the following command:

```
versions user | grep ".*\.X"
```

See "The *versions* Command", in Section 3.5 for more information.

You can find detailed explanations of the purpose and format of configuration files in other sections of this manual, as well as in the *System Tuning and Configuration Guide* and *Network Communications Guide.*

Error Recovery

If *inst* terminates abnormally, the system and the on-line installation history file is abandoned in an undefined state. This renders the product being installed or removed unusable. In such cases, you should correct the cause of the error if possible, and then either re-install or remove the software product that was being installed (or removed) when the problem occurred.

If the system is reset or has a power failure during installation, the system automatically reboots into the *inst* tool. This reduces the chances of having a partially updated system, which would probably result in strange behavior or system crashes later.

In the event that this happens, all that is necessary to restore the old boot partition information is to *quit* normally from *inst*. If, instead, you attempt to reload the *miniroot*, you will get a warning from *sash* that the root and swap partitions are the same. This is usually due to an interrupted installation.

If all else fails, the boot partition must be reset either by *fx*(1) (standalone or kernel) or by *dvhtool*(1M) under UNIX. The *System Tuning and Configuration Guide* contains information on booting from an alternate root partition.

Trouble Reading a Local Tape

If there is a problem in the early stages of the installation process, while the PROMS are attempting to load the *sash* program or copy the *miniroot* to the disk, it might be th result of a problem in reading the tape. Retry the operation at least one more time. Also, try cleaning the tape drive.

If there is still a problem, and there is another system with a tape drive available, try to read the tape on the other drive:

```
mt rewind
dd if=/dev/nrtape of=/dev/null bs=16k
mt rewind
```

This serves only to read through the data of the first file on the tape and copy it "nowhere", and may take several minutes. You should see no error messages, other than a count of records copied in and out.

If the problem occurs later in the process, after *inst* is running and an attempt is made to begin installing software, you might want to verify that the subsequent files on the tape can be read. Note that access is always achieved through the no-rewind variant of the tape device. If you are specifying a non-standard tape device, make sure you are using the no-rewind device. Escape to a shell with the *sh* command, and issue these commands:

```
mt rewind
mt fsf 2
dd if=/dev/nrtape of=/dev/null bs=16k
dd if=/dev/nrtape of=/dev/null bs=16k
dd if=/dev/nrtape of=/dev/null bs=16k
dd if=/dev/nrtape of=/dev/null bs=16k
mt rewind
```

Each of the *dd* commands serves to read through the data of the next physical file on the tape. You should see no error messages other than a count of records in and out.

If the files cannot be read, there might be a problem with your copy of the installation tape and/or the tape drive.

If the files can be read, the tape and tape drive are probably OK. Exit the shell by typing *exit*. Issue a `from tape` command to cause *inst* to try and read from the tape again.

If the problem cannot be isolated using these techniques, call the Geometry Hotline.

Remote Installation Failure

If there is trouble reading from a remote tape drive early in the process, make sure that the tape is readable on the remote system's tape drive, as described in the section on "Trouble Reading a Local Tape". If the tape can be read properly, the problem is probably due to a poor network connection.

If you are attempting to install from a directory on a remote system, double check the installation source system name and path names and then assume that the problem may be due to the network connection.

The PROMS require that the *netaddr* variable contain your system's IP address. If the remote system is being accessed through a network gateway system, the PROMS also require that the *gateaddr* variable contain the gateway system's IP address. The techniques in "Network Addresses," will help you get these addresses. Use the PROM monitor prompt (>>) to examine this (with *printenv*) and/or change the current environment variables (with *setenv*). Double check the addresses and correct them as necessary.

If you are installing through a network gateway system, make sure the gateway system is configured to forward *bootp* requests. Refer to "Network Addresses", in Section 3.1. See also the material under "Booting Across the Network with BOOTP," in Section 2.7.4 of Chapter 2.

If network access fails later in the process (for example, when *inst* is attempting to read the installation software), it might be due to a permissions problem. If you haven't explicitly specified an account name to be used on the remote system (through the *from* menu item), *inst* tries the *guest* account. If *guest* account access is not granted, this attempt fails.

When *inst* is running in the *miniroot*, it initializes TCP/IP just before the first attempt is made to read from the directory on the remote system. Once this occurs, you should be able to test various network connections using some of the standard TCP/IP commands. If you want to verify that a connection can be made to a particular system, through a particular user account, escape to a shell with the *sh* menu item and issue this command:

```
rsh system_name -l user date
```

The remote system should respond with the current date. If this command fails, *inst* will also fail for the same reason. It might be that your system is not configured correctly. Examine the */etc/hosts* file.

If the error is permissions related, the user *inst* on your system is not allowed to use the *user* account on the remote system. There might be a more appropriate *user* account on the remote system. Sometimes, administrators set up a user account (and restricted shell) for use only by *inst*.

To direct *inst* to get software through a particular user account, use the *from* menu item:

```
from user@server:/dev/nrtape
```

or

```
from user@server:/distdir
```

If you wish, you can issue this command as soon as *inst* starts running. See Section 3.4.1, "Manual Installation Menus," for more information on the *from* command.

Recovering from an Unbootable Kernel

The following procedure explains how to recover from an unbootable kernel (*/unix*) and describes how to get a viable version of software running after an unsuccessful reconfiguration attempt.

1. If the system fails to reboot, try to reboot it a few more times. If it still fails, you need to interrupt the boot process and direct the boot PROM to boot from an alternate kernel (for example, */unix.save*).
2. Press the ***reset*** button. You want to get to the Command Monitor prompt that looks like this:

   >>

   If you see the System Maintenance Menu, choose option 5, "Enter Command Monitor".

3. Once you have selected the option for the Command Monitor, find out your *root* disk ''type'' by entering this command:

```
printenv
```

root equals one of these disk types:

Code	Disk Type
dks0d1sw0	SCSI drive
ips0d0sw0	ESDI drive
xyl0d0s0	SMD drive
ipi0d0s0	IPI drive

Table 3-3. Disk Drive Device Names

4. At the Command Monitor prompt >>, enter the appropriate command to boot the alternate kernel. The table below lists the boot commands for each of the three standard disk drive types. *unix.save* is an example of what an alternate kernel might be named.

Type of Drive	Enter
SCSI	boot dksc(0,1,0)unix.save
ESDI	boot dkip(0,0,0)unix.save
SMD	boot xyl(0,0,0)unix.save
IPI	boot ipi(0,0,8)unix.save

Table 3-4. Boot Command Lines

The system should boot the alternate kernel.

5. Once the system is running, make the alternate kernel the real kernel:

```
mv /unix.save  /unix
```

Now you can boot the system normally while you investigate the problem with the new kernel.

3.4 The Installation Menus

inst provides a number of menus that let you control the installation process. Most of these menus apply only to using the manual installation features, but some, such as the Interrupt/Error menu, apply to both installation modes.

The menu items *help*, *set*, *quit*, *sh*, and *shroot* are available from every menu even though they are not listed on the menu. (Two exceptions to this are the *sh* and *shroot* items, which are never available from the Interrupt Menu.) These "hidden" menu items provide administrative support without cluttering up your screen. These items also are available from the the Administration Functions menu, which is covered in detail in Section 3.4.3.

Some menu items can be used only from the *miniroot*. For example, you can invoke items that affect file systems and network initialization only from the *miniroot*.

You can choose an item on a menu by name or by number, confirming the selection by pressing **<enter>**. The names of the menu items are the first words on the line and can be abbreviated if you like.

Some of the menu items accept an argument (a name or number), which affects the way the menu item reacts. *inst* prompts you for an argument if it expects one and you have not supplied it.

3.4.1 Manual Installation Menus

The Manual Installation menu contains these items:

from [source] Specifies the distribution source. The three kinds of source are the no-rewind tape device, a specific product descriptor file in a distribution directory, or a distribution directory. Any of these can be local or remote.

If the distribution source is tape, *inst* tries to position the tape and read the *product descriptor* file. The product descriptor is an internal representation of the product, image, and subsystem hierarchy, providing all necessary information on the product to *inst*.

To prepare a distribution directory, use the *distcp* command to copy software products from tape to disk. You can use the product descriptor files in the directory as distribution sources. You can name the distribution directory itself as a source. If you do this, *inst* reads all product descriptors in the directory. For more information on preparing a distribution directory, see Section 3.5, "Special Features".

If the distribution source is on a remote system that you can access over an Ethernet via TCP/IP, prefix the source name with *host:* or *user@host:*.

Here are some example distribution source names:

```
Manual> from /dev/nrtape
Manual> from dserver:/dev/nrtape
Manual> from guest@dserver:/u/distdir
```

If you encounter problems accessing the distribution, see "Trouble Reading a Local Tape" or "Remote Installation Failure" in this chapter.

list [names] Lists information for the subsystems of the current distribution source. By default, all known subsystems are listed. You can use names or patterns to restrict the listing to specific subsystems.

There are also several special keywords that you can use to identify the subsystems you want to list, including all of the keywords in Table 3–5 (in "The Subsystem Selection Menu" section) and the following:

Keyword	Function
products	lists products, not subsystems
images	lists images, not subsystems
sizes	list absolute subsystem sizes (rather than disk space deltas

All remaining items are taken as subsystem names or patterns.

Each line in the output of the list menu selection describes one subsystem and contains several columns.

If the subsystem is currently selected for installation, the first column is "i". If a version of the subsystem is already installed and will be left alone during this installation, the first column is "k". If an "r" is present, then the subsystem will be removed from your machine.

If the subsystem is already installed on the disk, the second column is "I". If a subsystem from a previous release has been installed on the disk, the second column is "X." An "N" in this column indicates that you are trying to install an older version of the subsystem than what is currently installed on your machine. If there is no related version on the disk, the second column is blank.

The third column is the subsystem name in the form *product.image.subsystem*. This name identifies the subsystem to the various menu items during subsystem selection.

The next columns are the disk space "deltas" for the subsystem. There is one column for the parent directory of each file system, usually / and */usr*. The heading shows the parent directory names. The delta is a number followed by a plus or minus sign to indicate the estimated change in disk space, in 512 byte blocks, that would result from installing that subsystem. (The numbers are estimates based on size of the existing files on the disk, and the size of the new files to be installed.) These numbers can be negative, indicating that the new subsystem requires less room on that file system than the subsystem(s) it replaces.

The final column is the description of the subsystem, indicating the function of the files in the subsystem.

If the *verboselist* option is set, the subsystem list includes the product and image names, delta totals, and descriptions, along with the subsystems they contain.

standard	Installs the default set of subsystems. *inst* performs the standard pre-installation checks, then begins installation.
all	Installs all of the subsystems that are supplied with the distribution.
select	When both *standard* and *all* are inappropriate, use the *select* item to invoke the Subsystem Selection menu, from which you can choose the specific subsystems you want to install, and perform other related functions.
recalculate	Recalculates the disk space deltas. If you remove or otherwise alter system files (i.e. files that have been installed as part of a distribution) during a shell escape, any previous disk space computations will become invalid. If you may have altered the sizes of system files during a shell escape, you should issue a *recalculate* command to recompute the disk space deltas. There is no harm in recalculating more often than necessary, other than the delay. A reminder is given after each shell escape during which the free disk space changes significantly.
clean	Clears all files and directories from the / and */usr* file systems. All existing files and directories on the *root* and *user* file systems are lost. (See *remove* in the Subsystem Selection menu to remove specific subsystems.) *inst* requests confirmation when a file system already exists on the disk, and does not proceed unless you explicitly answer **yes**.

Caution: Clean destroys all files on the / and */usr* file systems. Use it with extreme care, only when you want to discard the entire contents of the disk. For peace of mind, back up the entire system before you invoke the clean item. See "Backing Up and Restoring Your System" in your owner's guide.

admin [cmd]	Invokes the Administration Functions menu. You can return to the Manual menu when you are finished. If you give arguments, *inst* treats them as an administrative function to be executed immediately, without leaving this menu.
return	Returns to previous menu.
help [item]	The standard *help* item. The default help from this menu is a brief description of the menu and the items on it. For additional information, give the topic you want to find out about as an argument to *help*.
quit	Performs certain cleanup tasks and exits the software installation tool, returning control to IRIX or the *miniroot* special configuration program.

3.4.2 The Subsystem Selection Menu

The Subsystem Selection menu lets you control which subsystems to install on the disk, as well as other supporting tasks. You can examine the list of available subsystems, select or de-select them by name, examine a list of the file names in a subsystem, or remove previously installed subsystems from the disk. Once you remove the unwanted subsystems and make the desired selections, you can start the installation.

To determine whether to install a particular subsystem, consider the need for the functionality provided by the subsystem, and the subsystem's size in relation to the available disk space. The subsystem listing should help you by showing the description of each subsystem, and the change in disk space that results from installing it. You can remove at any time subsystems that have been installed but that you no longer need or want. Also, you can install at a later date any subsystems that you did not install initially. The Subsystem Selection menu contains these items:

list [names]
: Lists current subsystem selections. This is the same as the *list* item of the Manual Installation menu; see the complete description above.

install [names]
: Selects the named subsystems for installation. The names can be

  - Product, image, or subsystem names
  - Patterns
  - Any of the special keywords listed in Table 3–5

  Here are some examples:

  install eoe2.sw.demos
  : Selects the demos subsystem of the eoe2.sw image.

  install eoe1
  : All of the subsystems in the eoe1 product.

  install eoe1.*.*
  : Equivalent; all subsystems in eoe1.

  install eoe[12].sw
  : All subsystems in the eoe1.sw and eoe2.sw images.

install X

Selects those subsystems for which an older version is installed on the disk. (See Table 3–5)

remove [names] Selects the named subsystems for removal from your machine.

keep [names] Cancels any *install* or *remove* selections for the identified subsystems.

There are several special keywords that you can use to specify the subsystems operated on by the *list*, *install*, *remove*, and *keep* menu items. Most of these keywords also have a one-letter abbreviation that may be used. Note that these one-letter abbreviations are case-sensitive. Table 3–5 lists the keywords and their abbreviations.

Letter	Keyword	Identifies subsystems which:
i	install	are marked for installation
k	keep	are marked for "keep"
r	remove	are marked for removal
I	installed	are already installed
U	uninstalled	are not installed
X	replaces	are replacements for an older installed version
N	replaced	are replaced by a newer installed version
d	default	are declared "default" by the maker
c	candidate	are subsystems for which this or and older version could (and may) have been installed at some point in the past (In other words, "not new")
(none)	inplace	are declared as being installable without the miniroot by the maker

Table 3-5. Keywords

Where applicable, you can combine these keywords (or letters). However, all tests must pass in order for the subsystem to be included. For example, to identify all subsystems that could have been installed at some point in the past but were not, you would use the keywords `candidate uninstalled` (or `c u`) together.

default [names]

Sets default selections. The pre-installation checks determine which subsystems to install by default. If you install a product for the very first time, the default subsystems are predefined on the installation tape(s). If you update a product, the default subsystems are those that replace previously installed, corresponding subsystems. New software directly replaces corresponding subsystems already on the disk.

Note: The *install*, *remove*, *keep* and *default* menu items change the state of subsystem selection as reflected in the subsystem list; they do not start installation. You can review and alter the selections until you are satisfied with them, and then use the *go* item to initiate installation.

step [names]

Selects subsystems using interactive step mode. This item provides a convenient method of traversing the lists. The easiest way to select individual subsystems within a product or image is to use *step*. You can use the following keyword abbreviations in step mode:

i	install; select this subsystem for installation
r	remove; select this subsystem for removal
k	keep; cancel any install or remove request
d	default; use the default operation
p, -	go to the previous subsystem; `<ctrl-p>` has the same effect
n, +	go to the next subsystem; `<enter>` and `<ctrl-n>` have the same effect
/pat	search for a subsystem matching the pattern *pat* and continue from there. This is a convenient means of starting at a specific subsystem.
l	list subsystems, up to the current subsystem

f list the file names in the current subsystem

h display help for step mode

q quit interactive step mode

step mode also supports the use of the arrow keys:

up-arrow same as "p": go to previous subsystem

down-arrow same as "n": go to next subsystem

left-arrow same as "k": keep this subsystem as is

right-arrow same as "i": select subsystem for installation

Using the **<Shift>** key with any of these keys applies that command to the remainder of the product, rather than just the currently displayed subsystem. (An exception to this is **<Shift>**right-arrow, which is equivalent to **D** rather than **I**.)

Note: The easiest way to select subsystems over large, easily identifiable parts of the product is to use patterns and the *install* or *remove* items. For example, `remove *.man` will select all on-line manual pages for removal.

recalculate Recalculates the disk space deltas. If you remove or otherwise alter system files (i.e. files that have been installed as part of a distribution) during a shell escape, any previous disk space computations will become invalid. If you may have altered the sizes of system files during a shell escape, you should issue a *recalculate* command to recompute the disk space deltas. There is no harm in recalculating more often than necessary, other than the delay. A reminder is given after each shell escape during which the free disk space changes significantly.

go Performs the standard pre-installation checks and then installs or removes subsystems based on the current selections.

files [names]	Lists the names of files in subsystems. By default, lists the names of all files in all subsystems. If subsystem names or patterns are given, *inst* lists only the files in those subsystems.
admin [cmd]	Performs miscellaneous administrative functions. This item invokes the Administrative Functions menu, which is useful for dealing with file systems and network initialization. See "The Administration Functions Menu" below. If you supply arguments, *inst* treats them as an administrative command to be executed immediately, without leaving this menu.
return	Returns to the previous menu from which the Subsystem Selection menu was invoked.
help [keyword]	Displays help information for general or specific subjects. Use *help topics* for a list of available topics and keywords.
quit	Terminates software installation.

3.4.3 The Administration Functions Menu

The Administration Functions Menu contains three kinds of items:

- Items that are hidden on other menus. Such items are included on the administration menu for reference.
- Items that initialize TCP/IP network access.
- Items that work with file systems.

The items related to the network and file systems are useful only in the *miniroot*. The administration menu displays these items:

set [options]

Sets, clears, or lists options. Options take values that are Boolean, integer, or string, depending on their function. Boolean valued functions are set to *on* or *off*.

If you give no arguments, *inst* lists the current settings.

If you give one argument, it should be the name of an option to be set to the default function. The default for Booleans is always *on*. The default for other types depends on their value. If you supply two arguments, *inst* uses the first as the option name and the second as the new value. For example, the following command sets the number of lines displayed on the screen between prompts to continue (e.g., `more`):

```
Admin> set lines 24
```

Use *help set* to find out about the available list of options and how to use them.

sh [cmd]

Escapes to a shell or immediately runs the given command.

If the free disk space count changes significantly during a shell escape, you will be reminded to use the *recalculate* command if you altered or removed any system files.

shroot [cmd]	Escapes to a chrooted shell or immediately runs the given command, chrooted to the standard file system.
host [name] [addr]	Identifies local host name and/or address, setting the hostname of the system and making the appropriate entries in the *miniroot*'s */etc/hosts* file. If you do not provide a *name*, *inst* prompts for it. If the address of the given name is not in the *miniroot*'s */etc/hosts* file, or in the */etc/hosts* from the normal IRIX file system, *inst* prompts for it.
server [name] [addr]	Identifies remote host name and/or address, making appropriate entries in the *miniroot*'s */etc/hosts* file. If you do not provide *name*, *inst* prompts for it. If the address is not given, and is not in the normal file system's */etc/hosts* file, *inst* prompts for it.
mount [name] [dir]	Mounts an additional file system relative to the normal file system's *root*. By default, all file systems listed in the normal file system's */etc/fstab* are mounted during *miniroot* installations. This item makes it possible to mount other file systems. During *miniroot* installations, all normal file systems are mounted relative to */root*, but are displayed without the */root* prefix.
umount [name]	Unmounts a file system. You can use this item to unmount file systems that were automatically mounted, or were mounted explicitly with the *mount* item.
mkfs [names]	Makes new file systems on the named devices. By default, the *root* and *user* file systems are remade; this is aliased as *clean* in the Manual Installation menu. You can name a specific device on which to make new file systems. You can then mount the new file system. **Caution:** This deletes all files on the named device.

versions [options]	Runs the *versions* command. Using *versions remove* at this menu allows you to immediately remove a subsystem. This item is also available from all other *inst* menus as a hidden menu option. See Section 3.5, "Special Features", for more information on the *versions* command.
return	Returns to the previous menu.
help [item]	The standard help item.
quit	The standard quit item; ends the software installation process.

3.4.4 The Interrupt/Error Menu

The Interrupt/Error menu pops up if you interrupt an installation or if *inst* detects an error. To interrupt the installation, press `<ctrl-c>` unless the interrupt characters have been redefined by the *stty* commands in your configuration files. The menu allows you to:

- stop the installation,
- proceed with the installation, or
- abort the installation, without cleaning up.

If you type `<ctrl-c>` (interrupt) while software is not actually being installed, you terminate operation of the current item. Sometimes, the current operation must be complete before the interrupt is acknowledged, so it might take a moment for the operation to stop. If an error occurs, *inst* displays an appropriate error message before it displays the menu.

Note: While *inst* is executing certain ''auxiliary commands'' it will ignore your `<ctrl-c>`. Some of these procedures, which are clearly identified if you *set verbose on*, can take a few minutes to complete at which time *inst* will allow interruptions to occur again.

The Interrupt/Error menu contains these items:

stop	Stops the installation prematurely, before the next file is installed. (There might be a few moments delay while work on the current file finishes.) Use *stop* if the error condition is such that you cannot or should not proceed with installation.
continue	Proceeds with installation, ignoring the interrupt or error condition. Recommended if the error condition is well understood and known to be innocuous, or if you have simply interrupted the installation to *set verbose* on or off.
help [item]	The standard help command. The default help topic is an explanation of the Interrupt/Error menu.

abort — Immediately terminates *inst*, with absolutely no cleanup actions. This is a rather brutal way of stopping installation, and should be used only in extreme cases where *stop* does not seem to work.

Warning: While installing software, *inst* does not let you use the *sh* or *shroot* items to run the *versions* command or another *inst* because the on-line installation history might become damaged. You can, however, access all of the *versions* functions from the administration menu within *inst*.

3.5 Special Features

inst is the main tool you use to install software. Other tools, such as *versions*(1M) and *distcp*(1M), allow you to do certain types of system and installation administration. This section describes these and other installation tools.

The *distcp* Command

distcp(1M) is an IRIX command that lets you copy a software product from the installation tape(s) to a directory on a remote IRIS-4D Series workstation or server. The word *remote* describes a computer that is connected to your IRIS-4D Series workstation or server over a network.

A remote computer might contain hardware or software that you want to use during the installation. Once a product is in such a directory, you can install it on your system from the remote directory instead of using the cartridge tape on your IRIS-4D Series workstation or server. *distcp* does not work in system software releases prior to 4D1-3.2.

Installing software from a remote directory is helpful in situations where many computers must be updated, because network access is generally faster than tape access.

The following example shows how to copy a product from a cartridge tape in the no-rewind tape device (*/dev/nrtape*) to a directory called *dir*. You must always use the no-rewind tape device with the *distcp* command.

```
distcp /dev/nrtape /dir
```

You must use the 4D1-3.3 versi9on of *distcp* on 4D1-3.3 tapes and images. (4D1-3.3 *distcp* can be used to read 4D1-3.2 tapes and images.)

Caution: Do not alter any of the files in a remote distribution directory because they might become unusable.

See the *IRIX System Administrator's Reference Manual* for detailed information about tape drive device names and using *distcp*(1M).

The *versions* command

inst maintains an on-line installation history of the products, images, subsystems, and files that are installed or removed from the system during installation. *versions* is an IRIX command that gives you access to that history and lets you perform limited administrative duties. Using *versions*, you can:

- see which subsystems are installed
- list the file names in a product or subsystem
- locate configuration files
- see whether a configuration file was modified since it was installed
- remove subsystems

One common use of *versions* is to determine which configuration files were changed when new software was installed. When you are finished installing your software, enter the following command to identify the configuration files in question.

```
versions changed
```

Use *diff*(1) to compare the configuration files and a text editor to modify the active version. *versions* is available as a menu item from all installation menus, whether it is listed or not. See *versions*(1M) in the *IRIX System Administrator's Reference Manual* for detailed information.

The *lboot* Command

The *lboot*(1M) command allows an experienced system administrator to rebuild a kernel to reflect changes that were made to the operating system or system configuration files. *lboot* configures a bootable kernel using the files in */usr/sysgen/system*, and the files in the directories */usr/sysgen/boot* and */usr/sysgen/master.d*.

Use *lboot* not only after changing a parameter, but also after adding configurable software subsystems, adding software drivers for new hardware devices, or removing software drivers for hardware devices that no longer exist.

To reconfigure the system, first become the supersuser, and then follow the procedure below.

1. It is best to save the original kernel (in case, for some reason, you need it later):

   ```
   cp /unix /unix.save
   ```

2. Change your working directory to */usr/sysgen* and run *lboot*, specifying the new kernel as "unix.install":

   ```
   cd /usr/sysgen
   lboot -u /unix.install
   ```

3. Send a message to all users logged into your system that you are bringing it down to install a new kernel. After making sure everybody has logged off, reboot the system:

   ```
   reboot
   ```

   When you run the *lboot* command, the system overwrites the current kernel, */unix*, with the kernel you have just created, */unix.install*. An autoconfiguration script, found in */etc/rc2.d/S95autoconfig*, runs during the startup process. (To override the autoconfiguration, you can rename this file. The *autoconfig* script operates when there have been changes to files in */usr/sysgen* or a new device driver has been installed.)

4. System Security

This chapter deals with maintaining the security of your computer system. It includes:

- Security guidelines for setting passwords and permissions to protect the system from unauthorized access.
- Configuring the login process on the graphics console to the level of security you need
- Setting the system date and time
- Naming the workstation
- Aging passwords to control the amount of time passwords can be kept for logins, locking logins to prevent unauthorized use, and protecting administrative commands and logins with passwords
- Preventing unauthorized use of programs conditioned to execute via administrative logins Set-UID and Set-GID
- Managing system accounting

4.1 Important Security Guidelines

Ultimately, system security is the responsibility of all who have access to the system. Some security items to consider are:

- Anyone with physical access to the machine can simply walk off with it
- Access permissions to directories and files should be set to allow only the necessary permissions for owner, group, and others
- All logins need passwords and passwords should change regularly. Do not pick obvious passwords. Six-to-eight character nonsense strings using letters and numbers are recommended over standard names. Logins that are not needed should be either removed or blocked.
- Systems with dial-up ports should have logins but even then a system with dial-up ports is not totally secure. Sensitive information should not be kept on a system with dial-up ports.
- Users who frequently use the *su* command can compromise the security of the system by accessing files belonging to other users without consent. This command is also dangerous since you must know another user's login password to use it. The more users who know a given login and password, the less secure access is to the system. For this reason, a log is kept on the use of the command. Check the file */usr/adm/sulog* to monitor use of the *su* command. The format of */usr/adm/sulog* is described in Appendix A, Directories and Files.
- Login directories, *.profile* files, and files in */bin*, */usr/bin*, and */etc* that are writable by others are not secure.
- Sensitive data files should be encrypted. The *crypt*(1) command together with the encryption capabilities of the editors (*ed* and *vi*) provide protection for sensitive information.
- If you must be away from the data terminal, log off the system. Do not leave a logged-in console or terminal unattended, especially if you are logged in as *root*.

4.2 logins and passwords

The discussion of logins and passwords covers the following:

- System login options
- Use of the *passmgmt* command
- Password aging
- Sample */etc/passwd* entries
- Locking unused logins
- Special administrative logins

4.2.1 System Login Options

For security purposes, a *chkconfig*(1M) option file */etc/config/login.options* is provided. *login* reads this file to determine machine-specific options. See the man pages for *login*(1) and *login*(4) for information in addition to that provided in this section.

To use these options, you should create the file */etc/config/login.options*, and insert the appropriate option keywords. The recognized options are:

```
maxtries=number
disabletime=seconds
syslog= { all | fail }
passwdreq
lastlog
```

The *maxtries* option specifies the number of consecutive unsuccessful login attempts permitted, before disabling the line. After *maxtries* attempts, further login attempts are disabled for for a period of time specified in seconds by the *disabletime* option. Without these options, after five failed login attempts, *login* will sleep for 20 seconds before dropping the line.

The *passwdreq* option forces users who do not have passwords defined for them in the password file to choose a password before logging in.

The *lastlog* option enables login to inform the user of the last login attempt if the login is successful. It lists the most recent login date and the name of the terminal or remote host from which the previous login attempt occurred.

All this login attempt information is recorded in the file */usr/adm/lastlog/username*.

Successful and unsuccessful login attempts can be logged to the file */usr/adm/SYSLOG* if the *syslog* option is set to *all*. Unsuccessful login attempts only are logged if *syslog* is set to *fail*.

The visual login process (*pandora*) does not provide these security options. To have these security functions, the configuration variable *visuallogin* should be turned off. Use the ordinary login process.

4.2.2 Using passmgmt

The *passmgmt* command updates information in the password file */etc/passwd*. Its syntax is as follows:

```
passmgmt -a [-c comment] [-h homedir] [-u uid [-o]] [-g gid]
            [-s shell] name
passmgmt -m [-c comment] [-h homedir] [-u uid [-o]] [-g gid]
            [-s shell] [-l logname] name
passmgmt -d name
```

The **-a** option adds an entry for user *name* to the password file. You may specify a home directory with the **-h** option, but the *passmgmt* command does not create this directory. The new user is without a password until either the user or the administrator creates one with the *passwd*(1) command.

The **-m** option modifies an existing entry for user *name*. Note that all fields in */etc/passwd* except the password field can be modified with *passmgmt*. Passwords and password aging as described below can be modified with the *passwd*(1) command. All other modifications can be accomplished with the *passmgmt* tool, however.

The **-d** option deletes user *name* from */etc/passwd*. It will not remove the user's home directory, or any other files the user may own on the system. You must remove those files manually.

In all of the above three operations, the **-u** option specifies a user-id number, **-g** a group-id number, and **-s** a shell name. For further information on the *passmgmt* command and its options, refer to the man page.

4.2.3 Password Aging

The password aging mechanism forces users to change their password on a periodic basis. It will also prevent a user from changing a new password before a specified time interval. Password aging is selectively applied to logins by editing the */etc/passwd* file, or by use of the *passwd*(1) command. Realistically, password aging forces a user to adopt at least two passwords for a login. If you require more access control than what is provided by password aging, refer to the man page on *login*(1) for information on how to prompt for a second access code as part of the login process.

The password aging information is appended to the encrypted password field in the */etc/passwd* file. The password aging information consists of a comma ",", and up to four bytes (characters) in the format:

,Mmww

The meaning of these fields is as follows:

,	The comma is the delimiter between the password itself and the aging information.
M	The Maximum duration of the password
m	The minimum time interval before the existing password can be changed by the user (in weeks, following the notation in Figure 4-1).
ww	The week (counted from the beginning of 1970) when the password was last changed and two characters, *ww*, are used. You do not enter this information. The system automatically adds these characters to the password aging information.

All times are specified in weeks (0 through 63) by a 64-character alphabet. Figure 4-1 shows the relationship between the numerical values and character codes. Any of the character codes may be used in the four fields of the password aging information.

Character	Number of Weeks
. (period)	0 (zero)
/ (slash)	1
0 through 9	2 through 11
A through Z	12 through 37
a through z	38 through 63

Figure 4-1. Password Aging Character Codes

Two special cases apply for the character codes:

- If *M* and *m* are equal to zero (the code, ..), the user is forced to change the password at the next login. No further password aging is then applied to that login.
- If *m* is greater than *M* (for example, the code, ./), only *root* is able to change the password for that login.

4.2.4 Sample /etc/passwd Entries

Password administration can be set up in a variety of ways to meet the needs of different organizations. Some examples are discussed in the following sections.

Changing Passwords Every Two Weeks

The following shows the password aging information required to establish a new password every 2 weeks (0) and to deny changing the new password for 1 week (/).

1. Here is a typical login/password entry in the */etc/passwd* file for the typical user *anthrax*:

```
anthrax:RTKESmMOE2m.E:100:1:J. Q. Anthracite:/usr/anthrax:
```

2. To cause *anthrax* to change the password at least every 2 weeks, but keep it at least for 1 week, you can add **,0/** to the password field, or you can use the command:

```
passwd -x 14 -n7 anthrax
```

   If you use *passwd*(1), the numbers specified by options **-x** (the maximum field) and **-n** (the minimum field) will be translated to the 64-character alphabet automatically. After the change, the entry looks like this:

```
anthrax:RTKESmMOE2m.E,0/:100:1:J. Q. Anthracite:/usr/anthrax:
```

   After the password entry is changed, *anthrax* will have to change the password at the next login and every 2 weeks thereafter.

3. After *anthrax*'s first login following the change, the system automatically adds the two-character, "last-time-changed" information to the password field.

```
anthrax:RTKESmMOE2m.E,0/W9:100:1:J. Q. Anthracite:/usr/anthrax:
```

   In this example, *anthrax* changed the password in week W9.

Changing the Password One Time Only

The following shows the password aging information required to establish for one time only a new password when the user next logs in (using the **..** code).

1. Here is the typical login/password entry for user *anthrax* again:

```
anthrax:RTKESmMOE2m.E:100:1:J. Q. Anthracite:/usr/anthrax:
```

2. To cause *anthrax* to change the password at the next login (and to cause this only once), you can add the code `,..` to the password field, or you can use the command:

   ```
   passwd -x0 -n0
   ```

   After the change, the entry looks like this:

   ```
   anthrax:RTKESmMOE2m.E,..:100:1:J. Q. Anthracite:/usr/anthrax:
   ```

   After the password entry is changed, *anthrax* will have to change the password at the next login only.

3. After *anthrax* supplies the new password, the system automatically removes the aging code (,..) from the password field:

   ```
   anthrax:EFDNLqsFUj.fs:100:1:J. Q. Anthracite:/usr/anthrax:
   ```

   Note that the encrypted password information has changed.

Changing the Password as root

The following shows the password aging information required to establish a password for a login so that only the *root* user can change the password (using the *./* code).

1. Here is the typical login/password entry for user *anthrax* again:

   ```
   anthrax:RTKESmMOE2m.E:100:1:J. Q. Anthracite:/usr/anthrax:
   ```

2. To prevent *anthrax* from changing the password, you should use the code *./*. After you edit the */etc/passwd* file, adding a comma, period, and slash to the password field, the entry looks like this:

   ```
   anthrax:RTKESmMOE2m.E,./:100:1:J. Q. Anthracite:/usr/anthrax:
   ```

   Now only *root* can change the password for the *anthrax* login. If *anthrax* tries to change the password, a "permission denied" message is displayed.

4.2.5 Locking Unused Logins

If a login is not used or needed, you should remove the entry from */etc/passwd* or disable (lock) the login.

A login is locked by editing the */etc/passwd* file and changing the encrypted password field to contain one or more characters that are not used by the encryption process.

The easiest way to change the */etc/passwd* file in this manner is to use:

passwd **-l** *name*

The option –l instructs *passwd* to change the password field for *name* to ***LK***. The asterisk (*) is an unused encryption character. You may directly edit the password file yourself, if you wish, to change the password field to contain some other descriptive phrase, such as the expression **Locked;**. In this expression, the semicolon (;) is an unused encryption character. The text, however, only serves to remind you that the login is locked.

The other common method of disabling a password is to put an asterisk (*) into the password field. The IRIX */etc/passwd* file as shipped disables unused logins in this manner.

The following line entry from the */etc/passwd* file shows the "locked" *bin* login.

```
bin:*:2:2:System Tools Owner:/bin:
```

4.3 Setting up the Console Login

Login on the graphics console is handled differently from login on ordinary terminals and remote logins over the network. By default, a visual login program replaces the ordinary login process. There are three different possible configurations of the console login process. Information about the visual login process is also contained in the manual page *pandora*(1).

The login process is determined by two system configuration variables, called *visuallogin* and *noiconlogin*. These variables may have the values **on** or **off**. To view the value of these variables, type

```
chkconfig
```

This displays the value of all system configuration variables. To set the value of one of these variables, use the *chkconfig*(1M) command, as in the following example:

```
chkconfig visuallogin off
```

The effect of this command will be seen upon the next console login.

Default (visuallogin on, noiconlogin off) In this mode, the visual login program displays icons of all the users who can legitimately log into the system, and who have a valid home directory. Users may log in by clicking on their icons. If the users have a password, they are required to enter the password. If users do not have a password, they are logged in immediately.

The default login process, while convenient, does not provide maximum security against unauthorized logins on the graphics console. If unauthorized users can gain access to the graphics console, and system security is a major concern, one of the other two settings may be more appropriate.

No Icons (visuallogin on, noiconlogin on) This mode is identical to the default mode, except that the icons of the users are not displayed. Unauthorized users cannot obtain a list of valid users from the login program. For secure installations, this the preferred mode.

Non-visual (visuallogin off, noiconlogin on or off) In the non-visual mode, the ordinary IRIX login process, i.e., via the textport, is used which provides the level of security standard to UNIX.

4.4 Setting the Time and Date

Setting the time and date synchronizes system time with clock time or resets the system time after it has been corrupted.

Caution: If you are setting the date ahead, bring the system to single-user mode.

When you boot IRIX for the first time, the system prompts you to set the time-of-day clock. To set the time and date whenever you are running IRIX, type *sysadm syssetup*, then select item 2 from the System Setup menu; or, issue the *sysadm datetime* command at the system prompt. Follow this sequence:

```
Current time and time zone is: 15:43 PDT
Change the time zone? [y, n, ?, q] n
Current date and time:  Wed. 04/04/90 15:43
Change the date and time? [y, n, ?, q] y
Month    default 04      (1-12): <enter>
Day      default 04      (1-31): <enter>
Year     default 90      (70-99): <enter>
Hour     default 15      (0-23): <enter>
Minute   default 43      (0-59): 42
Date and time will be set to:  04/04/90 15:42.  OK? [y, n, q] y
Wed Apr  4 15:42:00 PDT 1990
The date and time are now changed.
The cron has been restarted to pick up the
new time and/or time zone.
```

Note: The system accepts defaults when you press **<enter>**.

You can also set the clock with the *date* command. You must be logged in as *root* to use *date*, and you should be in single-user mode. The arguments to *date* are in this sequence: month, day, hour, minute, year. See *date*(1M).

4.5 Defining and Changing the System Name

You can set up the system name of the workstation in three ways:

- Use the *hostname*(1) command. This command changes the name of the system while the system is up. The next time you reboot, the system will reassume its original name. For example, to change the name of the system to *birch* while the system is up, until the next reboot, type:

```
hostname birch
```

Caution: Do not change the system name arbitrarily. Coordinate changes with your networking connections.

- Edit the file */etc/sys_id* so it contains only one line that consists of the name of the system. After you edit this file, you must reboot the system to put the name change into effect. Each time you reboot, the system retains this name.

- Use the *sysadm nodename* command, which has the same effect as issuing the *hostname* command, then editing the file */etc/sys_id*. The system name is changed while the system is up and when you next reboot the system. You must reboot the system to change the system name for networking purposes.

  To display the current system name, issue the *hostname* command without arguments, or issue the command `cat /etc/sys_id`.

  The example below shows how to use *sysadm nodename* to change the system name.

```
Running subcommand 'nodename' from menu 'syssetup',
SYSTEM SETUP

This machine is currently called "iris".
Do you want to change it? [y, n, ?, q] y
What name do you want to give it? [q] ABcd5678
```

4.6 Special Administrative Passwords

There are two familiar ways to access the system: either via a conventional user login or the *root* login. If these were the only two ways to access the system, however, effective use of the system would have to be curtailed (because *root* would own many directories) or many users would have to know the *root* password (a bad security risk) or the system would be wide open (because *root* would own few directories). All of these conditions are undesirable.

Special system logins and administrative commands allow you to successfully combine system security measures and normal system use. Logins can be protected by establishing a password in */etc/passwd*. As such, the *sysadm* command is a login and can be password-protected.

These login/commands allow access to selected directories and system functions. If you log in to the system with *sysadm*, for example, the system executes the command after login and exits to the *login* prompt once you quit or complete the function performed by the command.

Most of these system functions allow a user access to critical portions of the operating system. For this reason, it is recommended that you assign such a login a password, or lock the account using one of the methods described above.

Once you assigned a password, any user attempting to log in using one of these commands as a login (and any user attempting to execute one of these commands from the shell) is prompted for the password. The passwords to the *sysadm* command/login should be known only to a few users.

Login	Use
root	This user has no restrictions on it and it overrides all other logins, protections, and permissions. It allows the user access to the entire operating system. The password for the *root* login should be very carefully protected.
sys	This user has the power of a normal user login over the files it owns, which are in */usr/src*. Its login should be disabled.
bin	This user has the power of a normal user login over the files it owns, which are in */bin*. Its login should be disabled.
adm	This user has the power of a normal user login over the object files it owns, which are located in */usr/adm*. You may *su* to the adm login. This login should be disabled.
uucp	This login owns the object and spooled data files in */usr/lib/uucp*.
nuucp	This login is used by remote machines to log into the system and initiate file transfers via */usr/lib/uucp/uucico*.
daemon	This user is the system daemon, which controls background processing. Its login should be disabled.
lp	This user owns the object and spooled data files in */usr/spool/lp*. Its login should be disabled.
sysadm	This user owns the files in */usr/admin*, and runs the IRIX system manager tools.

4.7 Set-UID and Set-GID

The set-user identification (set-UID) and set-group identification (set-GID) bits must be used very carefully. These bits are set through the *chmod*(1) command and can be specified for any executable file. When a user runs an executable file that has either of these bits set, the system gives the user the permissions of the owner of the executable.

System security can be compromised if a user copies another program onto a file with **-rwsrwxrwx** permissions. For example, if the switch user (*su*) command has the write access permission allowed for others, anyone can copy the shell onto it and get a password-free version of *su*. The following paragraphs provide a few examples of command lines that can be used to identify the files with a set-UID.

For more information about the set-UID and set-GID bits, see *chmod*(1) and *chmod*(2).

4.7.1 Check Set-UIDs Owned by root

The following command line:

```
find / -user root -perm -4000 -exec ls -l {} \;| mail root
```

lists all set-UID files owned by *root*. The results are mailed to *root*. All mounted paths are checked by this command starting at /. Any surprises in *root*'s mail should be investigated.

```
-r-sr-xr-x   1 root     bin       38836 Aug 10 16:16 /usr/bin/at
-r-sr-xr-x   1 root     bin       19812 Aug 10 16:16 /usr/bin/crontab
-r-sr-xr-x   1 root     bin       27748 Aug 10 16:16 /usr/bin/shl
---s--x--x   1 root     sys       46040 Aug 10 15:18 /usr/bin/ct
-r-sr-xr-x   1 root     sys       12092 Aug 11 01:29 /usr/lib/mv_dir
-r-sr-sr-x   1 root     bin       33208 Aug 10 15:55 /usr/lib/lpadmin
-r-sr-sr-x   1 root     bin       38696 Aug 10 15:55 /usr/lib/lpsched
---s--x---   1 root     bignam    45376 Aug 18 15:11 /usr/bignam/bin/sh
-r-sr-xr-x   1 root     sys       11416 Aug 11 01:26 /bin/mkdir
-r-sr-xr-x   1 root     sys       11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     bin       12524 Aug 11 01:27 /bin/df
-rwsr-xr-x   1 root     sys       21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys       23000 Aug 11 01:27 /bin/passwd
-r-sr-xr-x   1 root     sys       23824 Aug 11 01:27 /bin/su
```

In this example, an unauthorized user (**bignam**) has made a personal copy of */bin/sh* and has made it set-UID to *root*. This means that **bignam** can execute */usr/bignam/bin/sh* and become the super user.

4.7.2 Check Set-UIDs in the Root File System

The following command line reports all files with a set-UID for the root file system. The *ncheck*(1M) command, by itself, can be used on a mounted or unmounted file system. Only the superuser may use *ncheck*. The normal output of the *ncheck –s* command includes special files. Here, the *grep* command is used to remove device files from the output. The filtering done in this example to remove the device files is applicable only for the root file system. The output of the modified *ncheck* is used as an argument to the *ls* command. The use of this *ls* command:

```
ls -l `/etc/ncheck -s /dev/root | cut -f2 | grep -v dev`
```

is possible only if the file system is mounted. In this example of the resulting output, nothing looks suspicious.

```
-r-sr-xr-x   1 root     bin        12524 Aug 11 01:27 /bin/df
-rwxr-sr-x   1 root     sys        32272 Aug 10 15:53 /bin/ipcs
-r-xr-sr-x   2 bin      mail       32852 Aug 11 01:28 /bin/mail
-r-sr-xr-x   1 root     sys        11416 Aug 11 01:26 /bin/mkdir
-rwsr-xr-x   1 root     sys        21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys        23000 Aug 11 01:27 /bin/passwd
-r-xr-sr-x   1 bin      sys        27964 Aug 11 01:28 /bin/ps
-r-xr-sr-x   2 bin      mail       32852 Aug 11 01:28 /bin/rmail
-r-sr-xr-x   1 root     sys        11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     sys        23824 Aug 11 01:27 /bin/su
-r-xr-sr-x   1 bin      sys        21212 Aug 10 16:08 /etc/whodo
```

4.7.3 Check Set-UIDs in Other File Systems

The command line:

```
/etc/ncheck -s /dev/usr | cut -f2
```

entry shows the use of the *ncheck* command to examine the *usr* file system (*/dev/usr*, assuming a single-disk system with default partitioning) for files with a set-UID. In this partial example, the complete pathnames for the files start with */usr*. */usr* is not part of the *ncheck* output. In this example, the */usr/bignam/bin/sh* should be investigated.

```
/dev/usr:
/bin/at
/bin/crontab          /lib/mv_dir
/bin/shl              /lib/expreserve
/bin/sadp             /lib/exrecover
/bin/timex            /lib/accept
/bin/cancel           /lib/lpadmin
/bin/disable          /lib/lpmove
/bin/enable           /lib/lpsched
/lib/reject           /lib/lpshut
/lib/sa/sadc          /bin/lp
/lib/uucp/uucico      /bin/lpstat
/lib/uucp/uusched     /bin/ct
/bin/uucp             /bin/cu
/bin/uuname           /lib/uucp/uuxqt
/bin/uustat           /bignam/bin/sh
```

5. User Services

This chapter deals with providing a variety of services to the users of your IRIX system, including:

- Assigning user and group IDs to persons authorized to be users of your system and maintaining the */etc/passwd* and */etc/group* files.
- Setting up a master profile and helping users develop individual profiles and establish environment variables.
- Establishing and maintaining such communications services as message-of-the-day, news, and mail.
- Developing an organized plan for responding to user problems.

5.1 Login Administration

5.1.1 Adding Users with sysadm

To add users to the system using the *sysadm* tools, either select item **2** from the User Management menu, or issue the *sysadm adduser* command at the system prompt.

In order to use the *sysadm adduser* command, follow these steps:

1. Invoke *sysadm adduser*.

   Anytime you want to quit, type **q** If you are not sure how to answer any prompt, type **?** for help, or see the owner's guide. If a default appears in the question, press **<enter>** for the default.

2. Enter user's full name and login ID when prompted.
3. The system then asks you if you would like a YP (Yellow Pages) style entry. You need NFS installed on your system before you may choose this option. Enter **y** if you would like the system to match the information for this login with the information in the master password file on the network server. Otherwise, answer **n**.
4. If you do not choose a YP style entry, enter in turn each parameter for which you are prompted. Press **<enter>** to choose the defaults.

   If you choose a YP style entry, skip the next four steps and continue from there.
5. The system prompts you for a user ID number, a group ID number, a home directory, and a default shell. The system supplies reasonable defaults for all of these parameters.
6. The system prompts you as to whether you really want to add the user.
7. If you choose to keep the account you are creating, assign a temporary password to the new user's account when you are prompted to do so.
8. You are prompted as to whether you want to add another user. If you chose not to use a YP style entry and do not want to add another user, you can stop here.
9. If you choose a YP type entry, the system tries to match the login ID in the master password file on the network server. If it succeeds, then it also matches the password and the name of the home directory in the master password file, and adds an entry to */etc/passwd*. If the user's login ID is 'loginid', the system adds an entry for '+loginid' to */etc/passwd*. The prepended plus (+) indicates that the entry is a YP type entry.

   After you provide the login ID, the system prompts you to install the entry.

   Note: If you choose a YP type entry for a new login, do not try to change the password for this login by hand. Let the system match the password to the master password file.

If the system does not succeed in matching the login ID in the master password file on the network server, you see a message telling you to edit the master password file. In this case, the system takes no local action, and does not add an entry to */etc/passwd*.

See the chapter entitled The YP Service in the *NFS User's Guide* for more information on Yellow Pages.

By default, the system assigns the C shell to new logins. To change a user's shell to the Bourne shell (or to another shell, such as the Korn shell), see "Changing User Information".

When you add a new user with *sysadm adduser*, the system copies default shell startup files from */etc* to the user's home directory. The files in */etc* are called *stdprofile*, *stdlogin*, and *stdcshrc*. The system copies them to the user's home directory as *.profile*, *.login*, and *.cshrc*. Users can customize these files to suit their own needs.

The Bourne shell and the C shell read files in a user's home directory each time a login shell is started. The Bourne shell looks for a file called *.profile*; the C shell looks for a file called *.login*. In addition, the C shell reads a file called *.cshrc* in a user's home directory each time a shell is started. These files are called *shell startup files*.

The system assigns the C shell and copies the shell startup files whether or not the login is a YP type entry. For more information on the Bourne shell and the C shell, see the chapters on these topics in the *IRIS-4D User's Guide*.

To identify new groups to the system, follow these steps:

1. Select item **1** from the User Management menu or enter the *sysadm addgroup* command at the system prompt.
2. Follow this sequence, type:

   ```
   sysadm addgroup
   ```

   Anytime you want to quit, type **q**. If you are not sure how to answer any prompt, type **?** for help, or see your Owner's guide.

   If a default appears in the question, press **<enter>** for the default.
3. You are prompted to enter the group name and ID. Do so now.

4. You are prompted as to whether you wish to keep this entry. Enter the letter that corresponds to `install`, `edit`, `skip`, or `quit`.
5. You are prompted as to whether you want to add another group. You may do so now or stop here.

5.1.2 Adding Users Manually

Before users are permitted to log in to your system, they must be listed in the */etc/passwd* file. The *adduser* selection from the *sysadm usermgmt*(1) menu leads you through a series of prompts that create an entry in the */etc/passwd* file.

If you prefer, you can make the necessary changes to the *passwd* file yourself using an editor. You need to login as *root* to do this; */etc/passwd* is generally installed as a read-only file. An entry in the *passwd* file consists of a single line with seven colon-separated fields.

```
bugeye:gOQ3xsv05bWuM,9/TA:103:123:Allen B. Cipher:/usr/bugeye:
```

The fields are:

login name	A valid name for logging onto the system. A login name is from three to six characters; the first character must be alphabetic. It is usually chosen by the user.
password	The encrypted form of the password, if any, associated with the login name in the first field. All encrypted passwords occupy 13 bytes. The actual password can be a maximum of 8 characters. At least one character must be numeric. This is to discourage users from choosing ordinary words as passwords. When you add a user to the file you may use a default password, such as *shards*, and instruct the user to change it at the first login. Following the encrypted password, separated by a comma, there may be a field that controls password aging. See Password Aging in Chapter 4, "System Security".

user id	The user-ID number (**uid**) is between 0 and 60,000. The number must not include a comma. Numbers below 100 are reserved. User-ID 0 is reserved for the super-user. The System Administration menu package does not permit you to specify a number below 100 when adding a user.
group id	The same conditions apply to the group-ID (**gid**) number as to the **uid**, except that the group-ID 1 is reserved for the "other" group.
account	The account is the name of and optional additional information about the user. There are three optional fields separated by commas: *name, office,* and *phone.*
home directory	The home directory is where the user is placed upon logging in. The name is usually the same as the login name, preceded by a parent directory such as */usr*. The **modadduser** menu of the System Administration package allows you to specify the default parent directory. The home directory is the origination point of the user's directory tree.
program	This is the name of a program invoked at the time the user logs in. If the field is empty, the default program is */bin/csh*. This field is most commonly used to invoke a special shell, such as */bin/sh* (Bourne shell).

As noted above, the password field may contain a subfield that controls the aging of passwords. A description of how the process works can be found in Chapter 1 and in *passwd*(4) in the *IRIS-4D Programmer's Reference Manual.* The effect is to force users periodically to select a new password. If password aging is not implemented, a user can keep the same password indefinitely.

If you inspect the */etc/passwd* file on your workstation you will see several commands listed among the user login names. These are commands, such as *sysadm*(1), that can have passwords assigned to them.

5.1.3 Changing or Deleting Password Entries

As with adding users, there are *sysadm usermgmt* menu selections for changing or deleting user entries from the */etc/passwd* file. You have the option, however, of using an editor to make the changes.

5.1.4 Changing User Information with sysadm

This procedure covers three functions:

- Changing the default values in the *adduser* sequence (*modadduser*)
- Changing the name of a group (*modgroup*)
- Changing a user's login, password, or login shell (*moduser*)

Changing the *adduser* Sequence

To change the default values in the adduser sequence, select item 7 from the User Management menu or issue the *sysadm modadduser* command at the system prompt.

This command allows you to change these default values:

- The group number that new users receive
- The name of the parent directory for users' home directories

To change the values for an individual user's login, including a user's default login shell, see the subsection entitled "Changing a User's Login Information" in this section.

To change the default group number from 1 to 100, type this command:

```
sysadm modadduser
```

Enter a response to each prompt:

```
Running subcommand 'modadduser' from menu 'usermgmt'
USER MANAGEMENT


Anytime you want to quit, type "q".
If you are not sure how to answer any prompt, type "?" for help,
or see the Owner/Operator manual.

Current defaults for adduser:
group ID                1       (other)
parent directory        /usr/people
Do you want to change the default group ID? [y,n,?,q] y
Enter group ID number or group name [?, q] 100
Do you want to change the default parent directory? [y,n,?,q] n

These will be the new defaults:
group ID:               100
parent directory:       /usr/people
Do you want to keep these values? [y, n, q] y
Defaults installed.
```

Changing the Name of a Group

To change the name of a group, select item **8** from the User Management menu, then select item **1** from the Modify Group menu; or, issue the *sysadm chgname* command at the system prompt.

Enter appropriate responses to the follow sequence of prompts:

```
Running subcommand 'chgname' from menu 'modgroup',
MODIFY GROUP ATTRIBUTES

Anytime you want to quit, type "q".
If you are not sure how to answer any prompt, type "?" for help.

Which group name do you wish to change [q]: oldname
          Group name:  oldname
          Group ID:    000

Enter new group name [?, q]: newname
Do you want to change the name of the group 'oldname'
     to 'newname'? [y, n, ?, q] y

The name of the group 'oldname' has been changed to 'newname'.
Do you want to change the name of another group? [y, n, q] n
```

Changing a User's Login Information

This section tells you how to change the values for an individual user's login.

Note: You cannot use these commands for a login you installed as a YP (Yellow Pages) type entry. You must change the master */etc/passwd* file on the network server. See the chapter entitled "The YP Service" in the *NFS User's Guide* for more information about Yellow Pages.

To change the values for a login that you did not install as a YP type entry, follow these steps:

1. Select item **9** from the User Management menu, or issue the *sysadm moduser* command at the system prompt.

   You see this menu:

```
Running subcommand 'moduser' from menu 'usermgmt'
MODIFY USER'S LOGIN

1 chgloginid      change a user's login ID
2 chgpasswd       change a user's password
3 chgshell        change a user's login shell

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP, q to QUIT:
```

2. To change the login name for a user, select item **1** from this menu, or issue the *sysadm chgloginid* command at the system prompt. The system prompts you for a new login ID.
3. To change a user's password, select item **2** from this menu, or issue the *sysadm chgpasswd* command at the system prompt. The system prompts you for a new password.

4. When you first give a user a login ID, the system assigns a default login shell. The *chgshell* subcommand allows you to assign a different shell. To change a user's login shell, select item **3** from the Moduser menu, or issue the *sysadm chgshell* command at the system prompt. You see a sequence similar to the one below, which gives user *tammyb* a Bourne shell instead of a C shell. For a more thorough discussion of the Bourne shell and the C shell, see the chapters on the Bourne shell and C shell in the *IRIS-4D User's Guide.*

```
Running subcommand 'chgshell' from menu 'moduser',
MODIFY USER'S LOGIN

Enter user's login ID [?, q]: tammyb
The current shell is /bin/csh
Enter new shell command[q]: /bin/sh

Do you want to change the login shell of another login?
[y, n, q] q
```

5.1.5 Removing Users/Groups with sysadm

Listing Users or Groups

The two *sysadm* subcommands in this procedure allow you to list the groups and users that the system recognizes. To list groups, select item **5** from the User Management menu, or issue the *sysadm lsgroup* command at the system prompt.

You see a report similar to this one:

```
Groups currently in the computer
(press <RETURN> to start listing each time you hear the bell)
<enter>

group          group          logins permitted to become
name           number         members using newgrp
-----          ------         ----------------------------
adm            3              root,adm,daemon
bin            2              root,bin,daemon
daemon         1              root,daemon
demos          997
diag           996
games          999
guest          998
lp             9
mail           4              root
nuucp          10             nuucp
other          995
rje            8              rje,shqer
root           0              root
sys            3              root,bin,sys,adm
user           20
uucp           5              uucp
#
```

To list users, select item **6** from the User Management menu, or issue the *sysadm lsuser* command at the system prompt. You see a report similar to this one:

```
Users currently in the computer
(press <RETURN> to start listing each time you hear the bell)
<enter>

login name      user name
----------      ---------
adm             Accounting Files Owner
bin             System Tools Owner
daemon          Unused
demos           Demonstration User
diag            Hardware Diagnostics,,
guest           Guest Account,,
lp              Print Spooler Owner
man             On-line Manual Owner
nuucp           Remote UUCP User
root            Super-User
sys             System Activity Owner
sysadm          System V Administration
tutor           Tutorial User
uucp            UUCP Owner
```

Deleting a Group

To delete a group ID, follow these steps:

1. Select item **3** from the User Management menu, or issue the *sysadm delgroup* command at the system prompt.
2. Follow this sequence:

```
Running subcommand 'delgroup' from menu 'usermgmt'
USER MANAGEMENT

Which group name do you wish to delete?[q] seventy7
Do you want to delete group name 'seventy7', group ID 45201?
[y, n, ?, q] y
seventy7 has been deleted
Do you want to delete any other group?[y, n, q] q
```

The *sysadm delgroup* command deletes only the specified group and not the user login(s) assigned to that group. Use *sysadm deluser* to delete the logins belonging to the group.

Deleting a User

This procedure deletes the user's home directory and all the files in and below that directory. The system prompts you to make sure you want to do this. Follow these steps:

1. Select item **4** from the User Management menu, or issue the *sysadm deluser* command at the system prompt.
2. Follow this sequence:

```
Running subcommand 'deluser' from menu 'usermgmt',
USER MANAGEMENT

This function COMPLETELY REMOVES THE USER, their mail file,
home directory and all files below their home directory
from the machine.  Once this is done, there is no way
guaranteed to get them all back.

         BE SURE THIS IS WHAT YOU WANT TO DO!

Enter login ID you wish to remove[q]: tammyb
         'tammyb' belongs to 'John Q. Public'
         whose home directory is /usr/people/tammyb
Do you want to remove login ID 'tammyb'?[y, n, ?, q] y

/usr/people/tammyb and all files under it have been removed.

tammyb has been completely removed.
Enter login ID you wish to remove [q]: q
```

Every so often a user forgets his or her password. When that happens (let's say to user *bugeye*), you can login as *root* and enter the command:

```
passwd bugeye
```

Enter an appropriate response to each prompt. You see:

```
New password: shards                    (The password entered is not echoed.)
Re-enter new password: shards
```

Because you did this as the super-user (*root*), you were not prompted for the old password. The command changes *bugeye*'s password to *shards*. You should make sure the user changes the password immediately.

When you delete a login from */etc/passwd* using the *sysadm usermgmt* menu, all of the user's files and directories are removed. If you remove an entry from the *passwd* file using an editor, you have only removed the entry. The user's files remain.

5.1.6 Group IDs

Group IDs are a means of establishing another level of ownership of and access to files and directories. Users with some community of interest can be identified as members of the same group. By default, files are created with the group-ID of the user. This behavior can be changed (see *chmod*(2)). By manipulating the permissions field of the file, the owner (or someone with the effective user-ID of the owner) can grant read, write, or execute privileges to other group members.

Users may belong to more than one group. Normally a user is a member of only one group at a time, but may change groups using *newgrp(1)*. *Optionally, a user can choose to belong to multiple groups simultaneously by invoking the multgrps(1) command. In this case, files created by the user will have their group-ID set to the primary group. The user will have access permissions to any file whose group-ID matches any of the groups the user is in. To determine which groups one is a member of, use id(1).*

Information about groups is kept in the /etc/group file. A sample entry from this file is shown and explained below:

```
prog::123:tammyb,bugeye
```

Each entry is one line; each line has the following fields:

group name	The group name can be any length, though some commands will truncate the name to 8 characters. The first character must be alphabetic.
password	The password field should not be used.
group id	The group id is a number from 0 to 60,000. The number must not include a comma. Numbers below 100 are reserved.
login names	The login names of group members are in a comma-separated list. Currently, this list cannot exceed 100 login names.

5.2 The User's Environment

For C shell users, the key element in establishing an environment in which users can successfully communicate with the computer are the files *.login* and *.cshrc*. When a new C shell user is create, the system copies the files */etc/stdcshrc* and */etc/stdlogin* into the respective files *.cshrc* and *.login* in the user's home directory. These files should contain standard or suggested environments that users can customize to suit their individual needs.

For Bourne shell users, a single file, the *.profile*, replaces the *.login* and *.cshrc* files. These three files are referred to as "startup files" in this chapter.

Three startup files are executed for C shell users, in the following order:

1. */etc/cshrc*

   This is an ASCII text file that contains commands and shell procedures, and sets environment variables that are appropriate for all users on the system. Whenever a user logs in, the *login* process executes this file.

   A sample */etc/cshrc* is shown below:

```
# default settings for all users
#       loaded <<before>> $HOME/.cshrc

umask 022

if ($?prompt) cat /etc/motd
set mail=$MAIL
if ( { /bin/mail -e } ) then
        echo 'You have mail.'
endif
```

   In this example, several useful commands are executed. The message of the day is displayed if a prompt exists. The location of the user's mail file is set, and finally, a message informs the user if he or she has mail.

2. The individual user's *.cshrc*.

   This is an executable commands file, *.cshrc*, that resides in a user's home directory. The *.cshrc* can contain additional commands and variables that further customize a user's environment.

A sample */etc/cshrc* is shown below:

```
set history = 20

setenv  VISUAL          /usr/bin/vi
setenv  EDITOR          /usr/bin/emacs

set filec
setenv ROOT /
setenv SHELL csh
```

In this example, four environment variables are set, file completion is turned on, and the length of the history of recently issued commands is set to 20.

3. The individual user's login.

   This is an executable commands file, *.login*, that resides in the user's home directory. The *.login* further customizes the environment.

   A sample *.login* is shown below:

```
umask 022

stty line 1 erase '^H' kill '^U' intr '^C' echoe
eval `tset -s -Q`

set path = (.  /bin /usr/bsd /bin /usr/bin /usr/sbin \
/usr/bin/X11 /usr/demos /usr/local/bin)

set prompt = "`hostname` % "
```

   In this example, the file creation mask is set to 022. Then some useful key bindings are set, using the command *stty*(1). The user's terminal is further initialized with *tset*(1). The default path is expanded to include the user's own binary directory, and others. Then the prompt is customized.

   For information on the shell programming commands used in these examples, see *csh*(1) in the *IRIS-4D User's Reference Manual.*

5.2.1 Environment Variables

An array of strings called the environment is made available by *exec*(2) when a process begins. Since *login* is a process, the array of environment strings is made available to it. An example of a typical array of strings for the C shell is shown here:

```
LOGNAME=bugeye
PWD=/usr/people/bugeye
HOME=/usr/people/bugeye
PATH=.:/usr/people/bugeye/bin:/usr/bsd:/bin:/etc:/usr/sbin:
/usr/bin: /usr/local/bin:/usr/lib/news:/usr/etc:
/usr/lbin:usr/NeWS
SHELL=/bin/csh
MAIL=/usr/mail/bugeye
TERM=iris-ansi
PAGER=more
TZ=EST5EDT
EDITOR=emacs
DISPLAY=myhost:0
VISUAL=vi
```

The environment variables shown in the above example give values to 13 names for user *bugeye*. Other programs make use of the information. For example, the user's terminal is defined as an *iris-ansi* (TERM=iris-ansi). When the user invokes the default visual editor *vi*(1), *vi* checks this environment variable, then looks up the characteristics of an *iris-ansi* terminal. New strings can be defined at any time. By convention they are defined with the variable in uppercase, followed by a space, followed by the value. The individual *.login* and *.cshrc* files can set whichever variables the user wants.

5.2.2 umask

A system default controls the permissions mode of any files or directories created by a user. The workstation has default values of 644 for files and 755 for directories. That means that for files everyone automatically gets read and write permission, but not execute permission. For directories, everyone gets read, write, and execute permission. (Execute permission on a directory means the ability to **cd** to the directory and to copy files from it.)

Users frequently set up a user mask in their startup file by means of the *umask*(1) command. *umask* alters the default permission levels by a specified amount.

For example,

```
umask 027
```

leaves the permission level for owner unchanged, lowers the permission level for group by 2, and reduces the permissions for others to zero. The system default was 666; this user mask changes it to 640, which translates into read and write permission for the owner, read permission for the group, and no permission for others.

5.2.3 Default Shell and Restricted Shell

Generally, when a user logs in the default program that is started is */bin/csh*. There may be cases, however, where a user needs to be given a restricted shell, */bin/rsh*.

A restricted shell is one where the user is not allowed to

- change directories
- change the value of $PATH
- specify pathnames or command names containing a slash (/). That is, the user of a restricted shell may not access files or directories other than the present working directory or those included in $PATH
- redirect output

The restrictions are enforced after *profile* has been executed.

The administrator can use a restricted shell strategy to limit certain users to the execution of a small number of commands or programs. By setting up a special directory for executables (*/usr/rbin*, for example), and controlling PATH so it only references that directory, the administrator can restrict the user's activity in whatever way is appropriate.

Note: Two shells invoked as "rsh" are shipped with IRIX. The other shell is the Berkeley remote shell, in */usr/bsd/rsh*. Be careful not to confuse the two.

5.3 User Communications Services

There are several ways to communicate with and among users in the IRIX system. Some of the most frequently used are described in this section.

5.3.1 Message of the Day

Items of broad interest that you want to make available to all users can be put in the */etc/motd* file. The contents of */etc/motd* are displayed on the user's terminal as part of the login process. The login process executes a file called */etc/cshrc* (for the C shell), which is an executable shell script that, among other things, commonly contains the command:

cat /etc/motd

Any text contained in */etc/motd* is displayed for each user each time the user logs in. For this information to have any impact on users, you must take pains to use it sparingly and to clean out outdated announcements. A typical use for the Message of the Day facility might be

```
5/30:  The system will be unavailable from 6-11pm Thursday, 5/30
       for preventive maintenance.
```

Part of the preventive maintenance should be to remove the notice from */etc/motd*.

5.3.2 news

Another electronic bulletin board facility is the */usr/news* directory and the *news*(1) command. The directory is used to store announcements in text files, the names of which are usually used to provide a clue to the content of the news item. The *news* command is used to print the items on your terminal.

The */etc/cshrc* file is also used to inform users about news items. A typical */etc/cshrc* contains the line

news –n

The **–n** argument causes the names of files in the */usr/news* directory to be printed on a user's terminal as the user logs in. Item names are displayed only for current items, that is, items added to the */usr/news* directory since the user last looked at the news. The idea of currency is implemented like this: when you read a news item an empty file named *.news_time* is written in your login directory. As with any other file, *.news_time* carries a time stamp indicating the date and time the file was created. When you log in, a comparison is made between the time stamp of your *.news_time* file and time stamp of items in */usr/news*.

Unlike the Message of the Day where users have no ability to turn the message off, with *news* users have a choice of several possible actions:

read everything	If the user enters the command, *news* with no arguments, all news items posted since the last time the user typed in the command are printed on the user's terminal.
select some items	If the *news* command is entered with the names of one or more items as arguments, only those items selected are printed.
read and delete	After the *news* command has been entered, the user can stop any item from printing by pressing the delete key. Pressing the delete key twice in a row stops the program.
ignore everything	If the user is too busy to read announcements at the moment, they can safely be ignored. Items remain in */usr/news* until removed. The item names will continue to be displayed each time the user logs in.

flush all items

If the user simply wants to eliminate the display of item names without looking at the items, a couple of techniques will work:

```
touch .news_time
```

updates the time-accessed and time-modified fields of the *.news_time* file thus faking out the currency mechanism of *news*.

```
news > /dev/null
```

prints the news items on the null device.

5.3.3 *write* to All Users

The ability to write to all logged-in users, via the **wall**(1M) command, is an extension of the *write*(1) command. It is fully effective only when used by the super-user. While *wall* is a useful device for getting urgent information out quickly, users tend to find it annoying to have messages print out on their terminal right in the middle of whatever else is going on. The effect is not destructive, but is somewhat irritating. Many users guard against this distraction by including the command

```
mesg n
```

in their *.cshrc*. This blocks other ordinary users from interjecting a message into your *stdout*. The *wall* command, when used by the super-user, overrides the *mesg n* command. It is best to reserve this for those times when you as the system administrator need to ask users to get off the system.

5.3.4 Mail

The IRIX system offers electronic mail utilities through which users can communicate among themselves. If your system is connected to others by networking facilities, *Mail*(1), the BSD mailer, can be used to communicate with persons on other systems. See also *mail*(1).

5.4 Anticipating User Requests

As the system administrator for your workstation you can expect users to look to you to help solve any number of problems. In addition to the system log described in Chapter 6, Processor Operations, you will find it helpful to keep a user trouble log. The problems that users run into fall into patterns. If you keep a record of how problems were resolved, you will not have to start from scratch when a problem recurs. Another technique that is strongly recommended is an organized way for users to report problems. For example, you might wish to set up a "trouble" mail alias, so that users with problems can simply send mail to "trouble" for assistance.

6. Processor Operations

This chapter deals with the day-to-day operations of your IRIS-4D Series workstation or server. It covers:

- Guidelines for balancing the needs of system maintenance and the interests of your user community; suggestions for record keeping; lists of important administrative directories and files.
- How to turn on and off the computer and how to change operating modes
- The definition of the operating levels of the system; how they are controlled
- How to operate system accounting

6.1 General Operating Policy

Many administrative tasks require the system to be shut down to a run level other than the multi-user state (see the discussion on Operating Levels below). This means that conventional users cannot access the system. When the machine is taken out of the multi-user state, the users on the machine at the time are requested to log off. You should do these types of tasks when they will interfere the least with the activities of the user community.

Sometimes situations arise that require the system to be taken down with little or no notice provided to the users. Try to provide the user community as much notice as possible about events affecting the use of the machine. When the system must be taken out of service, also tell the users when to expect the system to be available. Use the Message of the Day (*/etc/motd*) to keep users informed about changes in hardware, software, policies, and procedures.

At your discretion, the following items should be done as prerequisites for any task that requires the system to leave the multi-user state.

1. When possible, schedule service-affecting tasks to be done during periods of low system use. For scheduled actions, use the Message of the Day (*/etc/motd*) to inform users of future actions.
2. Check to see who is logged in before taking any actions that would affect a logged-in user. The */etc/whodo* and */bin/who* commands can be used to see who is on the system. You may also wish to check for large background tasks, such as background compilations, by executing *ps -ef*.
3. If the system is in use, provide the users advanced warning about changes in system states or pending maintenance actions. For immediate actions, use the */etc/wall* command to send a broadcast message announcing that the system will be taken down at a given time. Give the users a reasonable amount of time to terminate their activities and log off before taking the system down.

6.2 Maintaining a System Log

In a multi-user environment it is strongly recommended that a complete set of records be maintained. A system log book can be a valuable tool when trouble shooting transient problems or when trying to establish system operating characteristics over a period of time. Some of the things that you should consider entering into the log book are:

- Maintenance records (dates and actions)
- Printouts of error messages and diagnostic phases
- Equipment and system configuration changes (dates and actions)
- Copies of important configuration files

The format of the system log and the types of items noted in the log should follow a logical structure. Think of the log as a diary that you update on a periodic basis. To a large measure, how you use your system will dictate the form and importance of maintaining a system log.

6.3 Administrative Directories and Files

This section briefly describes the directories and files that are frequently used by a system administrator. For additional information on the formats of the system files, refer to Section 4 of the manual pages in the *IRIS-4D Programmer's Reference Manual.*

6.3.1 Root Directories

The main directories of the *root* file system (/) are as follows.

bin	Directory that contains public commands.
dev	Directory containing special files that define all of the devices on the system.
etc	Directory that contains administrative programs and tables.
lib	Directory that contains public libraries.
lost+found	Directory used by *fsck*(1M) to save disconnected files and directories.
tmp	Directory used for temporary files.
usr	Directory used to mount the */usr* file system.

6.3.2 Important System Files

The following files and directories are important in the administration of your system.

/etc/config	Directory containing boot and runtime configuration information.
/etc/cshrc	File containing the standard (default) environment for */bin/csh* users.
/etc/exports	File containing the list of NFS filesystems currently exported to NFS clients.

/etc/fstab	File used to specify the file system(s) to be mounted by */etc/mountall*.
/etc/gettydefs	File containing information used by */etc/getty* to set the speed and terminal settings for a line.
/etc/group	File describing each group to the system.
/etc/hosts	File containing information about the known hosts on the network.
/etc/hosts.equiv	File containing a list of hosts trusted for *rlogin* and *rsh* execution.
/etc/init.d	Directory containing executable files used in upward and downward transitions to all system run levels. These files are linked to files beginning with **S** (start) or **K** (stop) in */etc/rcn.d*, where *n* is replaced by the appropriate run level.
/etc/inittab	File containing the instructions to define the processes created or terminated by */etc/init* for each initialization state.
/etc/lvtab	File containing information describing the logical volumes used by the machine. This file is read by the logical volumes utilities.
/etc/motd	File containing a brief Message of the Day.
/etc/passwd	File identifying each user to the system.
/etc/profile	File containing the standard (default) environment for */bin/sh* users.
/etc/rc0	File executed by */etc/shutdown* that executes shell scripts in */etc/rc0.d* and */etc/shutdown.d* directories for transitions to system run-level 0.

/etc/rc0.d	Directory containing files executed by */etc/rc0* for transitions to system run-level 0. Files in this directory are linked from files in the */etc/init.d* directory and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and **S** indicates processes that are started when entering run-level 0.
/etc/rc2	File executed by */etc/init* that executes shell scripts in */etc/rc2.d* and */etc/rc.d* on transitions to system run-level 2.
/etc/rc2.d	Directory containing files executed by */etc/rc2* for transitions to system run-levels 2 and 3. Files in this directory are linked from files in the */etc/init.d* directory and begin with either a **K** or an **S**. **K** indicates processes that should be stopped and **S** indicates processes that should be started when entering run-levels 2 or 3.
/etc/shutdown	File containing a shell script that gracefully shuts down the system in preparation for system backup or for scheduled downtime.
/etc/sys_id	File containing the system name.
/etc/ttytype	File containing a list, ordered by terminal port, of what kind of terminal is likely to login to that port.
/etc/TIMEZONE	File used to set the time zone shell variable TZ.
/etc/utmp	File containing the information on the current run-state of the system.
/etc/wtmp	File containing a history of system logins.
/etc/xwtmp	File containing an extended history of system logins.
/usr/adm/acct	Directory containing information collected by the accounting option.

/usr/adm/crash	Directory where crash dumps of the system are stored. These can safely be removed (and are often quite large). See *savecore*(1).
/usr/adm/sa	Directory containing information collected by *sar*(1).
/usr/adm/sulog	File containing a history of *su* command usage. This file should be checked periodically for size.
/usr/adm/SYSLOG	File containing all system and daemon error messages.
/usr/etc/yp/ypdomain	File containing the domain name if the machine is using yellow pages.
/usr/lib/cron/log	File containing a history of all the actions taken by */etc/cron*. This file should be checked periodically for size.
/usr/lib/cron/cron.allow	File containing a list of users allowed to use *crontab*(1).
/usr/lib/cron/cron.deny	File containing a list of users denied access to *crontab*(1). Checked if */usr/lib/cron/cron.allow* does not exist.
/usr/spool/news	Directory containing news files. This directory should be checked periodically, and old files should be discarded.
/usr/spool/cron/crontabs	Directory containing crontab files for the *adm*, *root*, and *sys* logins and ordinary users listed in *cron.allow*.
/usr/sysgen/master.d	Directory containing files that define the configuration of hardware devices, software drivers, system parameters, and aliases.

Appendix A, ''Device Files'', contains further information on the device files and directories that reside in the */dev* directory.

6.4 Administering Operation

This section covers these procedures:

- Turning on the computer
- Turning off the computer
- Bringing the system into single-user mode
- Returning to multi-user mode
- Halting and rebooting the system

6.4.1 Turning On the Workstation or Server

To turn on power to the system, follow these steps:

1. Set the **Power** switch on the display monitor to **On**.
2. Set the **Power** switch on the status panel at the base of the drive tower to **On** (see Figure 2-3). If the power for the computer is already on, press the **Reset** button, which is located below the **Power** switch.

   The system runs power-on diagnostics. The messages that they produce appear on the screen of the *diagnostics terminal*, which is an ASCII terminal connected to the port labeled **1** (see Section 7.1, Attaching an ASCII Terminal), and the console (see Section 5.4.3, The PROM Monitor Environment).

   If the computer displays the PROM menu, enter **1** to start the system. Go to Step 4.

   If the computer displays the PROM monitor prompt:

   ```
   >>
   ```

   Wait at least 20 seconds for the disk to spin up to speed, then enter the *auto* command and press <enter>.

   ```
   auto
   ```

You see a display similar to this:

```
Starting System...

IRIX System V Release 4D1-3.3 IP5 Version 09261636
Total real memory   : 12582912
Available memory    : 11530240

root on dev 0x1620, swap on dev 0x1621
Available memory    = 11452416

The system is coming up.  Please wait.
```

If the system determines that the file systems need checking, it checks them with the *fsck* program. *fsck* repairs any damage it finds before the system mounts the file systems. The system comes up in multi-user mode (see Section 4.4, Administering Machine Operation).

A login prompt appears:

```
IRIS console login:
```

3. Log in to the system. You can log in with a system or user login.

```
IRIS console login: root
IRIX System V Release 4D1-3.3 IRIS
Copyright (c) 1990 Silicon Graphics, Inc.
All Rights Reserved
```

After you log in to the IRIS, it takes a few seconds for 4Sight to start. Then the console window appears in the lower left corner of the screen.

The system is now running IRIX, Silicon Graphics' version of AT&T's UNIX System V Release 3, and 4Sight, the integrated window system on the IRIS. To learn more about 4Sight, see the *4Sight User's Guide* and your owner's guide.

Note: It is possible to turn off the 4Sight environment if you wish. After typing your login ID, add the environment variable *NOGRAPHICS* on the same line.

4. When you want to log out, move the mouse to a background portion of the screen and press the right button. Select *logout*.

6.4.2 Turning Off the Workstation or Server

This section describes how to turn off the computer from multi-user or single-user mode.

Turning Off from Multi-User Mode

To turn off the computer when the system is in multi-user mode, issue the *sysadm powerdown* command at the system prompt. This command causes the system to flush the system buffers, close open files, stop all user processes and *daemons*, unmount file systems, and bring the system to the PROM monitor. A daemon is a background process that performs a system-wide function, for example, *lpsched*(1M). Follow these steps:

1. Check to see if any users are logged in by issuing the *sysadm whoson* command at the system prompt. You see a report similar to this one:

```
These users are currently logged in:

   ID    terminal number   sign-on time
------- --------------- --------------
root        console       Apr  4 12:42
bugeye      ttyq1         Apr  4 10:26
```

2. Notify users that the system is shutting down. Issue the */etc/wall*(1M) command.

```
Broadcast Message from root (console) Wed Feb 26 07:30:27...
The system will be coming down in 5 minutes.
Please log off.
<ctrl-d>
```

3. Issue the *sysadm powerdown* command, then bring the system to the PROM monitor or System Maintenance menu. Follow this sequence:

```
Running subcommand 'powerdown' from 'machinemgmt',
MACHINE MANAGEMENT

Once started, a powerdown CANNOT BE STOPPED.
Do you want to start an express powerdown? [y, n, ?, q] n
Enter the number of seconds to allow
between the warning messages (default 60): [?, q]30

Shutdown started.    Fri Aug 28 17:10:57...

Broadcast Message from root (console) Fri Aug 28 17:10:59
THE SYSTEM IS BEING SHUT DOWN!
Log off now.

INIT: New run level: 0
The system is coming down.  Please wait.
(System services are stopped.)
```

The PROM monitor prompt or System Maintenance menu appears.

Caution: Wait for the PROM monitor prompt or System Maintenance menu to appear before turning off power to the computer.

5. Turn off the power switch located on the status panel on the front of the drive tower.

Turning Off from Single-User Mode

If the system is in single-user mode, follow these steps:

1. Use the *shutdown* command to turn off the system and guarantee file system integrity.

```
shutdown -y -i0 -g0
```

The arguments have these meanings:

-y assume yes answers to all questions
-i0 go to state 0 (PROM monitor)
-g0 allow grace period of 0 seconds

As *root*, type **shutdown –y –i0 –g0**. You see a display similar to this one:

```
Shutdown started.      Fri Aug 28 17:11:50 EDT 1987

INIT: New run level: 0
The system is coming down. Please wait.
```

The system stops all services and the PROM monitor prompt or System Maintenance menu appears.

Caution: Wait for the PROM monitor prompt or System Maintenance menu to appear before turning off power to the computer.

2. Turn off the power switch located on the status panel on the front of the drive tower.

6.4.3 Going to Single-User Mode

When you bring the system to single-user mode, only the console can access the system. For this reason, it is a good policy to perform the tasks listed above during off hours when the system is not in great demand.

To bring the system to single-user mode, follow these steps:

1. Log in as *root* at the console, or execute *su*.

2. Check on system activity, with *who*, *whodo*, and *ps -ef*.

3. There are three commands that instruct the system to go into single user mode. One is *single*, which switches the system to single-user mode and turns the *gettys* off.

   Another is:

   ```
   init 0
   ```

   See section 6.5, "Operating Levels", for a further discussion of this command.

   The third method, *shutdown*, is the most forgiving:

   ```
   shutdown -i1 -g0
   ```

   By default, *shutdown* sends broadcast messages and asks:

   ```
   Do you want to continue? (y or n)
   ```

   If you decide that this is not a good time to put the system in single-user mode, type

   ```
   n
   ```

   The system broadcasts the messages:

   ```
   False alarm: The system will not be brought down.

   Shut down aborted.
   ```

   If you decide to continue with the shutdown, type

   ```
   y
   ```

   This continues with the shutdown and brings the system to single-user mode. Once you are in single-user mode, you see these messages:

   ```
   INIT: New run level: S

   INIT: SINGLE USER MODE
   #
   ```

   When you see the system prompt (#), you can begin administrative tasks.

6.4.4 Returning to Multi-User Mode

This section tells how to return the system to multi-user mode from single-user mode. After you finish administrative tasks, bring the system back to multi-user mode with either the *init* or the *multi* command.

```
init 2
```

```
multi
```

multi is a shell script that causes *init* to inspect */etc/inittab* and to execute entries that initialize the system to multi-user mode. You see this message:

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

The computer checks the file systems, if necessary, and prints the current system configuration. At the end of this sequence, you see this login prompt:

```
IRIS console login:
```

The system is now in multi-user mode, ready for you to log in with a system or user login.

6.4.5 Halting and Rebooting the Operating System

This procedure allows you to halt the system and reboot from the disk. Its main purpose is to force a reconfiguration of the operating system after hardware and/or software modifications.

To halt and reboot the system from the hard disk, issue either the *reboot*, *sysadm reboot*, *init 6*, or *shutdown –i6* command:

```
SYSTEM ADMINISTRATION

1 filemgmt  file management menu
2 machinemgmt  machine management menu
3 syssetup  system setup menu
4 ttymgmt  tty management
5 usermgmt  user management menu
```

```
Enter a number, a name, the initial part of a name,
or ? or <number>? for HELP, q to quit: 2

MACHINE MANAGEMENT

1 powerdown  stop all running programs go to the PROM monitor
2 reboot  stop all running programs then reboot the machine
3 whoson  print a list of who is on the machine

Enter a number, a name, the initial part of a name,
or ? or <number>? for HELP, q to quit: 2

Once started, a reboot CANNOT BE STOPPED.
Do you want to start an express reboot? [y, n, ?, q] y
Shutdown started.     Thu May 17 17:45:07 EDT 1988
Broadcast Message from root (console) Thu May 17 17:45:09...
THE SYSTEM IS BEING SHUT DOWN! Log off now.

INIT: New run level: 6
The system is coming down.  Please wait.
(System services are stopped.)
The system is being restarted.
```

The system is completely booted when you see the login prompt.

6.5 Operating Levels

6.5.1 General

After you have set up your computer for the first time (plugging it in, hooking all the hardware together, booting it, running the setup programs,) as documented in the *IRIS-4D Series Owner's Guide*, you and other users can use the system. When you bring the system to multiuser mode, the following things happen:

- The file systems are mounted.
- The *cron* daemon is started for scheduled tasks.
- Network services are started, if turned on.
- The basic networking functions of *uucp* are available for use.
- The spooling and scheduling functions of the LP package (if added to the system) are available for use.
- Users can log in. The *getty*s are spawned on all connected terminal lines listed in */etc/inittab* to have *getty*s respawned. (*gettys* are not on when the system is installed. You have to turn them on yourself.)

This is defined as the multi-user state. It is also referred to as *init* state "2" because all of the activities of initializing the system are under the control of the *init* process. The "2" refers to entries in the special table */etc/inittab* used by *init* to initialize the system to the multi-user state.

Not all activities, however, can be performed in the multi-user state. For example, if you were able to unmount a file system while users were accessing it, you would cause a lot of data to be lost. Hence, for unmounting and other system administration tasks, there is a need for another state, the single-user state.

The single-user state is an environment in which only the console has access to the system and the root file system alone is mounted. You are free to do tasks that affect the file systems and the system configuration because you are the only one on the system.

There are other system states (see Figure 6-1), but first a note of clarification. One of the more confusing things about the discussion of system states is that there are many terms used to identify the same thing: the particular operating level of the system.

Here is a list of frequently encountered synonyms:

- run state
  run level
  run mode
- init state
  system state

Likewise, each system state may be referred to in a number of ways, for example:

- single-user
- single-user mode
- run level 1, and so on

In any case, each state or run level clearly defines the operation of the computer. Figure 6-1 defines each of them.

Run Level	Description
0	Power-down state.
1, s, or S	Single-user mode is used to install/remove software utilities, run file system backups/restores, and to check file systems. Both states unmount everything except root and kill all user processes, except those that relate to the console.
2	Multi-user mode is the normal operating mode for the system. The default is that the *root* (/) and user (*/usr*) file systems are mounted in this mode. When the system is powered up it is put in multi-user mode.
6	Reboot mode is used to bring the system down and the back up again. This mode is useful when you are changing certain system configuration parameters.

Figure 6-1. System States

6.5.2 How *init* Controls the System State

IRIX systems always run in one state or another. The actions that cause the various states to exist are under the control of the *init* process, which is the first general process created by the system at boot time. It reads the file */etc/inittab*, which defines exactly which processes exist for which run level.

In the case of the multi-user state (run level 2), *init* scans the file for entries that have a tag for the run level (the tag is a 2) and executes everything after the last colon (:) on the line containing the tag.

If you look at your */etc/inittab*, you'll see something that looks like the following. (It is most unlikely that yours will look exactly like this one; */etc/inittab* changes from one configuration to another.)

```
is:2:initdefault:
fs::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
mt::sysinit:/etc/brc </dev/console >/dev/console 2>&1
cons::sysinit:/etc/gl/setupcons </dev/console >/dev/console 2>&1
link::wait:/etc/lnsyscon > /dev/console 2>&1 < /dev/null
s0:06:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/console 2>&1 </dev/console
s2:23:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
s3:3:wait:/etc/rc3 >/dev/console 2>&1 </dev/console
```

Note: If */etc/inittab* was removed by mistake and is missing during shutdown, *init* will enter the single user state (*init s*). While entering single user state, */usr* will remain mounted and processes not spawned by *init* will continue to run. You should replace */etc/inittab* before changing states again.

The format of each line is

id:level:action:process

- *id* is one or two characters that uniquely identify an entry.
- *level* is zero or more numbers and letters (**0** through **6**, **s**, **a**, **b**, and **c**) that determines what *level*(s) *action* is to take place in. If *level* is null, the *action* is valid in all levels.
- *action* can be one of the following:

sysinit	run *process* before *init* sends anything to the system console (`Console Login:`).
bootwait	start *process* the first time *init* goes from single-user to multi-user state after the system is booted. (If *initdefault* is set to **2**, the process will run right after the boot.) *init* starts the process, waits for its termination and, when it dies, does not restart the process.
wait	when going to *level*, start *process* and wait until it's finished.
initdefault	when *init* starts, it will enter *level*; the *process* field for this *action* has no meaning.
once	run *process* once and don't start it again if it finishes.

powerfail	tells *init* to run *process* whenever a direct powerdown of the computer is requested.
respawn	if process does not exist, start it, wait for it to finish, and then start another.
ondemand	synonymous with *respawn*, but used only with *level* **a**, **b**, or **c**.
off	when in *level*, kill process or ignore it.

- *process* is any executable program, including shell procedures.
- # can be used to add a comment to the end of a line. *Init* will ignore all lines not in the proper format. Note that # is not a comment character when used at the beginning of a line.

When changing levels, *init* kills all processes not specified for that level. We'll go through more manageable pieces of this table below to get a clearer idea of how the system is controlled by *init*.

6.5.3 Entering the Multi-User State

Powering Up

When you power up your system, it enters multi-user state by default. (You can change the default by modifying the *initdefault* line in your *inittab* file.) In effect, going to the multi-user state follows these broad lines (see Figure 6-1):

1. The operating system is loaded and the early system initializations are started by *init*.
2. The run level change is prepared by the */etc/rc2* procedure.
3. Finally the system is made public via the spawning of *gettys* along the enabled terminal lines.

Early Initialization

Just after the operating system is first loaded into core via the specialized boot programs the *init* process is created. It immediately scans */etc/inittab* for entries of the type *sysinit*:

```
fs::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
mt::sysinit:/etc/brc </dev/console >/dev/console 2>&1
cons::sysinit:/etc/gl/setupcons </dev/console >/dev/console 2>&1
```

They are executed in sequence and perform the necessary early initializations of the system. Note that each entry indicates a standard input/output relationship with */dev/console*. This is the way communication is established with the system console before the system has been brought to the multi-user state.

Preparing the Run Level Change

Now the system must be placed in a particular run level. First, *init* scans the table to find an entry that specifies an *action* of the type *initdefault*. If it finds one, it uses the run level of that entry as the tag it will use to select the next entries to be executed. In our sample */etc/inittab*, the *initdefault* entry specifies run level 2 (the multi-user state) as the level to select and execute other entries:

```
is:2:initdefault:
s2:23:wait:/etc/rc2 >/dev/console 2>&1 </dev/console
co:23:respawn:/etc/gl/conslog
t1:23:respawn:/etc/getty -s console ttyd1 co_9600 # alt console
t2:23:off:/etc/getty ttyd2 co_9600              # port 2
t3:23:off:/etc/getty ttyd3 co_9600              # port 3
t4:23:off:/etc/getty ttyd4 co_9600              # port 4
```

The other entries shown above specify the actions necessary to prepare the system to change to the multi-user run level. First, */etc/rc2* is executed. It executes all files in */etc/rc2.d* that begin with the letter S, accomplishing (among other things) the following:

- Sets up and mounts the file systems
- Starts the *cron* daemon
- Makes *uucp* available for use
- Makes line printer (lp) system available for use, if installed
- Starts accounting, if installed
- Starts networking, if installed
- Starts the mail daemon
- Starts the system monitor
- Indicates connected ports (ttys)

init then starts a *getty* for the Console, and starts *getty* on the lines connected to the ports indicated.

At this moment, the full multi-user environment is established, and your system is available for users to log in.

6.5.4 A Look at the System Life Cycle

Changing Run Levels

In effect, changing run levels follows these broad lines:

1. The system administrator enters a command that directs *init* to execute entries in */etc/inittab* for a new run level, such as *multi*, *single*, or *reboot*.
2. Key procedures, such as */etc/shutdown*, */etc/rc0*, and */etc/rc2* are run to initialize the new state.
3. The new state is reached. If it is state 1 the system administrator can proceed.

Run Level Directories

Run levels 0 and 2 each have a directory of files that are executed in transitions to and from that level. These directories are *rc0.d*, *rc2.d*, and *rc3.d*, respectively. All files in these directories are linked to files in */etc/init.d*. The run-level filenames look like this:

S*00name*

or

K*00name*

The filenames can be split into three parts:

S or **K**	The first letter defines whether the process should be started (**S**) or stopped (**K**) upon entering the new run level.
00	The next two characters are a number from 00 to 99. They indicate the order in which the files will be started (S00, S01, S02, etc.) or stopped (K00, K01, K02, etc).
name	The rest of the filename is the */etc/init.d* filename this file is linked to.

For example, the *init.d* file *cron* is linked to the *rc2.d* file and *rc0.d* file **K70cron**. When you enter *init 2*, this file is executed with the **start** option: **sh S75cron start**. When you enter *init 0*, this file is executed with the **stop** option: **sh K70cron stop**. This particular shell script will execute */usr/bin/cron* when run with the **start** option and *kill* the *cron* process when run with the **stop** option.

Because these files are shell scripts, you can read them to see what they do. You can modify the files, though it is preferable to add your own since the delivered scripts may change in future releases. To create your own scripts you should follow these rules:

- Place the file in */etc/init.d*.
- Link the file to files in appropriate run state directories using the naming convention described above.
- Have the file accept the **start** and/or **stop** options.

Going to Single-User Mode

At times in a given work week, you will need to perform some administrative functions in the single-user mode, such as backing up the hard disk onto tape. The normal way to go to single-user mode is through the */etc/shutdown* command. This procedure executes all the files in */etc/rc0.d* and */etc/shutdown.d* directories by calling the */etc/rc0* procedure, accomplishing, among other things, the following:

- Closing all open files and stopping all user processes
- Stopping all daemons and services
- Writing all system buffers out to the disk
- Unmounting all file systems except root

The entries for single-user processing in the sample */etc/inittab* are:

```
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/console 2>&1 </dev/console
```

There are three major ways to start the shutdown processing.

1. You can enter the **shutdown –i1** command (recommended). The option **–g** specifies a grace period between the first warning message and the final message.
2. You can enter the *single* command, which runs a shell script that switches to single-user mode and turns the *gettys* off. You can enter the *init 1* command, which forces the *init* process to scan the table. The first entry it finds is the **s1** entry, and it starts the shutdown processing.

Now the system is in the single-user environment, and you can perform the appropriate administrative tasks.

Turning the System Off

The final step in the life cycle of the system is turning it off. The following entries apply to powering the system down:

```
s0:06s:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
of:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
```

One way to turn the system off is to cause a powerfail signal to be sent to the system. You can either enter the *powerdown* command, the *init 0* command, or directly invoke the */etc/shutdown –i0* command.

In either case, the */etc/shutdown* and */etc/rc0* procedures are called to clean up and stop all user processes, daemons, and other services and to unmount the file systems. Finally, the */etc/uadmin* procedure is called, which indicates that the last step—physically removing power from the system—is under firmware control.

6.6 System Accounting

IRIX includes utilities that, when activated, log certain types of system activity. This is called process accounting. Process accounting allows a system administrator to keep a watchful eye on system resources such as:

- The number of programs a user runs
- The size and duration of user programs
- Data throughput (I/O)

As an administrator, you are responsible for the smooth operation of an IRIX system. If something affects the continuity or performance of that system, generally it is your job to figure out what happened and prevent it from reoccuring.

Process accounting is a tool for collecting information on system activity. Once you have this information, you can interpret it to gain understanding about how system resources are being used. For example, if the performance of your system is unusually poor and you are running process accounting, you can determine whether someone is using more than their share of CPU time.

Significant system events, such as security breaches, are much easier to trace and understand if process accounting is turned on. From accounting logs, you can construct a "moment", or sequence of events, which helps you to determine what happened to or on your system. If, for example, an unauthorized person logs into your system and you are running process acccounting, you can determine, for the most part, what they did, how long they did it, and whether it was important. The next sections describe how to turn process accounting on and off and how to look at the log files, which are created by process accounting.

6.6.1 General

The following list is a synopsis of the actions of the accounting system.

- At process termination, the IRIX kernel writes one record per process in */usr/adm/pacct* in the form of *acct.h*
- The *login* and *init* programs record connect sessions by writing records into */etc/wtmp*. Date changes, reboots, and shutdowns (using *acctwtmp* are also recorded in this file.
- The disk utilization program *acctdusg* and *diskusg* break down disk usage by login.
- Each day, *runacct* shell procedure is executed by *cron* to reduce accounting data and produce summary files and reports.
- The *monacct* procedure, which saves and restarts summary files, can be executed on a regular basis. This procedure generates a report and cleans the *sum* directory.

6.6.2 Accounting Files and Directories

The directory */usr/lib/acct* contains the programs and shell scripts necessary to run the accounting system. Process accounting uses a login (*/usr/adm*) to perform certain tasks. */usr/adm* contains active data collection files used by the process accounting. See "Process Accounting Files" at the end of this section. Here is a description of the primary subdirectories in */usr/adm*:

/usr/adm/acct/nite	contains files that are reused daily by *runacct*.
/usr/adm/acct/sum	contains the cumulative summary files updated by *runacct*.
/usr/adm/acct/fiscal	contains periodic summary files created by *monacct*.

6.6.3 Daily Operation

When IRIX enters multiuser mode, */usr/lib/acct/startup* is executed. This is what happens:

1. The *acctwtmp* program adds a "boot" record to */etc/wtmp*. This record is signified by using the system name as the *login* name in the *wtmp* record.
2. Process accounting is started by *turnacct* which, in turn, executes *accton /usr/adm/pacct*.
3. *Remove* is executed to clean up the saved *pacct* and *wtmp* files left in the *sum* directory by *runacct*.

The ckpacct procedure is run via *cron* every hour of the day to check the size of */usr/adm/pacct*. If the file grows past 1000 blocks (default), turnacct switch is executed. The advantage of having several smaller pacct files becomes apparent when trying to restart *runacct* after a failure processing these records.

The *chargefee* program can be used to bill users for file restores, etc. It adds records to */usr/adm/fee* which are picked up and processed by the next execution of *runacct* and merged into the total accounting records.

runacct is executed via *cron* each night. It processes the active accounting files, */usr/adm/pacct*, */etc/wtmp*, */usr/adm/acct/nite/disktacct*, and */usr/adm/fee*. It produces command summaries and usage summaries by *login*.

When the system is shut down using shutdown, the shutacct shell procedure is executed. It writes a shutdown reason record into /etc/wtmp and turns process accounting off.

After the first reboot each morning, the computer operator should execute */usr/lib/acct/prdaily* to print the previous day's accounting report.

Setting Up the Accounting System

If you chose to install the system accounting option, all the files and command lines for implementation have been set up properly. You may wish to verify that the entries in the system configuration files are correct. In order to automate the operation of the accounting system, you should check that the following have been done:

1. The file */etc/init.d/acct* should contain the following lines (among others):

   ```
   /usr/lib/acct/startup
   /usr/lib/acct/shutacct
   ```

   The first line starts process accounting during the system startup process; the second stops it before the system is brought down.

2. For most installations, the following three entries should be in */usr/spool/cron/crontabs/adm* so that cron will automatically run the daily accounting. These lines should already exist.

   ```
   0 4 * * 1-6 /usr/lib/acct/runacct 2>/usr/adm/acct/nite/fd2log
   5 * * * * /usr/lib/acct/ckpacct
   ```

   The following line should be in */usr/spool/cron/crontabs/root*:

   ```
   0 2 * * 4 /usr/lib/acct/dodisk
   ```

3. To facilitate monthly merging of accounting data, the following entry in */usr/spool/cron/crontabs/adm* allows *monacct* to clean up all daily reports and daily total accounting files and deposit one monthly total report and one monthly total accounting file in the fiscal directory.

   ```
   0 5 1 * * /usr/lib/acct/monacct
   ```

   The above entry takes advantage of the default action of *monacct* that uses the current month's date as the suffix for the file names. Notice that the entry is executed at such a time as to allow *runacct* sufficient time to complete. This will, on the first day of each month, create monthly accounting files with the entire month's data.

4. You may wish to verify that an account exists for *adm*. Also, verify that the PATH shell variable is set in */usr/adm/.profile* to:

```
PATH=/usr/lib/acct:/bin:/usr/bin
```

5. To start up system accounting, simply type **chkconfig acct on.** The next time the system is booted, accounting will start.

6.6.4 Runacct

Runacct is the main daily accounting shell procedure. It is normally initiated via cron during nonprime time hours. Runacct processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by prdaily or for billing purposes. The following files produced by *runacct* are of particular interest:

nite/lineuse	Produced by acctcon, reads the *wtmp* file, and produces usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logoffs to logins exceeds about 3/1, there is a good possibility that the line is failing.
nite/daytacct	This file is the total accounting file for the previous day in tacct.h format.
sum/tacct	This file is the accumulation of each day's nite/daytacct and can be used for billing purposes. It is restarted each month or fiscal period by the *monacct* procedure.
sum/daycms	Produced by the acctcms program. It contains the daily command summary. The ASCII version of this file is nite/daycms.
sum/cms	The accumulation of each day's command summaries. It is restarted by the execution of *monacct*. The ASCII version is nite/cms.
sum/loginlog	Produced by the last login shell procedure. It maintains a record of the last time each *login* was used.

sum/rprtMMDD Each execution of *runacct* saves a copy of the daily report that can be printed by prdaily.

runacct takes care not to damage files in the event of errors. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and terminate processing in such a way that *runacct* can be restarted with minimal intervention. It records its progress by writing descriptive messages into the file active. (Files used by *runacct* are assumed to be in the nite directory unless otherwise noted). All diagnostics output during the execution of *runacct* is written into fd3log. Runacct will complain if the files lock and lockl exist when invoked. The lastdate file contains the month and day *runacct* was last invoked and is used to prevent more than one execution per day. If *runacct* detects an error, a message is written to */dev/console*, mail is sent to root and adm, locks are removed, diagnostic files are saved, and execution is terminated.

In order to allow *runacct* to be restartable, processing is broken down into separate reentrant states. A file is used to remember the last state completed. When each state completes, statefile is updated to reflect the next state. After processing for the state is complete, statefile is read and the next state is processed. When *runacct* reaches the CLEANUP state, it removes the locks and terminates. States are executed as follows:

SETUP The command turnacct switch is executed. The process accounting files, */usr/adm/pacct?*, are moved to */usr/adm/Spacct?.MMDD*. The */etc/wtmp* file is moved to /usr/adm/acct/nite/wtmp.MMDD with the current time added on the end.

WTMPFIX The *wtmp* file in the nite directory is checked for correctness by the wtmpfix program. Some date changes will cause acctcon1 to fail, so wtmpfix attempts to adjust the time stamps in the wtmp file if a date change record appears.

CONNECT1 Connect session records are written to ctmp in the form of ctmp.h. The lineuse file is created, and the reboots file is created showing all of the boot records found in the *wtmp* file.

CONNECT2 Ctmp is converted to ctacct.MMDD which are connect accounting records. (Accounting records are in tacct.h format.)

PROCESS	The acctprc1 and acctprc2 programs are used to convert the process accounting files, */usr/adm/Spacct?.MMDD*, into total accounting records in *ptacct?.MMDD*. The Spacct and ptacct files are correlated by number so that if *runacct* fails the unnecessary reprocessing of Spacct files will not occur. One precaution should be noted; when restarting *runacct* in this state, remove the last ptacct file because it will not be complete.
MERGE	Merge the process accounting records with the connect accounting records to form daytacct.
FEES	Merge in any ASCII tacct records from the file fee into daytacct.
DISK	On the day after the dodisk procedure runs, merge disktacct with daytacct.
MERGETACCT	Merge daytacct with sum/tacct, the cumulative total accounting file. Each day, daytacct is saved in sum/tacctMMDD, so that sum/tacct can be recreated in the event it becomes corrupted or lost.
CMS	Merge in today's command summary with the cumulative command summary file sum/cms. Produce ASCII and internal format command summary files.
USEREXIT	Any installation dependent (local) accounting programs can be included here.
CLEANUP	Clean up temporary files, run prdaily and save its output in sum/rprtMMDD, remove the locks, then exit.

Recovering From a Failure

The *runacct* procedure can fail for a variety of reasons; usually due to a system crash, */usr* running out of space, or a corrupted *wtmp* file. If the activeMMDD file exists, check it first for error messages. If the active file and lock files exist, check fd2log for any mysterious messages. The following are error messages produced by *runacct* and the recommended recovery actions:

ERROR: locks found, run aborted

> The files lock and lock1 were found. These files must be removed before *runacct* can restart.

ERROR: acctg already run fo date: check */usr/adm/acct/nite/lastdate*

> The date in lastdate and today's date are the same. Remove lastdate.

ERROR: turnacct switch returned rc=?

> Check the integrity of turnacct and accton. The accton program must be owned by root and have the setuid bit set.

ERROR: Spacct?.MMDD already exists

> File setups probably already run. Check status of files, then run setups manually.

ERROR: */usr/adm/acct/nite/wtmp.MMDD* already exists, run setup

> manually Self-explanatory.

ERROR: wtmpfix errors see */usr/adm/acct/nite/wtmperror*

> Wtmpfix detected a corrupted *wtmp* file. Use fwtmp to correct the corrupted file.

ERROR: connect acctg failed: check */usr/adm/acct/nite/log*

The acctcon1 program encountered a bad *wtmp* file. Use fwtmp to correct the bad file.

ERROR: Invalid state, check */usr/adm/acct/nite/active*

The file statefile is probably corrupted. Check statefile and read active before restarting.

Restarting Runacct

Runacct called without arguments assumes that this is the first invocation of the day. The argument MMDD is necessary if *runacct* is being restarted and specifies the month and day for which *runacct* will rerun the accounting. The entry point for processing is based on the contents of statefile. To override statefile, include the desired state on the command line.

For example:

To start *runacct*:

```
nohup runacct 2> /usr/adm/acct/nite/fd2log&
```

To restart runacct:

```
nohup runacct 0601 2>  /usr/adm/acct/nite/fd2log&
```

To restart runacct at a specific state:

```
nohup runacct 0601 WTMPFIX 2> /usr/adm/acct/nite/fd2log&
```

6.6.5 Fixing Corrupted Files

Unfortunately, this accounting system is not entirely foolproof. Occasionally, a file will become corrupted or lost. Some of the files can simply be ignored or restored from the file save backup. However, certain files must be fixed in order to maintain the integrity of the accounting system.

Fixing WTMP Errors

The *wtmp* files seem to cause the most problems in the day-to-day operation of the accounting system. When the date is changed and the IRIX system is in multiuser mode, a set of date change records is written into */etc/wtmp*. The wtmpfix program is designed to adjust the time stamps in the *wtmp* records when a date change is encountered. However, some combinations of date changes and reboots will slip through wtmpfix and cause acctcon1 to fail. The following steps show how to patch up a *wtmp* file.

```
cd /usr/adm/acct/nite
fwtmp <wtmp.MMDD>xwtmp
ed xwtmp
delete corrupted records or
delete all records from beginning up to the date change
fwtmp -ic <xwtmp> wtmp.MMDD
```

If the *wtmp* file is beyond repair, create a null *wtmp* file. This will prevent any charging of connect time. Acctprc1 will not be able to determine which *login* owned a particular process, but it will be charged to the *login* that is first in the password file for that user id.

Fixing Tacct Errors

If the installation is using the acocunting system to charge users for system resources, the integrity of sum/tacct is quite important. Occasionally, mysterious tacct records will appear with negative numbers, duplicate user IDs, or a user ID of 65,535. First check sum/tacctprev with prtacct. If it looks all right, the latest sum/tacct.MMDD should be patched up, then sum/tacct recreated. A simple patchup procedure would be:

```
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
```

Remember that the *monacct* procedure removes all the tacct.MMDD files; therefore, sum/tacct can be recreated by merging these files together.

6.6.6 Updating Holidays

The file */usr/lib/acct/holidays* contains the prime/nonprime table for the accounting system. The table should be edited to reflect your location's holiday schedule for the year. The format is composed of three types of entries:

1. Comment Lines: Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk.

2. Year Designation Line: This line should be the first data line (noncomment line) in the file and must appear only once. The line consists of three fields of four digits each (leading white space is ignored). For example, to specify the year as 1982, prime time at 9:00 a.m., and nonprime time at 4:30 p.m., the following entry would be appropriate:

   ```
   1982 0900 1630
   ```

   A special condition allowed for in the time field is that the time 2400 is automatically converted to 0000.

3. Company Holidays Lines: These entries follow the year designation line and have the following general format:

   ```
   day-of-year Month Day Description of Holiday
   ```

   The day-of-year field is a number in the range of 1 through 266 indicating the day for the corresponding holiday (leading white space is ignored). The other three fields are actually commentary and are not currently used by other programs.

6.6.7 Daily Reports

Runacct generates five basic reports upon each invocation. They cover the areas of connect accounting, usage by person on a daily basis, command usage reported by daily and monthly totals, and a report of the last time users were logged in. The following paragraphs describe the reports and the meanings of their tabulated data.

In the first part of the report, the from/to banner should alert the administrator to the period reported on. The times are the time the last accounting report was generated until the time the current accounting report was generated. It is followed by a log of system reboots, shutdowns, power fail recoveries, and any other record dumped into */etc/wtmp* by the *acctwtmp* program [see *acct*(1M) in the *IRIX System Administrator Reference Manual*].

The second part of the report is a breakdown of line utilization. The TOTAL DURATION tells how long the system was in multiuser state (able to be accessed through the terminal lines). The columns are:

LINE	The terminal line or access port.
MINUTES	The total number of minutes that line was in use during the accounting period.
PERCENT	The total number of MINUTES the line was in use divided into the TOTAL DURATION.
# SESS	The number of times this port was accessed for a *login*(1) session.
# ON	This column does not have much meaning any more. It used to give the number of times that the port was used to log a user on; but since *login*(1) can no longer be executed explicitly to log in a new user, this column should be identical with SESS.
# OFF	This column reflects not just the number of times a user logged off but also any interrupts that occur on that line. Generally, interrupts occur on a port when the getty(1M) is first invoked when the system is brought to multiuser state. Where this column does come into play is when the # OFF exceeds the # ON by a large factor. This usually indicates that the multiplexer, modem, or cable is going bad, or there is a bad connection somewhere. The most common cause of this is an unconnected cable dangling from the multiplexer.

During real time, */etc/wtmp* should be monitored as this is the file that the connect accounting is geared from. If is grows rapidly, execute acctcon1 to see which tty line is the noisiest. If the interrupting is occuring at a furious

rate, general system performance will be effected.

Daily Usage Report

This report gives a by-user breakdown of system resource utilization. Its data consists of:

UID	The user ID.
LOGIN NAME	The *login* name of the user; there can be more than one *login* name for a single user ID, this identifies which one.
CPU (MINS)	This represents the amount of time the user's process used the central processing unit. This category is broken down into PRIME and NPRIME (nonprime) utilization. The accounting system's idea of this breakdown is located in the /usr/lib/acct/holidays file. As delivered, prime time is defined to be 0900 through 1700 hours.
KCORE-MINS	This represents a cumulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into PRIME and NPRIME amounts.
CONNECT (MINS)	This identifies "Real Time" used. What this column really identifies is the amount of time that a user was logged into the system. If this time is rather high and the column "# OF PROCS" is low, this user is what is called a "line hog". That is, this person logs in first thing in the morning and does not hardly touch the terminal the rest of the day. Watch out for these kinds of users. This coumn is also subdivided into PRIME and NPRIME utilization.
DISK BLOCKS	When the disk accounting programs have been run, the output is merged into the total accounting record (tacct.h) and shows up in this column. This disk accounting is accomplished by the program acctdusg.

OF PROCS — This coumn reflects the number of processes that was invoked by the user. This is a good column to watch for large numbers indicating that a user may have a shell procedure that runs amock.

O SESS — This is how many times the user logged onto the system.

DISK SAMPLES — This indicates how many times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE — An often unused field in the total accounting record, the FEE field represents the total accumulation of widgets charged against the user by the *chargefee* shell procedure [see acctsh(1M)]. The *chargefee* procedure is used to levy charges against a user for special services performed such as file restores, etc.

Daily Command and Monthly Total Command Summaries

These two reports are virtually the same except that the Daily Command Summary only reports on the current accounting period while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last invocation of *monacct.*

The data included in these reports gives an administrator an idea as to the heaviest used commands and, based on those commands' characteristics of system resource utilization, a hint as to what to weigh more heavily when system tuning.

These reports are sorted by TOTAL KCOREMIN, which is an arbitrary yardstick but often a good one for calculating "drain" on a system.

COMMAND NAME — This is the name of the command. Unfortunately, all shell procedures are lumped together under the name sh since only object modules are reported by the process accounting system. The administrator should monitor the frequency of programs called a.out or core or any other name that does not seem quite right. Often people like to work on their favorite version of backgammon only they do not want everyone to

know about it. Acctcom is also a good tool to use for determining who executed a suspiciously named command and also if superuser privileges were used.

NUMBER CMDS — This is the total number of invocations of this particular command.

TOTAL KCOREMIN — The total cumulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.

TOTAL CPU-MIN — The total processing time this program has accumulated.

TOTAL REAL-MIN — The total real-time (wall-clock) minutes this program has accumulated. This total is the actual "waited for" time as opposed to kicking off a process in the background.

MEAN SIZE-K — This is the mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.

MEAN CPU-MIN — This is the mean derived between the NUMBER CMDS and TOTAL CPU-MIN.

HOG FACTOR — This is a relative measurement of the ratio of system availability to system utilization. It is computed by the formula

```
(total CPU time) / (elapsed time)
```

This gives a relative measure of the total available CPU time consumed by the process during its execution.

CHARS TRNSFD — This column, which may go negative, is a total count of the number of characters pushed around by the read(2) and write(2) system calls.

BLOCKS READ — A total count of the physical block reads and writes that a process performed.

Last Login

This report simply gives the date when a particular *login* was last used. This could be a good source for finding likely candidates for the archives or getting rid of unused *login* and *login* directories.

6.6.8 Summary

The IRIX system accounting was designed from a IRIX system administrator's point of view. Every possible precaution has been taken to ensure that the system will run smoothly and without error. It is important to become familiar with the C programs and shell procedures. The manual pages should be studied, and it is advisable to keep a printed copy of the shell procedures handy. The accounting system should be easy to maintain, provide valuable information for the administrator, and provide accurate breakdowns of the usage of system resources for charging purposes.

6.6.9 Files in the */usr/adm* Directory

diskdiag	diagnostic output during the execution of disk accounting programs
dtmp	output from the *acctdusg* program
fee	output from the *chargefee* program, ASCII tacct records
pacct	active process accounting file
pacct?	process accounting files switched via turnacct
Spacct?.MMDD	process accounting files for MMDD during execution of *runacct*

Files in the */usr/adm/acct/nite* Directory

active	used by *runacct* to record progress and print warning and error messages. activeMMDD same as active after *runacct* detects an error
cms	ASCII total command summary used by prdaily
ctacct.MMDD	connect accounting records in tawcct.h format
ctmp	output of acctcon1 program, connect session records in ctmp.h format
daycms	ASCII daily command summary used by prdaily
daytacct	total accounting records for 1 day in tacct.h format
disktacct	disk accounting records in tacct.h format, created by dodisk procedure
fd2log	diagnostic output during execution of *runacct* (see *cron* entry)
lastdate	last day *runacct* executed in date +%m%d format
lock lock1	used to control serial use of *runacct*

lineuse	tty line usage report used by prdaily
log	diagnostic output from acctcon1
logMMDD	same as log after *runacct* detects an error
reboots	contains beginning and ending dates from wtmp, and a listing of reboots
statefile	used to record current state during execution of *runacct*
tmpwtmp	wtmp file corrected by wtmpfix
wtmperror	place for wtmpfix error messages
wtmperrorMMDD	same as wtmperror after *runacct* detects an error
wtmp.MMDD	previous day's *wtmp* file

Files in the */usr/adm/acct/sum* Directory

cms	total command summary file for current fiscal in internal summary format
cmsprev	command summary file without latest update
daycms	command summary file for yesterday in internal summary format
loginlog	created by lastlogin
pact.MMDD	concatenated version of all pacct files for MMDD, removed after reboot by remove procedure
rprtMMDD	saved output of prdaily programs
tacct	cumulative total accounting file for current fiscal
tacctprev	same as tacct without latest update
tacctMMDD	total accounting file for MMDD
wtmp.MMDD	saved copy of *wtmp* file for MMDD, removed after reboot by remove procedure

Files in the */usr/adm/acct/fiscal* Directory

cms?	total command summary file for fiscal ? in internal summary format
fiscrpt?	report similar to prdaily for fiscal ?
tacct?	total accounting file for fiscal ?

7. Disk Devices

This chapter covers what you need to know about the disk devices on your workstation or server. All system software and user files are kept on the hard disk device(s). The cartridge tape device is used primarily for high-speed file system backups and data transfer. The topics are:

- Making devices known to the operating system
- Identifying disk devices
- Disk drive formatting and partitioning
- Defining logical volumes
- Bad disk block handling
- Swap space

This chapter does not deal with file systems or the information stored on disk devices. Those subjects are covered in Chapter 8, File System Administration.

7.1 Identifying Devices to IRIX

Before a disk or tape device can be used on a computer running the IRIX operating system, it must be made known to the system. For equipment that comes with your computer, the process of identifying devices is part of configuration and is done automatically as the system is booted.

The traditional way of handling the identification is through an entry in the */dev* directory of the *root* file system. Of course, an entry in a directory is a file (or another directory), and conceptually a disk device is treated as if it were a file. There is a difference, however, which leads to the practice of referring to devices as "special" files. In the place where a regular file would show the character count for the file, for a special file you find two decimal numbers called the major and minor numbers. Figure 7-1 shows excerpts from the output of *ls*(1) *-l* commands on a user's directory and the */dev* directory structure.

(a regular file)

```
-rw-r-----   1 mildred   design     1050 Apr 23 08:14 dm.ol
```

(device files)

```
brw-------   2 root   sys   22,  32 Apr 15 10:59 /dev/dsk/dks0d1s01
brw-------   2 root   sys   22,  32 Apr 15 10:59 /dev/root
brw-------   2 root   sys   22,  38 Apr 12 13:51 /dev/dsk/dks0d1s1
brw-------   2 root   sys   22,  38 Apr 12 13:51 /dev/usr

crw-------   2 root   sys   22,  32 Apr 15 10:58 /dev/rdsk/dks0d1s0
crw-------   2 root   sys   22,  32 Apr 15 10:58 /dev/rroot
crw-------   2 root   sys   22,  38 Apr 12 13:51 /dev/rdsk/dks0d1s1
crw-------   2 root   sys   22,  38 Apr 12 13:51 /dev/rusr
```

Figure 7-1. Directory Listing Extracts: Regular and Device Files

Figure 7-2 displays the parts of device name in bold face to illustrate its construction:

Device Name		Component
/dev/dsk/**dks**0d1so	*dks*	= *SCSI Device*
	ips	= *ESDI Device*
	xyl	= *SMD Device*
	ipi	= *IPI Device*
/dev/dsk/dks**0**d1so	*0*	= *Controller 0*
	1	= *Controller 1*
/dev/dsk/dks0**d1**so	*d0*	= *Main Hard disk (d1 for SCSI)*
	d1	= *Second Hard disk (d2 for SCSI)*
/dev/dsk/dks0d1**sn**	*sn n*	= partition number

Figure 7-2. Device Name Construction

What appears from directory listings, in Figure 7-1, show a regular file (indicated by the dash "-" in the first position of the line) with these characteristics:

- The file has 1050 characters
- The filename is *dm.ol*
- It is owned by user **mildred** who is a member of the **design** group
- The owner has read/write permission, group members have read permission only, other users have permissions for nothing at all

The device files, as shown in the bottom half of Figure 7-1, have these characteristics:

- Major and minor device numbers appear where the character count appears in the listing of a normal file

  The major number is the number of the device controller or driver. This number indicates the device type, such as a terminal or hard disk. Actually, the major device number is an index into a table of devices in the kernel.

The minor device number is passed as a parameter to the driver. For example, an ESDI hard disk device name with a major number of 4 and a minor number of 0 indicates a primary drive and the root partition. The major and minor device numbers are found in the inode.

- There are devices that have identical major and minor numbers, but they are designated in one entry as a block device (a `b` in the first column) and in another entry as a character device (a `c` in the first column).

  Notice that such pairs of files have different file names or are in different directories (for example, */dev/dsk/dks0d1s0* and */dev/rdsk/dks0d1s0*).

- There are alias names.

- The files are owned by *root*, and no group or other user has any permission to use them. This means that only processes with the *root* ID can read from and write to the device files. The cartridge tape device, 1/2" tape device, and tty terminals are exceptions to this rule.

  This is the area where there are distinct differences between block and character devices. For more information on block and character devices, see the following section.

7.1.1 Block and Character Devices

The identification as a block device or a character device has more to do with how the device is accessed rather than with any physical characteristics of the device.

Block device names are used when the device is read from and written to in logical blocks. In the IRIX system, standard C language file I/O routines work with blocks. So random access storage devices such as hard disks often require block device names.

When the device is read from or written to in arbitrarily sized blocks, a character device name is used. For example, some file maintenance routines do I/O in this manner. Character devices are also referred to as "raw" devices. The device file directory listing in Figure 7-1 gives an example of the naming difference: the raw device name of the disk drive is in directory */dev/rdsk*, where the "r" indicates raw.

7.1.2 Defining a New Special File

The need to define new special device files occurs infrequently. If you add devices, the autoboot process takes care of defining the files. When the need does occur, however, there is a IRIX system command, *mknod*(1M), available to do it.

The general format of the *mknod* command is:

mknod name **b** | **c** *major minor*

mknod *name* **p**

The options of *mknod* are:

name	Specifies the *name* of the special file.
b	Specifies a block device.
c	Specifies a character device. The or sign (|) indicates you must specify one or the other.
major	The *major* number indicates a device type that corresponds to the appropriate entry in the block or character device switch tables.
minor	The *minor* number indicates a unit of the device. It distinguishes peripheral devices from each other.
p	Specifies the special file as a first-in, first-out device. This is also known as a named pipe. (For more on pipes, see the *IRIS-4D Programmer's Guide*.)

A second system command, *MAKEDEV*(1M), creates specified device files in the directory in which it is executed. It is primarily used for constructing the standard */dev* directory. See the manual pages for further information on this command.

7.2 Formatting and Partitioning

Formatting a disk means establishing addressable areas on the storage medium. Partitioning means assigning file systems or other logical units to addressable areas.

7.2.1 Formatting Disks

Before a disk can be used for the storage of information, it must be formatted. Until the medium is formatted, its surfaces are simply uncharted areas treated with a substance that accepts and holds magnetic charges. Formatting imposes an addressing scheme onto these magnetic surfaces. For disks, formatting maps both sides of the disk into tracks and sectors that can be addressed by the disk controller. A portion of the disk is reserved for data having to do with the specific disk. The volume table of contents resides in that area. The volume table of contents (VTOC, also called the volume header) shows how the partitions on the disk are allocated. On a hard disk, an additional use of the reserved area is to map portions of the disk that may not be usable. Formatting a previously used disk, in addition to redefining the tracks, erases any data that may be there. The utility *fx*(1) is used to format disk drives. The utility *prtvtoc*(1) is used to print out the current VTOC contents. See Chapter 12 for more information on *fx*.

7.2.2 Hard Disk Partitioning

Hard disks are shipped from the factory already formatted. Partitions on the hard disk devices on your workstation are allocated in a standard arrangement. The arrangement varies according to whether you have one or more disk drives, and what size the drives are.

The first disk is partitioned to accommodate the *root*, *swap*, and */usr* partitions.

The default partitions are generic in nature and should be evaluated by the system administrator. After your system has been in operation for a few months, you may come to feel that a different arrangement would better serve the needs of your users.

7.2.3 Changing Hard Disk Partitions

This section describes how to change the partitioning of a disk drive. Once disks have been formatted and partitioned, these partitions may be used as file systems, parts of a logical volume, or as raw disk space.

The following steps should precede any partition changing:

1. If there is *any* valuable data on the disk to be repartitioned, you should make a complete backup. Repartitioning could make this data inaccessible.

2. The disk drive to be partitioned should not be in use. That is, you should make sure that no file systems are mounted, or that any programs are accessing the drive.

How the disk should be partitioned depends on the use of the drive. Many users just use the disk as a single shared file system. Other users may wish to have a smaller partition for private work. Before starting the partitioning process, you should also determine how much space is available on the drive, and then decide how to divide up that space.

You can partition the disk in up to 8 sections, each one of a different size. The size of a partition is measured in 512-byte blocks. Partitions are numbered starting at 0. Partition numbers 8, 9, and 10 are reserved on some types of drives for internal information.

As an example, we will partition a 380Mb SCSI drive into 4 even pieces. We will be using a new SCSI drive on a system with an ESDI root drive, so we will be examining the device */dev/dsk/dks0d1vh*.

First, we will use *prtvtoc*(1) to print out the current information about the drive:

```
prtvtoc /dev/dsk/dks0d1vh
```

The output looks like this:

```
* /dev/dks/dks0d1vh (bootfile "/unix")
*     512 bytes/sector
*      45 sectors/track
*       9 tracks/cylinder
*    1547 cylinders
*       6 cylinders occupied by header
*    1541 accessible cylinders
*
* No space unallocated to partitions

Partition  Type  Fs  Start: sec  (cyl)  Size:  sec   (cyl)   Mount Dir
 0          efs  yes       2484    (?)      32076     (?)
 1          raw           35190    (?)     102258     (?)
 6          efs  yes     137448    (?)     489087     (?)    /d
 7          efs            2484    (?)     624051     (?)
 8       volhdr               0 (   0)       2484     (?)
10       volume               0 (   0)     626535  (1547)
```

If *prtvtov* fails, use *hinv*(1) to make sure that the drive number is the one you think, or use *fx*(1) to make sure that the drive is formatted.

The (?) in the cylinder column means that the partition does not start on an exact cylinder boundary. For SCSI disks this is normal. For other disk types, there may be a small loss of performance if partitions do not start on a cylinder boundary.

Look at the size column for partitions 0, 1 and 6. In our example, we have 32706 + 102258 + 489087 = 624051 sectors to use. (Recall that a sector is a 512-byte block.) Look at the start sector numbers, and notice that partition 7 overlaps 0, 1, and 6. We want to divide this chunk into four equal pieces, of 624051 / 4 = 156012 sectors each.

We'll use *dvhtool*(1M) to partition the drive. Remember to back up *any* data you wish to preserve! Use the command:

```
dvhtool /dev/rdsk/dks0d1vh
```

Note that we use the raw disk device name in this case. *dvhtool* will respond:

```
Command?  (read, vd, pt, dp, write, bootfile, or quit):
```

Type **pt** for "partitions". *dvhtool* will respond:

```
Command?  (part nblks lstblk type, or l):
```

Typing **l** here will list the partition table. Typing return will return you to the main menu. The **l** output for our example drive looks like this:

```
current contents:
        part    n_blks  1st_blk type
        0:      32706   2484    efs
        1:      102258  35190   raw
        6:      489087  137448  efs
        7:      624051  2484    efs
        8:      2484    0       volhdr
        10:     626535  0       volume
```

Note that partition 0 does *not* begin at sector 0— the volume header must come first. Do *not* attempt to change the volume header location or decrease its size. You may increase it if you wish, though.

The first partition that we want will start at sector 2484 and be 156012 sectors long. We also wish to specify that this partition is going to be used as a file system. So type this response to the *dvhtool* prompt:

```
Command?  (part nblks lstblk type, or l) 0 156012 2484 efs
```

The next partition starts at 2484 + 156012 = 158496, so the next command to *dvhtool* would be:

```
1 156012 158496 efs
```

Likewise for partitions 2 and 3.

Be careful not to allow partitions to overlap. *dvhtool* will permit it, but the resultant partitions will be unsuitable for use as file systems.

When you're done, type **l** to look at your work. Then type RETURN to return to the top level menu. If you are satisfied with what you have done, commit the changes to disk by typing:

```
write
```

You may then quit.

Use *prtvtoc*(1) to make one more check that the disk is laid out the way you want it.

You may now create filesystems on these partitions using *mkfs*(1).

7.2.4 Changing Partitions to Increase Swap Space

If you run very large programs, you might run out of swap space. Use *swap –l* to monitor swap space usage. If you find that you are running out of swap space, two solutions are available: you can add more memory, or you can add more swap space. Adding swap space will not improve the performance of large programs, but it will permit them to be run successfully. See the next section for more discussion on swap space, and methods of increasing it.

7.3 Swap Space

The IRIX operating system uses a portion of the disk as *swap space* for temporarily saving part or all of a user's program when there is not enough physical memory to contain all the running programs.

To avoid loading more programs and data than there is available memory, IRIX ensures that there is enough physical memory, plus swap space, to hold all the programs that can run at one time.

You can determine the amount of memory a program uses by adding the amount of memory reserved for data to the amount of stack space and subtracting the size of the code from the result. For example, if a program has 6Mbytes of code and 25Mbytes of data, plus it uses 2Mbytes of stack space, then it takes up 27 of swap space and memory. (Since the program's code is always available in the disk file, it does not need to be added to the amount of required memory.)

The IRIS 4D is configured (by default) with between 8 and 16 Mbytes of main memory and approximately 50 Mbytes of swap space. For example, assume a program that requires 24 Mbytes to run. If your 4D has 12Mbytes of memory and 50Mb of swap space, you can run a total of 62Mb of programs. There is no per-process limit to the size of a program (see chapter 10 on how to create such a limit).

If you run multiple copies of these programs, the system may run out of space, and terminate one or more of the programs. If this happens, you need to increase the available swap space.

To determine how much swap space is already configured in your machine, use the *swap*(1M) command:

```
swap -l
```

You may also want to fine-tune the swap area of the disk used by the operating system if you are running applications which require the system to swap out programs frequently.

7.3.1 Increasing the Swap Space on a Multi-Disk System

In order to double the default amount of swap space, you might use another disk drive as follows:

```
Partition/slice
    0           Temporary space (mount as /tmp)
    1           Swap space
    6           usr2
```

Note that the operating system continually writes onto the partition that is used as swap space, completely destroying any data that might exist there. Be sure that the swap partition does not overlap any user file system partitions.

Once you have decided on the partition, edit the file */etc/init.d/MOUNTFSYS* to add this partition permanently as a swap partition. At the end of this file, include the line:

```
swap -a /dev/rdsk/devicename
```

where you replace the word *devicename* with the device name where the swap partition is located (such as `ips0d1s1`).

You could also modify the file */etc/fstab* to document (in the form of a comment) that the chosen partition is being used as a swap partition.

7.3.2 Increasing the Swap Space on a One Disk System

Suppose you don't have the luxury of a multiple-disk system. This section explains how to increase the size of the swap partition on a single disk by reallocating space reserved for use in another partition. The example worked through here reallocates space from the */usr* partition. This procedure can be dangerous if done incautiously. Explore other solutions before contemplating this one. Be *sure* that you have done a complete backup before starting.

Increasing the size of the existing swap space is a straightforward process, provided that a backup is made and a few simple calculations are done. The

steps involved are listed below, and are fully explained through the rest of the section.

1. Be sure there is room available in */usr* and note the current size of the swap partition. There must be excess space which can be taken away from */usr*.
2. Make a backup of both the */usr* and *root* file systems using the Backup command.
3. Print the current disk label for the drive to be modified, and calculate the changes to swap and */usr*.
4. Boot the *fx* program, modify the label, and write the label back to the disk.
5. Reboot the system, and mount the disk.
6. Restore the system.

Remember that available disk space is a constant, so by increasing the size of the swap space, you are decreasing the size of another file system.

7.3.3 Step 1: Verify Available Space

Begin by verifying the available disk space on */usr*. Do this with the *df*(1) command. *df* **–k** lists disk usage and availablility in kilobytes (KB), as in this example:

```
# df -k
Filesystem Type  kbytes    use   avail  %use   Mounted on
/dev/root  efs    15390   8380    7010   54%      /
/dev/usr   efs   236250 111774  124476   47%     /usr
```

In our example, there are 124,476 KB, or over 120 MB, available in the */usr* file system. If there is not enough space in the file system, you would need to free space by removing files and/or directories. In our example, we plan to add 16 MB to the swap space, so we have sufficient space available to us.

Next, find the current size of the swap partition, using *swap* **-l**. (For more information, see the man page on *swap*(1).)

```
# swap -l
```

```
path                dev   swaplo  blocks    free
/dev/dsk/dks0d1s1 22,33        0  102256  102256
```

It is approximately 102,000 blocks, or 50 megabytes. (Recall that a block is 512 bytes.)

7.3.4 Step 2: Make a Backup

Make a complete file system backup using *Backup*(1). This is a crucial step. Do not omit the backup. This procedure will create a new file system, destroying all existing data on the disk. You may wish to verify the success of the backup.

7.3.5 Step 3: Calculate Changes

Print a copy of the disk label for examination. You will use this information to calculate the changes to the swap and */usr* partitions. Use the system command *prtvtoc*(1M) to dump the label and direct it to a file for printing.

```
# prtvtoc /dev/rdsk/dks0d1vh > filename    (SCSI)
# prtvtoc /dev/rdsk/ips0d0vh > filename    (ESDI)
# prtvtoc /dev/rdsk/xyl0d0vh > filename    (SMD)
# prtvtoc /dev/rdsk/ipi0d0vh > filename    (IPI)
```

Below is an example of a disk label, from a SCSI 380 MB disk on a Personal Iris:

```
# prtvtoc dks0d1vh
* /dev/rdsk/dks0d1vh (bootfile ''/unix'') partition map
*
* Dimensions:
*     512 bytes/sector
*      45 sectors/track
*       9 tracks/cylinder
*    1547 cylinders
*       6 cylinders occupied by header & badblock replacement tracks
*    1541 accessible cylinders
*
* No space unallocated to partitions

Partition  Type  Fs Start:sec  (cyl) Size:sec  (cyl)  Mount
                                                      Directory
0          efs  yes     2430  (   6)   32400  (  80)     /
1          raw         34830  (  86)  102060  ( 252)
6          efs  yes   136890  ( 338)  489645  (1209)   /usr
7          efs          2430  (   6)  624105  (1541)
8       volhdr             0  (   0)    2430  (   6)
10      volume             0  (   0)  626535  (1547)
```

The information we need is in the two (cyl) columns. The first (cyl) column is the starting cylinder for the partition; the second is the length of the partition. In our example, partition 6 (which is the standard */usr* partition on most systems) starts at cylinder 338 and has a length of 1209 cylinders. This partition is an "efs" file system, and is currently mounted as */usr*.

The swap partition is partition 1. It is designated as a raw file system with no file structure. The swap space is never used for permanent storage, so there is no use for a file system on its partition.

Notice that the partition start and size always add up to the start of the next available partition, except for the */usr* partition. The */usr* area can be specified as one large contiguous file system (partition 6) or broken up into smaller ones (partitions 2 through 5). On many drive labels, partitions 2 thorugh 5 are various sizes so more than one file system can be made in the area that is normally the */usr* partition. Most systems will be using partition 6 for */usr*, as we do in this example. If you use any partition other than 0, 1, 6, or 7, you may have to create a device special file using *mknod*(1M), as discussed earlier in this chapter.

If a drive has multiple partitions, you must be careful when changing the partition locations and sizes. It is possible to change the label so that one partition overlaps another. This can be a fatal time bomb for a disk drive, and cause loss of data. To prevent this, find the starting and ending cylinder number for each partition and compare all the partition allocations against each other.

Notice under the type column that there are partitions labelled *volhdr* and *volume*. The *volhdr* contains the disk label describing the disk's configuration and timing, as well as the alternate track table, sash, and keyboard mapping tables. On ESDI, SMD, and IPI disk drives, you'll also see a partition 9, called *trkrepl*. These partitions should not be changed. Experimenting with them can lead to disastrous results.

The */usr* partition and the swap partition (partition 1) are the two on which we will need to focus, since cylinders you add to the swap space will be removed from */usr*. The end of the current swap space is 338 (86 [start] + 252 [size] = 338). As the label shows, this is where the */usr* partition begins.

To increase the size of the swap space, you need to add cylinders to it. To determine how many to add, perform the following calculation:

```
# of cylinders =          change in swap size in bytes
                  ------------------------------------------------------
                  (# bytes/sector)*(# sectors/track)*(# tracks/cylinder)
```

Round up to the next highest integer.

In our example, we want to add 16 MB to the swap space. 16 MB is 16,777,210 bytes (multiply by 1,048,576, or 2**20, the number of bytes per megabyte). 16,777,210 / (512 * 45 * 9) = 80.9, or 81 cylinders. We will add 81 cylinders to the length of partition 1, making it 333 cylinders long.

The next step is to calculate the new starting cylinder and the size of the */usr* partition. The new start cylinder count will be the old start plus 81 cylinders, and the overall length will be 81 cylinders shorter. Therefore:

old start + swap change = new start
338 + 81 = 419 new start of */usr* partition
1209- 81 = 1128 new size of */usr* partition

These numbers will be used later to change the disk label.

7.3.6 Step 4: Modify the Disk Label

Follow the following instructions to boot the disk exerciser program to modify the disk label.

- Bring down the system using the shutdown command.
- Once the system is at the "System Maintenance Menu" enter "Command Monitor" mode.
- From the ">>" prompt, boot *fx* from the disk using one of the following, or a similar command:

```
>>boot dksc(0,1,0)stand/fx -x    (SCSI disk)
>>boot dkip(0,0,0)stand/fx -x    (ESDI disk)
>>boot xyl(0,0,0)stand/fx -x     (SMD disk)
>>boot ipi(0,0,0)stand/fx -x     (IPI disk)
```

Note that the **-x** option invokes the extended mode of *fx*. Extended mode provides additional functions normally used during factory setup or servicing of disks, such as formatting and creating or modifying disk labels. You will need the extended functions in this procedure. You will *not*, however, need them in normal use of *fx*. A mistake in extended mode can destroy all the data on your disk, so avoid casual use of the **-x** option.

fx is a menu-driven program. The *fx* prompt is used to indicate the current menu level. For example, the prompt "fx/label/show" would indicate that you are in the menu of commands that show label information. If you are not familiar with *fx*, refer to its man page, and to Chapter 11 of this manual.

fx will begin by prompting for the name of the device you wish to modify.

At the fx> prompt, choose to examine 5) label. At the fx/label> prompt, choose 1) readin, to read in the current values. At the fx/label/readin prompt, choose 6) all, to read in all values. Move up one level, using the ".." command. At the fx/label prompt, choose 4) set, to set the new label values.

At the fx/label/set prompt, choose 2) partitions. fx will now prompt you for the new partition information. Enter the new values as you calculated them. You may wish to switch to the fx/label/show level, and show 2) partitions, to verify that the values shown are the desired ones. Then make sure to choose 3) sync at the fx/label level, to save the label. Exit from the program by returning to the highest level and choosing exit.

7.3.7 Step 5: Reboot

Now we will boot the system, remake the */usr* file system, and verify our changes.

First boot the system into single user mode. From the "#" prompt, make the */usr* file system and label it using the following commands (system responses are not shown):

```
# mkfs /dev/rusr
# labelit /dev/rusr usr sgi
# mount /dev/usr usr
```

Verify that the partitions modified in fx are correct by printing the disk label using the prtvtoc command, as before. The label should reflect the changes to the partition. Use *df* **–k** as before, to see the reduction in size of */usr*:

```
# df -k
Filesystem Type  kbytes    use   avail  %use   Mounted on
/dev/root  efs    15390   8380    7010   54%      /
/dev/usr   efs   219240      0  218240    0%    /usr
```

Previously, *df* reported */usr* as 236,250 KB in size. Now *df* reports */usr* as 219,240 KB. This verifies that */usr* has been reduced by 17,010 KB, or a little over 16 MB. Note that the *usr* filesystem is completely empty.

You may also wish to verify that the swap space added is recognized by the system by using the *swap –l* command.

7.3.8 Step 6: Restore

When you are satisfied that the new partitioning is correct, you need to restore the */usr* files. To do this, halt the system, and enter the System Maintenance Menu. From there you can execute the Restore System utility and recover the system. Consult the manual pages for Backup and Restore if you are unfamiliar with them.

7.4 Logical Volumes

We've just discussed the use of partitions, which cause one disk drive to behave as if it were several smaller drives. Several partitions may be built up into one logical disk drive through the use of *logical volumes*. A logical volume might include partitions from several physical disk drives, and thus be larger than any one of your physical disks.

These logical volumes behave like regular disk partitions. Filesystems may be made on them just as on a regular disk, and then mounted and used in the normal way. The only difference is that they may be built from several partitions, and across more than one physical disk device. A logical volume disk driver, referred to as *lv*, does the job of mapping requests on logical volume devices onto the underlying physical devices, in a manner transparent to the user.

The exception is that the root partition must never be a logical volume, since the utilities required for logical volume intialization must reside on it.

Logical volumes can be used to extend an existing file system onto a new disk drive, with the use of the *growfs*(1M) command. After reading this section, you may wish to refer to the man page or to Section 8.3.3 of this manual for further information on *growfs*.

IRIX allows for the creation of *striped* logical volumes. Disk storage on a striped volume is allocated alternately among partitions. The *granularity*, or width, of the stripes may optionally be set.

The concept of logical volumes thus adds a layer of abstraction to the concept of the physical disk drive, and allows more efficient disk use.

Logical volumes are administered by means of a file defining the volumes known to the system (*/etc/lvtab*), and three utilities: *mklv*, *lvinit*, and *lvck*. These will be described in more detail below.

A new logical volume is created by adding an entry to */etc/lvtab* and then running *mklv*. Existing logical volumes are "connected" to the system at boot time by *lvinit*: it is not normally necessary to run *lvinit* explicitly.

Lvck is a diagnostic tool. You need to run this only if there are problems.

7.4.1 */etc/lvtab*

The file */etc/lvtab* contains a table of the logical volumes used by IRIX. It is read by the utilities that create, install, and check the consistency of logical volumes. You can modify it with a text editor to add new logical volumes or to change existing ones.

The entries have the form:

volume_device_name: [*volume_name*] : [*options*] : devs=*device_pathnames*

The *volume_device_name* is of the form lv*n*, where *n* is an integer by default between 0 and 9. The logical volume will be accessed through the device special files */dev/dsk/lvn* and */dev/rdsk/lvn*, as described in section 7.1.

The *volume_name* is an arbitrary identifying name for the logical volume. This name is included in the logical volume labels on each of the partitions comprising the logical volume. It is then used by utilities to verify that the logical volume on the disks is actually the volume expected by */etc/lvtab*. Any name of up to 80 characters may be used; you should probably choose something meaningful as an aid to identification by humans. You may leave this field blank, but this is not recommended.

The options are specified in the format "option_name=number", with no spaces surrounding the equals sign. You may specify more than one option, separated by colons. Currently recognized options are stripes= and step=.

The stripes option creates a logical volume that is striped across its constituent devices. The number of device pathnames must be a multiple of this parameter. This specifies the number of ways the volume storage is striped across its constituent devices. For example, suppose you have a logical volume with 6 constituent devices and a stripes parameter of 3. The logical volume is set up to stripe across the first three devices until they are filled, then to stripe across the second three.

The step option futher specifies the granularity, or step size, with which the storage is distributed across the components of a striped volume. Granularity is measured in disk blocks. The default step is the device tracksize, which is generally a good value to use.

The device pathnames are listed following any options. They are the block special file pathnames of the devices constituting the logical volume, as described in section 7.1. They must be separated by commas. The partitions named must be legal for use as normal data storage, and not dedicated partitions, such as *swap*.

7.4.2 *mklv*

This utility constructs the logical volumes by writing labels for the devices that will make up the volume. The general format of the *mklv*(1M) command is:

```
/etc/mklv [-f] volume_device_name
```

mklv reads the entry in */etc/lvtab* identified by *volume_device_name*, and constructs the logical volume described. It labels the devices appropriately, then intializes the logical volume device for use.

An existing logical volume can be extended simply by adding additional device pathnames to the */etc/lvtab* entry, then running *mklv* on that entry.

Normally, *mklv* checks all the named devices to see if they are already part of a logical volume, or contain a filesystem. The option –f forces *mklv* to skip those checks.

Various errors can arise when trying to create a logical volume. For example, one of the specified disks might be missing, or there might be a typographical error in the new *lvtab* entry. The man page for *mklv* describes the possible error messages and their meanings.

7.4.3 *lvinit*

/etc/lvinit(1M) intializes the logical volume device driver which allows access to disk storage as logical volumes. It works from entries in */etc/lvtab*, as does *mklv*. Its form is:

```
lvinit [volume_device_names]
```

Without arguments, *lvinit* initializes every logical volume with an entry in */etc/lvtab*. With arguments, it initializes only those named in the argument list. The component devices of the logical volumes to be initialized must have been previously labelled as members of the volume by *mklv*.

lvinit is run automatically at system boot time. It is not normally necessary to invoke it explicitly. It performs sanity checks when initializing the volumes, and prints messages describing any error conditions.

If error messages from *lvinit* are seen during system boot, you will want to run *lvck* to obtain more information about the problem.

7.4.4 lvck

lvck(1M) checks the consistency of logical volumes by examining the logical volume labels of devices constituting the volumes.

Possible errors and the messages from *lvck* which describe them are detailed in the *lvck* man page.

Invoked without parameters, *lvck* checks every logical volume for which there is an entry in */etc/lvtab*.

Invoked with the name of a logical volume device, eg. **lv0**, *lvck* checks only that entry in */etc/lvtab*.

Invoked with the **–d** flag, *lvck* ignores */etc/lvtab* and searches through all disks connected to the system in order to locate all logical volumes present. *lvck* prints a description of each logical volume found in a form resembling an *lvtab* entry. This option facilitates recreation of an *lvtab* for the system, if necessary. You might use this option if, for example, */etc/lvtab* became corrupted or if you had somehow lost track of which disks were connected during a system reconfiguration.

Invoked with the device block special filename of a disk device, *lvck* prints any logical volume label which exists for that device, again in a form resembling an *lvtab* entry. Note that this mode of *lvck* is purely informational; no checks are made of any other devices mentioned in the label.

lvck has some repair capabilities. If it determines that the only inconsistency in a logical volume is that a minority of devices have missing or corrupt labels, it is able to restore a consistent logical volume by rewriting good labels. *lvck* interactively queries the user before attempting any repairs on a volume.

7.4.5 Examples

Logical volumes can be used:

1. To provide storage for a new filesystem on disks newly added to a system.
2. To allow an existing filesystem to grow onto newly added disks.

Let's look at examples of these two cases.

Creating a New Filesystem on Newly Added Disks

Suppose that some new disks have been added to your system in order to provide storage for a new filesystem. We assume that they are connected and initialized correctly.

Decide which partitions of these new disks you wish to use for the new filesystem. (Normally, when adding a new filesystem like this, you will want to use the whole of all the disks: i.e., partition 7 of each disk).

Decide if you want to make a striped volume. The benefit of striping is improved throughput; however, it does impose some minor restrictions. If you want to stripe, all the drives (or to be more exact, the partitions on them that you are using) must be exactly the same size. And if you later want to add more drives to the volume, you must add them in units of the striping. That is, if you want to add disks to a 3-way striped volume, you must add them three at a time.

Also, to obtain the best performance benefits of striping, you should arrange to connect the disks you are striping across on different controllers. In this arrangement, there are independant data paths between each disk and the system.

Add an entry to */etc/lvtab* containing the pathnames of the new disks that are to be part of the new volume. (See the *lvtab* man page for details of the syntax of *lvtab* entries). Example:

```
lv0:project one:stripes=2:devs=/dev/dsk/dks0d2s7,/dev/dsk/dks1d0s7
```

In this example, the logical volume named *project one* consists of two partitions from two separate disks. Storage is striped across the two disks. (Note that it is not normally necessary to specify the "step=" parameter. An appropriate value will be used automatically).

Run *mklv* to place the labels on the new disks which identify them as parts of a logical volume:

```
mklv lv0
```

Now you will find device names */dev/dsk/lv0* and */dev/rdsk/lv0* have been created. These can now be accessed exactly like any regular disk partition.

To create the new filesystem, run *mkfs* on */dev/rdsk/lv0* as you would on a regular disk:

```
mkfs /dev/rdsk/lv0
```

Then, you can mount */dev/dsk/lv0* exactly as you would mount a filesystem on a regular disk. You will probably want to add an entry to */etc/fstab* to automatically mount this filesystem, e.g.

```
/dev/dsk/lv0 /some_directory efs rw raw=/dev/rdsk/lv0 0 0
```

Note: When creating a striped volume, all the partitions must be the same size. Unfortunately, the default partitioning for drives of similar sizes but from different manufacturers can be slightly different. In this case you will see an error message similar to this from from mklv:

```
lv0:project one:stripes=2:devs= \
        /dev/dsk/dks0d2s7,      \
        /dev/dsk/dks1d0s7  <INCORRECT PARTITION SIZE>
```

In this case, you will need to adjust the partition sizes: refer to the section below on using *dvhtool* to do this.

Extending an Existing Filesystem

Suppose you have a filesystem (on a regular disk) which has become very full, and you would like to have more space available in it.

You will need a new disk (or at least a partition on a disk) which you can dedicate to providing this extra space. This new partition must not, of course, be currently used for anything, or contain any valuable data!

First, it's advisable to make a backup of the filesystem you are going to extend.

Next, add an entry to */etc/lvtab* for a new logical volume. The first device in this volume should be the device on which the existing filesystem resides. This is then followed in the description by the new device you are adding. Example:

```
lv2:new development volume:devs=/dev/dsk/ips0d1s7, /dev/dsk/ips0d2s7
```

In this example, */dev/dsk/ips0d1s7* would be the disk on which the existing filesystem resides, and */dev/dsk/ips0d2s7* is the added disk.

Note: When using a logical volume to extend an existing filesystem, the logical volume may NOT be striped.

Note: Before proceeding further, the filesystem MUST be unmounted.

Run *mklv* to place the labels on the disks which identify them as parts of a logical volume:

```
mklv lv2
```

mklv will mention that one of the devices contains a filesystem and ask if you want to proceed. This is just a safety measure to help avoid inadvertantly doing logical volume operations on disks which do contain important information: answer "y".

You will find device names */dev/dsk/lv2* and */dev/rdsk/lv2* have been created.

The next step is to extend the filesystem to take advantage of the newly added space. This is done with *growfs*. In the example above, you would invoke:

```
growfs /dev/rdsk/lv2
```

This will expand the filesystem into the new space while leaving its existing contents intact. From now on, you will access this filesystem via the logical volume device rather than the original disk device. So you might want to run fsck on */dev/rdsk/lv2* (in the example) to verify that your filesystem is valid (and has indeed increased in size).

You will now mount the logical volume device (eg /dev/dsk/lv2) rather than the original disk device; you will need to update your /etc/fstab to reflect this. (Don't forget to change "raw=" too)!

It is worth noting that this expansion procedure is not a one-shot thing: it can be repeated indefinitely. So you can always add a new disk, add its name to the lvtab entry and then rerun mklv and growfs to further expand the filesystem.

Using *dvhtool* to Adjust Partition Sizes for Striping

As mentioned earlier, disk partitions which are to be striped together must be the same size. However, if you have disks of similar sizes made by different manufacturers, the default partitioning may not be exactly identical.

In order to be able to stripe such disks, you will need to round down the larger partition to the size of the smaller. The current sizes of the partitions (in sectors) can be inspected using *prtvtoc*(1M), which prints out the disk partition tables.

Run *prtvtoc* on the disks you intend to stripe together, and make a note of the sizes in sectors of the partitions which are to be part of your striped volume. You will need to adjust partition sizes on each disk where the partition size is greater than the smallest of the group.

This is a good time to do a sanity check that the problem is really just slightly dissimilar drives, rather than accidental connection of a wrong drive! Incidentally, there is no absolute prohibition against striping a small drive with part of a larger one, but it unlikely to be the best use of resources. Smaller drives are usually slower, and the combination would be throttled by the speed of the slowest device.

Invoke *dvhtool* on the volume header of each disk that needs to be changed. For example, suppose you wish to stripe */dev/dsk/dks0d2s7* and */dev/dsk/dks1d0s7*, and you have found that partition 7 of *dks0d2* is 256000 sectors but partition 7 of *dks1d0* is only 255500 sectors. In this case, you need to change the larger, *dks0d2s7*. Usually, *dvhtool* is used on the raw device, so you would invoke:

```
dvhtool /dev/rdsk/dks0d2vh
```

Now, in answer to the prompt:

```
Command? (read, vd, pt, dp, write, bootfile, or quit):
```

respond **pt** to work on the partition table. You can then list the existing partitions by the **l** command:

```
Command? (part nblks 1stblk type, or l): l
```

The partition display will look something like this:

```
part     n_blks    1st_blk   type
0:       32640     2560      efs
1:       65600     35200     raw
6:       157760    100800    efs
7:       256000    2560      efs
8:       2560      0         volhdr
10:      258560    0         volume
```

Note, incidentally, that the terms "block" and "sector" mean the same thing: a 512-byte disk sector. We want to reduce the size of partition 7 to 255500 blocks, to match the corresponding partition on the other disk. So enter:

```
7 255500 2560 efs
```

Note that we change only the size. The starting block remains unaltered. You can list the partitions again to see the change. Now the changed table must be written back to the disk. Press Return to get back to the main menu, then ***w*** to write out the changes.

You can now proceed with making the striped volume.

7.5 The Bad Block Handling Feature

IRIS workstations and servers has a software feature called bad block handling. The purpose of this feature is to extend the useful life of the integral hard disk by providing mechanisms for:

- Detecting and remembering blocks that are no longer usable
- Reminding you that you need to "fix" some remembered bad blocks
- Restoring the usability of the disk in spite of the bad blocks that exist

Note: This feature applies only to ESDI, SMD, and IPI hard disk device(s). Bad blocks on SCSI drives are handled automatically by the SCSI controller. There is no comparable feature for diskettes or tapes.

It should be pointed out that new bad blocks seldom occur, particularly with the sealed environment of the integral disk, as long as you take reasonable precautions against movement or vibration of the computer while the disk is still spinning. But when a new bad block does occur, the data stored in the bad block is lost and the disk may be unusable in its current state.

The bad block handling feature addresses the problem of restoring the usability of the disk. However, you must address the data loss yourself with whatever backup procedures you deem appropriate for your situation. Backup procedures are also needed to protect against operational errors and other types of hardware failures. These other problems, particularly operational errors, are the dominant cause of data loss on a system. System backups provide protection against operational errors as well as protection against lost data due to new bad blocks. A discussion of backup procedures can be found in Chapter 8, File System Administration.

7.5.1 When Is a Block Bad?

A block is bad when it cannot reliably store data. This is discovered only when an attempt is made to read the data and the read fails. To make life more difficult, a read fail does not always guarantee a bad block. A read fail might also mean problems in the format of the disk or a failure in the controller or the hardware.

A write fail generally signals a problem with the format of the disk or a more basic failure in the disk or disk controller hardware. While all failures are reported, the bad block handling feature does not distinguish genuine bad blocks from format problems or hardware problems. To fix problems of those types you will need to reformat the disk or get the hardware repaired. In either case, you should call your service representative. Several distinct failures occurring about the same time should also prompt you to contact your service representative to check them out.

What Makes a Block Unreliable?

A disk is an analog medium used to store digital data. The analog phenomena involve the magnetic properties of the film coating on disk surfaces. The data are recorded with a very high bit density to get millions of bits in a small space. Because of the density, the significance of small variations in the magnetic properties of the recording medium become magnified. The variations mean that a given portion of the medium may prefer to represent some bit patterns and dislike representing others. Normally, these preferences are insignificant compared to the signal level thresholds. When variations pass the point of being insignificant, the disk has a bad block. If the data pattern in a block matches the preferences in the recording medium, the bad block may escape recognition for a while. If the disk is active, however, the block will eventually be judged unreliable.

How Are Bad Blocks Fixed?

It is not really so much that a bad block is fixed, but that the system finds a way to live with it. A small portion of the disk is set aside from the normally accessible portion of the disk. This portion, call it the alternate-track area, cannot be reached by normal IRIX system commands and system calls. This reserved portion of the disk contains a description of the properties of the disk and other media-specific data.

The volume header portion of the disk includes a set of blocks called the surrogate image region. The mechanism for preserving the apparent accessibility of most disk blocks is to use surrogate image blocks to contain the data that were to have been stored in the bad blocks. The media-specific data also includes a mapping table that maps bad blocks on the disk to these surrogate image blocks. The disk driver software in the operating system translates disk accesses so the data are read from or written to the surrogate

image block. This disk address translation is transparent to the calling software.

Most disks come with a few manufacturing defects. Bad blocks detected in the manufacturer's quality control checks are identified on a label when the unit is delivered. The bad block handling feature provides special software for remembering bad blocks that have been found and for mapping any additional ones that are found. If a surrogate block becomes bad, the software even remaps the original bad block to a new surrogate block.

A Few Blocks Cannot Be Mapped

A few special blocks, all in the volume header data portion of the disk, cannot be mapped:

- The disk block containing the physical description of the disk
- The disk block(s) containing the mapping table

All other blocks, including surrogate image blocks, can be mapped.

7.5.2 When Are Bad Blocks Detected?

Bad blocks are detected when input/output disk operations fail for several successive attempts. This means that data being input or output are lost, but the system can restore use of the disk by mapping bad blocks to surrogate blocks that are readable.

Often Asked Questions

Why doesn't the system try to discover that a given block is bad while the system still has the data in memory?

> Besides the undesirable increase in system size and complexity, severe performance degradation would result. Also, a block can become a bad block after the copy in memory no longer exists.

Why doesn't the system periodically test the disk for bad blocks?

Reading blocks with their current contents may not show a bad block to be bad. A thorough bit pattern test would take so long that you would never run it, even assuming a thorough test could be devised using ordinary write/read operations. The disk manufacturer already has tested the disk using extensive bit pattern tests and special hardware, so manufacturing defects have been dealt with.

Why are disks with manufacturing defects used?

Allowing the disks to contain a modest number of manufacturing defects greatly increases the yield, thereby considerably reducing the cost. Many systems, including this one, take advantage of this cost reduction to provide a more powerful system at lower cost.

8. File System Administration

The file system is the framework upon which IRIX is constructed and maintaining it is extremely important. Failure to do so might result in loss of valuable system- and user- information.

This chapter covers in detail:

- the file system as a functional data structure
- how the file system is constructed
- how to administer and maintain the file system

8.1 What is the File System

In the IRIX system, a file is a one-dimensional array of bytes with no other structure implied. Files are attached to a hierarchy of directories.

A directory is merely another type of file that the user is permitted to use, but not allowed to write; the operating system itself retains the responsibility for writing directories. The combination of directories and files make up a file system. Figure 8-1 shows the relationship between directories and files in a IRIX file system. The circles represent directories.

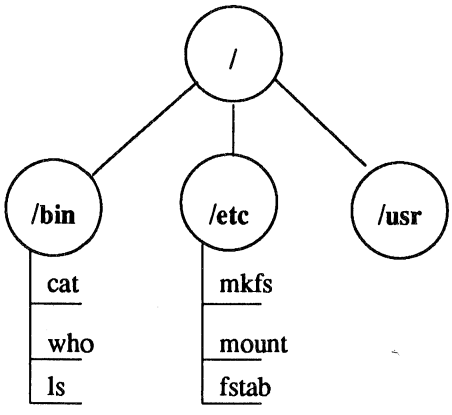


Figure 8-1. A IRIX File System

The starting point of any IRIX file system is a directory that serves as the root. In the IRIX operating system there is always one file system that is itself referred to by that name, *root*. Traditionally, the root directory of the *root* file system is represented by a single slash (/). The file system shown in Figure 8-1, then, is a *root* file system. If we graft another file system onto *root* at a directory called, for example, *usr*, the result can be illustrated by the diagram in Figure 8-2.

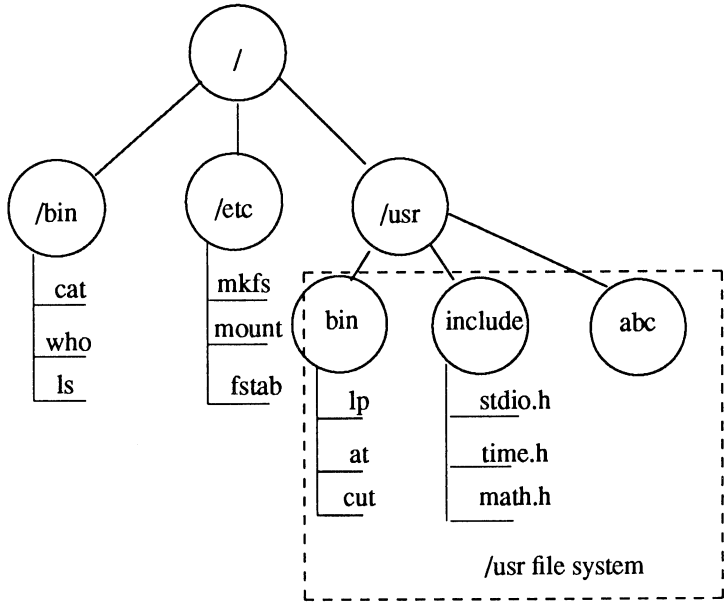


Figure 8-2. Adding the *usr* File System

A directory such as *usr* is referred to in various ways. You sometimes see the terms "leaf" and "mount point" used to describe a directory that is used to form the connection between the *root* file system and another mountable file system. Regardless of the terms used, such a directory is the root of the file system that descends from it. The name of that file system is, coincidentally, the name of the directory. In our example, the file system is *usr*.

The IRIX EFS file system may contain the following types of files:

- directories (d)
- regular files (–)
- character devices (c)
- block devices (b)
- named pipes (p)
- symbolic links (l)
- UNIX domain sockets (s)

The letter in parentheses following each item is the character used by *ls –l* to identify the file type.

Figures 8-1 and 8-2 illustrate the file and directory structure of IRIX file systems. However, IRIX sees things a little differently. As an operating system, IRIX views a file system as a data structure of addressable 512-byte blocks of disk space arranged as illustrated in Figure 8–3. This is the IRIX Extent-based File System (EFS).

An EFS file system contains the following types of blocks:

- the super-block
- inodes
- data blocks

The IRIX EFS file system consists of several "cylinder groups", each containing an inode list and data blocks. Block 0 of the filesystem is reserved (currently unused). Block 1 is the superblock, which describes the file system parameters, such as the block offset of the first cylinder group, the number of blocks in each cylinder group, the number of blocks per cylinder group reserved for inodes, and teh block offset at the free block bitmap. The free block bitmap is found either starting at block 2, or after the last cylinder group. The very last block of the file system is the replicated superblock.

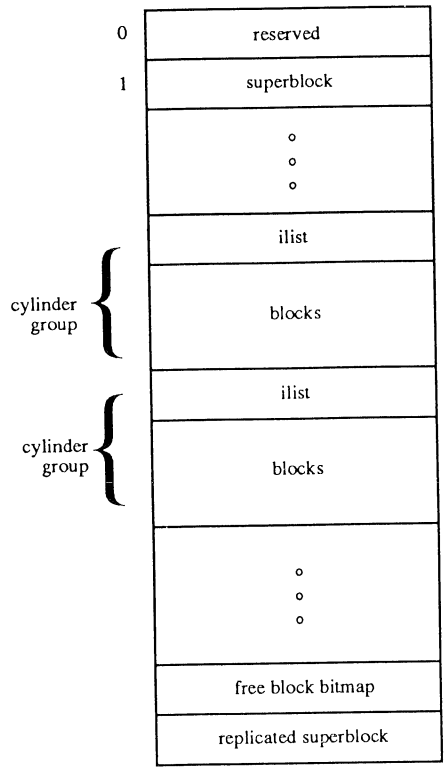


Figure 8-3. The IRIX View of a File System

Refer to the on-line system header files in */usr/include/sys/fs* for the C language data structure declarations of the file system structure itself, the superblock, the structure of an on-disk inode, and the structure of a directory block.

8.1.1 Block 0

Block 0, although considered to be part of the file system, is actually not used by it. It is reserved for storing booting procedures. Not all file systems are involved in booting. For a file system that is not, block 0 is left unused.

8.1.2 Block 1: the Super-Block

The information about the file system is stored in the super-block, including such things as:

file system size and status

- label, file system name
- size in physical and logical blocks
- read-only flag
- super-block modified flag
- date and time of last update

i-nodes

- total number of i-nodes allocated
- number of i-nodes free

storage blocks

- total number of free blocks
- starting block number of free block bitmap

A diagram of the fields in the super-block is shown in Figure 8-4. See the header information in */usr/include/sys/fs/efs_sb.h* for the C language data structure declarations of the superblock structure.

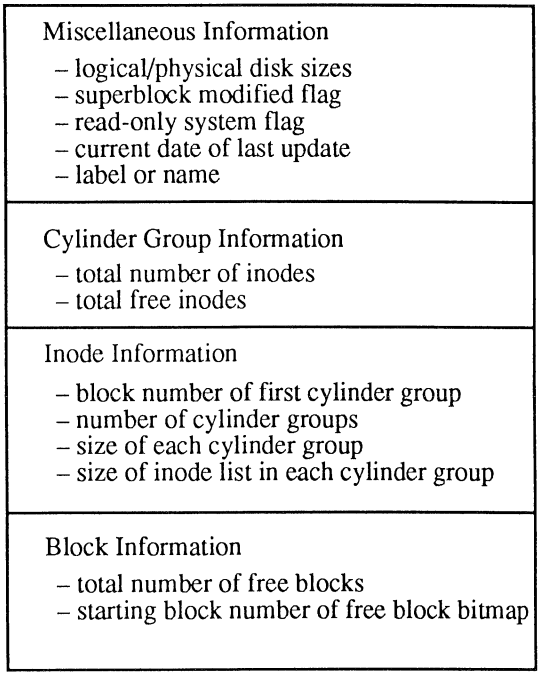


Figure 8-4. The Super-Block

8.1.3 I-Nodes

The term "i-node" stands for information node. (You will often see it spelled with no hyphen: inode.) The same formulation is used in other references to things associated with i-nodes. For example, the list of i-nodes is referred to as the i-list (or ilist); an i-number is the position of an i-node in the i-list.

The i-node contains all the information about a file except for its name, which is kept in a directory. An i-node is 128 bytes long, so there are 4 i-nodes to a physical block. The superblock gives the number of blocks reserved for inodes at the start of each cylinder group.

An i-node contains:

- the type and mode of file: type is regular (–); directory (d); block (b); character (c); FIFO, also known as named pipe, (p); symbolic link (l); or UNIX domain socket (s); the mode is the set of read-write-execute permissions
- the number of hard links to the file
- the owner's user-id number
- the group-id number to which the file belongs
- the number of bytes in the file
- an array of up to 12 disk block addresses of "extents"
- the number of extents
- the date and time last accessed
- the date and time last modified
- the date and time created

An *extent* is a variable length set of contiguous data blocks. An inode describes up to 12 extents with a starting disk block number and length. If the number of extents is larger than 12, then the extents described directly in the inode point to blocks which themselves in turn contain information about the "real" extents. These are known as indirect extents. A single extent may be up to 240 blocks in size. An optimally contiguous file can therefore be up to 1,474,560 bytes long before having to use indirect extents. The theoretical maximum file size is one where all 72 indirect extents contain 240 blocks worth of 8 byte pointers to data extents each of 240 blocks. The theoretical largest file is therefore over 20 gigabytes in size. For other reasons, however, the maximum file size is 2 gigabytes.

8.1.4 Storage Blocks

The remainder of the space allocated to the file system is taken up by storage blocks, also called data blocks. For a regular file, the storage blocks contain the contents of the file. The contents are undefined. For a directory, each block is of a specific structure. Each entry represents a file or subdirectory that is a member of the directory. An entry consists of the i-number and the filename of the member file or subdirectory.

8.1.5 Free Blocks

There is a free block bitmap which describes which filesystem data blocks are free. Each block is represented by a corresponding bit in this bitmap. If the bit is set, the block is unallocated.

8.2 How the File System Works

What we have discussed so far has been the organization of a IRIX file system on paper. We now want to describe what the IRIX system does with a file system when it is being used.

8.2.1 Tables in Memory

When a file system is identified to the IRIX system through a *mount*(1M) command, an entry is made in the mount table, and the super-block is read into an internal buffer maintained by the kernel. Disk inodes and free storage bitmaps are read in from the disk as needed.

The System I-Node Table

The IRIX system maintains a structure known as the system i-node table. Whenever a file is opened its i-node is copied from the secondary storage disk into the system i-node table. If two or more processes have the same file open, they share the same i-node table entry. The entry includes, among other things:

- the name of the device from which the i-node was copied
- the i-node number or i-number
- a reference count of the number of pointers to this entry (A file can be open for more than one process.)

A diagram of the system i-node table is shown in Figure 8-5.

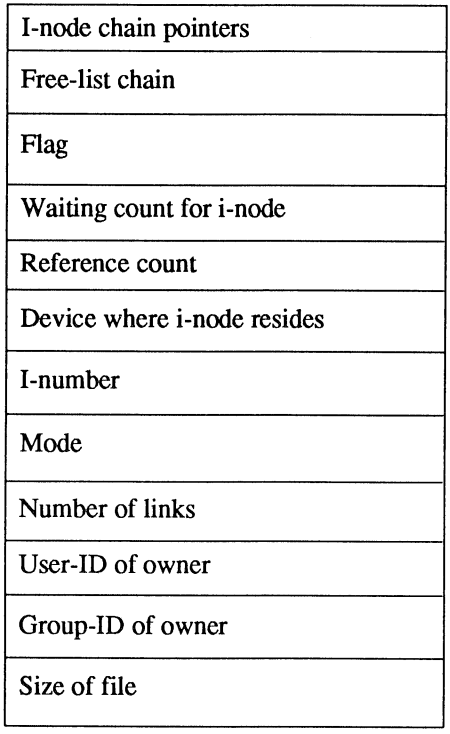


Figure 8-5. The System I-Node Table

The System File Table

The system maintains another table called the system file table. Because files may be shared among related processes a table is needed to keep track of which files are accessible by which process. For each file descriptor, an entry in the system file table contains:

- a flag to tell how the file was opened (read/write)
- a count of the processes pointing to this entry
  (When the count drops to zero, the system drops the entry.)
- a pointer to the system i-node table
- a pointer that tells where in the file the next I/O operation will take place

The Open File Table

The last table that is used to provide access to files is the open file table. It is located in the user area portion of memory. There is a user area for each process and, consequently, an open file table for each process. An entry in the open file table contains a pointer to the appropriate system file table entry. Figure 8-6 shows how these tables point to each other.

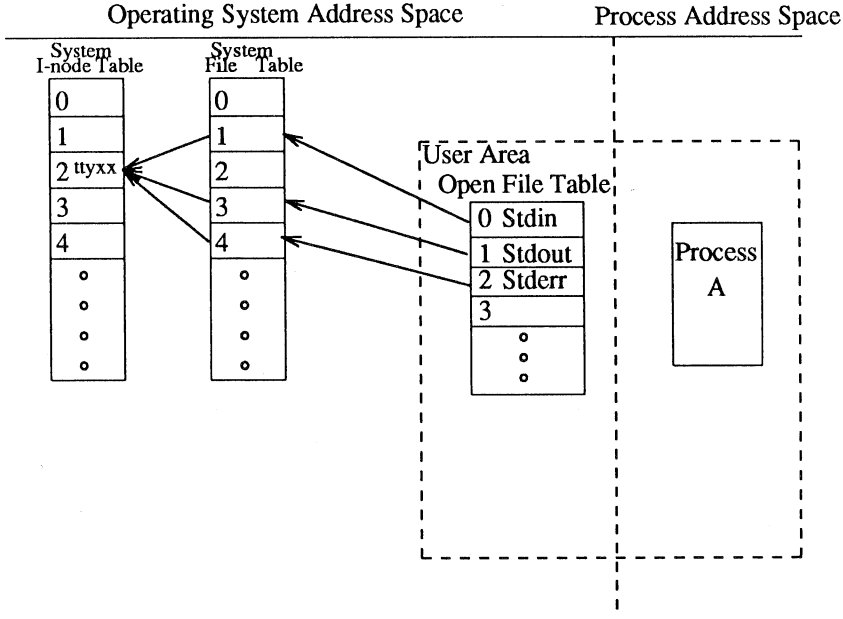


Figure 8-6. File System Tables and Their Pointers

8.2.2 System Steps in Accessing a File

In the next few paragraphs we describe steps followed by the operating system in opening, creating, reading, or writing a file.

Open

Suppose we give the pathname */a/b* to the *open*(2) system call. (Our program probably uses the *fopen*(3) subroutine from the standard I/O library, but that in turn invokes the system call.)

1. The operating system sees that the pathname starts with a slash, so the root i-node is obtained from the i-node table.
2. Using the root i-node, the system does a linear scan of the root directory file looking for an entry "a". When "a" is found, the operating system picks up the i-number associated with "a".
3. The i-number gives the offset into the i-node list at which the i-node for "a" is located. At that location, the system determines that "a" is a directory.
4. Directory "a" is searched linearly until an entry "b" is found.
5. When "b" is found, its i-number is picked up and used as an index into the i-list to find the i-node for "b".
6. The i-node for "b" is determined to be a file and is copied to the system i-node table (assuming it's not already there), and the reference count is incremented.
7. The system file table entry is allocated, the pointer to the system i-node table is set, the offset for the I/O pointer is set to zero to indicate the beginning of the file, and the reference count is initialized.
8. The user area file descriptor table entry is allocated with a pointer set to the entry in the system file table.
9. The number of the file descriptor slot is returned to the program.

The linear scan algorithm for locating the i-node of a file illustrates why it is advisable to keep directories small.

Create

Creating a file (the *creat*(2) system call) has these additional steps at the beginning:

1. The super-block is referenced for a free i-node number.
2. The mode of the file is established (possibly *and*-ed with the complement of a *umask* entry; see *umask*(2)) and entered in the i-node.
3. Using the i-number, the system goes through a directory search similar to that used in the *open* system call. The difference is that in the case of *creat* the system writes the last portion of the pathname into the directory that is the next to last portion of the pathname. The i-number is stored with it.

Reading and Writing

Both the *read*(2) and *write*(2) system call follow these steps:

1. Using the file descriptor supplied with the call as an index, the user's open file table is read and the pointer to the system file table obtained.
2. The user buffer address and number of bytes to read(write) are supplied as arguments to the call. The correct offset into the file is read from the system file table entry.
3. (Reading) The i-node is found by following the pointer from the system file table entry to the system i-node table. The operating system copies the data from storage to the user's buffer.
4. (Writing) The same pointer chain is followed, but the system writes into the data blocks. If new blocks are needed, they are allocated from the file system's list of free blocks. The EFS file system always attempts to grow the last extent if the following blocks are free, or to preallocate a large number of contiguous free blocks when allocating through a new extent. Disk blocks that aren't needed are freed when the file is closed.
5. Before the system call returns to the user, the number of bytes read(written) is added to the offset in the system file table.
6. The number of bytes read or written is returned to the user.

Files Used by More Than One Process

If related processes are sharing a file descriptor (as happens after a *fork*(1)) they also share the same entry in the system file table. Unrelated processes that access the same file have separate entries in the system file table, because they may be reading from or writing to different places in the file. In both cases the entry in the i-node table is shared; the correct offset at which the read or write should take place is tracked by the offset entry in the system file table.

Pathname Conversion

The directory search and pathname conversion takes place only once as long as the file remains open. For subsequent access of the file the system supplies a file descriptor that is an index into the open file table in your user process area. The open file table points to the system file table entry where the pointer to the system i-node table is picked up. Given the i-node, the system can find the data blocks that make up the file.

A running process will successfully read and write a file after it has opened it, even if someone has since renamed or unlinked it. This might be important when it seems that there are more used blocks than that accounted for in lookin just at files through the directory hierarchy. In this case, the blocks will be freed when the process *closes*(2) the file or *exits*(2).

8.2.3 Synchronization

The above description, while complex, may seem rather neat and orderly. The situation is complicated, however, by the fact that the IRIX system is a multi-tasking system. To give some tasks prompt attention, the system may make the decision that other tasks are less urgent. In addition, the system keeps a buffer cache and a cache of free blocks and i-nodes in memory together with the super-block to provide more responsive service to users. The stability that comes from having every byte of data in a file immediately written to the storage disk is traded for the gain of being able to provide more service to more users.

In normal processing, disk buffers are flushed periodically to the disk devices. This is a system process that is not related directly to any reads or writes of user processes. The process is called "synchronization." It includes writing out the super-blocks in addition to the disk buffers. The *sync* command can be used to cause the writing of super-blocks and updated i-nodes and the flushing of buffers. It is worth noting, however, that the return from the command simply means that the writing was scheduled, not necessarily completed.

Processes that must ensure that data is written to the disk immediately can open the file with the *O_SYNC* flag, which causes the data and inode information to be written to the disk at the time of the write system call. Alternatively, the *fsync*(2) system call will flush all data to disk associated with the particular file descriptor argument.

8.2.4 Search Time

There are several things that have a bearing on the amount of time the system needs to spend in looking for and reading in a file:

- the size of the directories being searched
- the size of the file itself
- The layout of the file extents

As described above, when the IRIX system is locating a file to be opened it searches linearly through all the directories in the pathname. Search time can be reduced by keeping the number of entries in a directory small.

8.2.5 Summary

We have tried in this section to give you an understanding of how the IRIX operating system controls file systems. Seeing how things are supposed to work can give us an appreciation of the steps that must be taken to keep a file system consistent.

8.3 Administering the File System

8.3.1 Creating a File System and Making it Available

Once a disk is formatted the next step is to define the file system. The *mkfs*(1M) and *growfs*(1M) commands are used for this purpose.

8.3.2 Using *mkfs*

The *mkfs* command has two formats:

```
mkfs  [-g] [-i] [-r] special [proto]
mkfs [-g] [-i] [-r] special blocks inodes heads sectors cgsize
```

Notice that in neither format is the file system actually given a name; it is identified by the filename of the special device file on which it will reside. The special device file, traditionally located in the directory *dev*, is tied to the identifying controller and unit numbers (major and minor, respectively) for the physical device.

The second format allows you to specify the file system parameters other than the defaults. This can be used, for example, to dedicate blocks to file data which would otherwise be used for file inodes. Note that the number of inodes determines the maximum number of files in the system.

8.3.3 Using growfs

The *growfs* command expands an existing filesystem. Its format is:

/etc/growfs `[-s size] special`

The *special* argument should be the pathname of the device special file of the device where the existing filesystem resides. The filesystem must be unmounted to be grown, using the *umount*(1M) command. The existing contents of the filesystem are undisturbed, and the additional space becomes available for use.

If a *size* argument is given, the filesystem is grown to occupy *size* basic blocks of storage. Otherwise, the filesystem is grown to occupy all the space available on the device.

Filesystems normally occupy all of the space on the device where they reside. Therefore, you must provide additional space in which to grow the filesystem. Either you must have at least one spare disk partition available, or you can add more space by using logical volumes. In fact, *growfs* is expected to be used with logical volumes.

If the existing filesystem is already a part of a logical volume, the new partition should be added to that logical volume, by first editing */etc/lvtab*, then running *mklv*. (See Section 7.3 for more information on logical volumes and *mklv*.) If the existing filesystem is currently on a regular partition, a new logical volume must be created. The first member should be the existing filesystem, and subsequent members the partitions you wish to add. Then run *mklv* to set up the logical volume.

In either case, after the logical volume is updated or created, run *growfs* on the logical volume device. The expanded filesystem is then available for use, as a part of the logical volume.

8.3.4 Relating the File System Device to a File System Name

A IRIX file system is generally referred to by the name of the highest level directory in its hierarchy. The file system shown in Figure 8–2 at the beginning of this chapter is called *usr* because that's the directory it is tied to. Similarly, the root file system is called that because its first directory is "root" (represented in IRIX system parlance by a slash (/)). But we saw above that when the file system was created, the only name on the command line (other than the name of a prototype file, if you used that option) was the name of a special device file. There are a couple of ways in which the file system name and the directory name can be tied together.

The first, and most explicit, is through the *labelit*(1M) command. *labelit* makes the connection between the device special file and the mounted name of the file system. It writes the name of the file system, that is, its highest level directory, into a field in the super-block. When *labelit* is used for removable file systems, such as those on diskette, one command line argument can be the identifying number of a volume. This number, too, is stored in a field in the super-block, but it is common practice to write it on a self-adhesive label that is attached to the diskette or tape that holds the file system.

The connection between the device and the file system name is also made by the *mount*(1M) command. This step is mandatory if the file system is to be available to users.

8.3.5 Mounting and Unmounting File Systems

For a file system to be available to users, the IRIX system has to be told to "mount" it. The *root* file system is always mounted as part of the boot procedure. The *usr* file system, which may be the same as *root*, is also automatically mounted as the system is being brought up to multi-user mode. The issuing of the *mount* command that brings these two file systems on line is hidden in start-up shell procedures, which use the */etc/fstab* to determine how the filesystems should be mounted. Regardless of whether the *mount* command is hidden or not, its execution causes the file system to be listed in an internal IRIX system table (called the mount table, or */etc/mnttab*) paired with the directory that is specified. For example, the command

```
mount  /dev/dsk/ips0d1s6  /usr
```

tells the system that *dev/dsk/ips0d1s0* contains a file system that begins at directory *usr*.

Note: The *mount* command has other arguments. See the *IRIS-4D System Administrator's Reference Manual* for complete details.

If you try to change directories (*cd*(1)) to a directory in *usr* before the *mount* command is issued, the *cd* command will fail. Until the *mount* command completes, the system does not know about any of the directories beneath *usr*. True, there is a directory *usr* (it must exist at the time the *mount* command is issued), but the structure of files and directories below that remain hidden from the IRIX system until the *mount*.

Note that if you mount a filesystem over an existing directory, the original directory and its subtree will also be hidden from IRIX until the filesystem is unmounted.

Unmounting is frequently a first step before using other commands that operate on file systems. For example, *fsck*(1M), which checks and repairs a file system, works on unmounted file systems. Unmounting is also an important part of the process of shutting the system down.

8.3.6 Summary

Thus far we have looked at file systems in the abstract. We have seen something of the way they are created, stored on disks, made available to the system, or removed from the system. In the next portion of this chapter we will see how you maintain the integrity of an active file system.

8.4 Maintaining a File System

Once a file system is created and made available to users it is always necessary to monitor how it is being used by the people in the organization. We get into a somewhat fuzzy area here where the distinction between a file system and its files may result in some confusion. The administrator's view is more likely to be of the file system, while users tend to think and work in terms of files. When we begin to talk about the tasks involved in keeping file systems working smoothly for users we have to be ready to deal with users' individual files as well as with the entire system.

Once a file system has been created and made available, there are several tasks routinely done to make certain that the file systems in regular use on a workstation are providing the level of service and stability they should. They can be grouped into procedures for

- checking for file system consistency
- monitoring disk usage
- compressing and reorganizing file systems
- backing up and restoring file systems

8.4.1 The Need for Policies

As with most other aspects of administration, file system administration should be based on establishing a set of policies that are appropriate for your organization. There can be no hard-and-fast rules for such things as the size of file systems, the number of users in a file system, the way in which backups are done, the extent to which users can be allowed to keep inactive files in the system, or the amount of disk space a single user is entitled to occupy. These questions can only be resolved within the context of the organization. The number of users, the type of work they are doing, the number of files needed—all are variables. The responsible administrator must determine what best meets the needs of the organization.

8.4.2 Shell Scripts for File System Administration

Once policies have been agreed on, many of the routine tasks connected with file system administration can be incorporated in shell scripts. Monitoring disk usage, for example, can be handled through shell scripts that do the monitoring for you and transmit messages to the system console when exceptions are detected. Here are a few ideas:

- Use a shell script running under *cron*(1M) control to investigate free blocks and free i-nodes and to report on file systems that fall below a given threshold.
- Use a shell script to do automatic clean-ups of files that grow.
- Use a shell script to highlight cases of excessive use of disk space.

8.4.3 Checking for File System Consistency

There is a separate section later in this chapter that describes *fsck*(1M), the file system checking utility. However, we want to include this mention of it here because file system checking is central to the whole problem of normal file system maintenance.

8.4.4 Monitoring Disk Usage

You need to monitor the level of usage of a file system for the following reasons.

- If not watched regularly, the percentage of disk space used increases until the allocated space is used up.
- When the allocated space is used up processes run very slowly, or not at all, the system spends its time putting out a message about being out of file space.
- There is a natural tendency for users to forget about files they no longer use so that the files just sit there taking up space.

- Some files grow larger as a result of perfectly normal use of the system. It is an administrative responsibility to keep them under control.
- Some directories, notably */tmp*, accumulate files during the day. When the system is first brought up, */tmp* needs to have enough free blocks to carry it through to *shutdown*(1M).

8.4.5 Monitoring Percent of Disk Space Used

Monitoring disk space may be done at any time to see how close to capacity your system is running. Until a pattern has emerged, it is advisable to check every day. In this example, the *df*(1M) command is used.

```
"df -k"
```

The *–k* option causes *df* to report useage in 1024-byte units, instead of the default 512-byte units. The *–i* option causes *df* to display the number and percentage of used inodes, as well as blocks. When no file systems are named, information about all mounted file systems is displayed, as in the example below:

```
df -k
Filesystem     Type  kbytes     use    avail %use  Mounted on
/dev/root       efs   15300    9676     5624  63%  /
/dev/usr        efs  231800  188425    43375  81%  /usr
/debug          dbg   66108   14224    51884  22%  /debug
```

8.4.6 Monitoring Files and Directories that Grow

Almost any system that is used daily has several files and directories that grow through normal use. Some examples are:

File	Use
/etc/wtmp	history of system logins
/usr/adm/sulog	history of *su* commands
/usr/lib/cron/log	history of actions of */etc/cron*
/usr/lib/spell/spellhist	words that *spell*(1) fails to match

The frequency with which you should check growing files depends on how active your system is and how critical the disk space problem is. A good technique for keeping them down to a reasonable size uses a combination of *tail*(1) and *mv*(1):

```
tail -50 /usr/adm/sulog > /tmp/sulog

mv /tmp/sulog /usr/adm/sulog
```

This sequence puts the last 50 lines of *usr/adm/sulog* into a temporary file, and then it moves the temporary file to *usr/adm/sulog*, thus effectively truncating the file to the 50 most recent entries.

8.4.7 Identifying and Removing Inactive Files

Part of the job of cleaning up heavily loaded file systems involves locating and removing files that have not been used recently. The commands you might use to do this work are shown below; the policy decisions involved are:

- how long should a file remain unused before it becomes a candidate for removal?
- should users be warned that old files are about to be purged?
- should the files be permanently removed or archived?

The *find*(1) command can locate files that have not been accessed recently. *find* searches a directory tree beginning at a point named on the command line. It looks for filenames that match a given set of expressions, and when a match is found, performs a specified action on the file. This example barely begins to suggest the full power of *find*.

```
find /usr -type f -mtime +60 -print > /tmp/deadfiles &
```

Here is what the example shows:

/usr	specifies the pathname where *find* is to start. Presumably, your machine is organized in such a way that inactive user files will not often be found in the *root* file system.
–type f	tells *find* to look only for regular files, and to ignore special files, directories, and pipes.
–mtime +60	says you are interested only in files that have not been modified in 60 days.
–print	means that when a file is found that matches the **–type** and **–mtime** expressions, you want the pathname to be printed.
> /tmp/deadfiles &	directs the output to a temporary file and indicates that the process is to run in the background. This is a sensible precaution if your experience tells you to expect a substantial amount of output.

The *sysadm fileage*(1) command can be used to produce similar information.

8.4.8 Identifying Large Space Users

Here again the most important questions are not what commands to use to learn who is occupying excessive amounts of disk space, but rather policy questions concerned with deciding what the limits should be. Policy questions include:

- On our system, what constitutes a reasonable amount of disk space for a user to need?
- If a user exceeds the normal amount by 25%, say, is it possible the user's job requires extraordinary amounts of disk space?
- Is our system as a whole running short of space? Do our existing limits need to be reviewed?

Two commands produce useful information in this area: *du*(1) and, once again, *find*(1).

du produces a summary of the block counts for files or directories named in the command line. For example:

```
du /usr
```

displays the block count for all directories in the *usr* file system. Optional arguments allow you to refine the output somewhat. For example, *du –s* may be run against each user's login to monitor individual users.

The *find* command can be used to locate specific files that exceed a given size limit.

```
find /usr -size +10 -print
```

This example produces a display of the pathnames of all files (and directories) in the *usr* file system that are larger than 10 (512-byte) blocks. Similar information can be produced by the *sysadm filesize*(1) command.

8.5 What Can Go Wrong With a File System

Most of the things that can corrupt a file system have to do with the failure of the address and count information to make it out to the storage medium. This can be caused by

- hardware failure
- program interrupts
- human error

or a combination of hardware/program failures and incorrect procedures.

8.5.1 Hardware Failure

There is no very effective way of predicting when hardware failure will occur. The best way of dealing with it is to be sure that recommended diagnostic and maintenance procedures are followed conscientiously. There is a utility that flags bad blocks on hard disk and uses a substitute area for blocks the system attempts to write to a flagged block, see Chapter 12, "The fx Disk Utility"

8.5.2 Program Interrupts

It is possible that errors that cause a program to fail might result in the loss of some data. It is not easy to generalize about this because the range of possibilities is so large. Perhaps the best thing to be said is that programs should be exhaustively tested before they are put into production with valuable data.

8.5.3 Human Error

While it may be painful to admit it, probably the greatest cause of file system corruption falls under this heading. We are going to recommend here some rules that should be followed by anyone who manages file systems.

1. NEVER remove a file system physically without first unmounting it.
2. ALWAYS use the *sync* command before shutting the system down or powering it off. Preferably, unmount the file system first.
3. NEVER physically write-protect a mounted file system, unless it is mounted "read only."

The random nature of all these mishaps simply underscores the importance of establishing and observing good backup practices. It is the most effective form of insurance against data loss.

8.6 Checking a File System for Consistency

When the IRIX operating system is brought up, a consistency check of the file systems should always be done. This check is automatically done as part of the power-up process for file systems that are automatically mounted. Included as part of that process is the command *fsstat*(1M). *fsstat* returns a code for each file system on the hard disk indicating whether the consistency checking and repair program, *fsck*(1M), should be run.

These same commands, or *sysadm checkfsys*(1), should be used to check file systems not mounted routinely as part of the power-up process. File systems that were not unmounted properly (via the *umount*(1) command) cannot be mounted until *fsck* is run on them. If inconsistencies are discovered, corrective action must be taken before the file systems are mounted. The remainder of this section is designed to acquaint you with the command line options of the *fsck* utility, the type of checking it does in each of its phases, and the repairs it suggests.

It should be said at the outset that file system corruption, while serious, is not all that common. Most of the time a check of the file systems finds everything all right. The reason we put so much emphasis on file system checking is that if errors are allowed to go undetected, the ultimate loss can be substantial.

8.6.1 The *fsck* Utility

The file system check (*fsck*) utility is an interactive file system check and repair program. *fsck* uses the information carried in the file system itself to perform consistency checks. If an inconsistency is detected, a message describing the inconsistency is displayed. You may elect to have *fsck* either make the repair or not. The reason you might choose to have *fsck* ignore an inconsistency is that you plan to go back to an earlier version of the file system. The decision to have *fsck* ignore inconsistencies and then do nothing about them yourself is not a viable one. File system inconsistencies do not repair themselves. If they are ignored, they only get worse.

8.6.2 The *fsck* Command

The *fsck* command is used to check and repair inconsistencies in a file system. With the exception of the *root* file system, a file system should be unmounted while it is being checked. The *root* file system should be checked only when the computer is in run level S and no other activity is taking place in the machine.

The following is the general format of the *fsck* command:

fsck [**-y**] [**-n**] [**-t***file*] [**-q**] [**-D**] [**-f**] [*fsdevice*]

The raw device name should be used for all filesystems except the *root* filesystem. Use the block device name for *root*.

The options of the *fsck* command are as follows:

-y	Specifies a "yes" response for all questions. This is the normal choice when the command is being run as part of a shell procedure. It generally causes *fsck* to correct all errors.
-n	Specifies a "no" response for all questions. *fsck* will not write the file system.
-t*file*	Specifies a scratch file for use in case the file system check requires additional memory. If this option is not specified, the process asks for a filename when more memory is needed.
-q	Specifies a "quiet" file system check. Output messages from the process are suppressed.
-D	Checks directories for bad blocks. This option is used to check file systems for damage after a system crash.
-f	Specifies that a fast file system check be done. Only Phase 1 (check blocks and sizes) and Phase 5 (check free list) are executed for a fast check. Phase 6 (reconstruct free list) is run only if necessary.
fsdevice	Names the special device file associated with a file system.

8.6.3 The *dfsck* Command

The *dfsck* command allows two filesystem checks on two different drives simultaneously. *dfsck* should *not* be used to check the *root* filesystem. Its format is as follows:

```
/etc/dfsck [options1] fsys1 ... - [options2] fsys2
```

options1 and *options2* are used to pass options to *fsck* for the two sets of filesystems. The – is the separator between the filesystem groups.

The *dfsck* program permits you to interact with two *fsck* programs at once. To distinguish between the two, *dfsck* prints the filesystem name for each message to the user. When answering a question from *dfsck*, you must preface your response with a 1 or a 2, indicating that the answer refers to the first or second filesystem group.

8.6.4 Sample Command Use

This command line shows *fsck* being entered to check the *root* file system.

```
fsck /dev/root
```

No options are specified. The system response means that no inconsistencies were detected. The command operates in phases, some of which are run only if required or in response to a command line option. As each phase is completed, a message is displayed. At the end of the program a summary message is displayed showing the number of files (i-nodes), blocks, and free blocks.

```
/dev/root
File System: root Volume: root

** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Free List
2138 files 20982 blocks 10068 free
```

8.6.5 File System Components Checked by *fsck*

Before getting into a discussion of the *fsck* phases and the messages that may appear in each, it is important to review the components of a IRIX file system and to describe the kinds of consistency checks that are applied to them.

Super-Block

The super-block is vulnerable because every change to the file system blocks or i-nodes modifies the super-block. If the CPU is halted, and the last command involving output to the file system is not a *sync* command, the super-block is almost certainly corrupted. In IRIX 3.3, a replicated super-block has been added to the EFS filesystem, situated at the end of the filesystem. If *fsck* cannot read the primary super-block, it will attempt to use the replicated super-block, printing a message to notify you of the

situation. This behavior is automatic; no user intervention is required. *fsck* determines if a replicated superblock exists. If not, it will optionally create one, thus allowing older file systems to benefit from this feature. If no superblock can be found on a damaged filesystem, you may be able to generate one using the –r option of *mkfs*(1M).

The super-block can be checked for inconsistencies involving:

- File system size
- I-node list size
- Free-block bitmap
- Free-block count
- Free i-node count

File System Size and I-Node List Size

Total file system size must be greater than the number of blocks used by the super-block plus the blocks used by the list of i-nodes. While there is no way to check these sizes, *fsck* can check that they are within reasonable bounds. All other checks of the file system depend on the reasonableness of these values.

Free-Block Bitmap

A field in the superblock gives the starting block number of the free block bitmap. A free block should have a corresponding 1 bit in the bitmap. An allocated block should have a corresponding 0.

A check is made to see that all the blocks in the file system were found.

When all the blocks have been accounted for, a check is made to see if the number of blocks in the free-block bitmap plus the number of blocks claimed by the i-nodes equals the total number of blocks in the file system. If anything is wrong with the free-block bitmap, *fsck* can rebuild it leaving out blocks already allocated.

Free-Block Count

The super-block contains a count of the total number of free blocks within the file system. The *fsck* program compares this count to the number of blocks it found free within the file system. If the counts do not agree, *fsck* can replace the count in the super-block by the actual free-block count.

Free I-Node Count

The super-block contains a count of the number of free i-nodes within the file system. The *fsck* program compares this count to the number of i-nodes it found free within the file system. If the counts do not agree, *fsck* can replace the count in the super-block by the actual free i-node count.

I-Nodes

The list of i-nodes is checked sequentially starting with i-node 2 (there is no i-node 0 or 1). Each i-node is checked for inconsistencies involving

- Format and type
- Link count
- Duplicate blocks
- Bad block numbers for extents
- I-node size

Format and Type

Each i-node contains a mode word. This mode word describes the type and state of the i-node. I-nodes may be one of seven types:

- Regular
- Directory
- Block special
- Character special
- Fifo (named pipe)
- Symbolic link
- UNIX domain socket

If an i-node is not one of these types, it is illegal. I-nodes may be in one of three states: unallocated, allocated, and partially allocated. This last state means an incorrectly formatted i-node. An i-node can reach this state if, for example, bad data are written into the i-node list through a hardware failure. The only corrective action *fsck* can take is to clear the i-node.

Link Count

Each i-node contains a count of the number of directory entries linked to it. The *fsck* program verifies the link count of each i-node by examining the total directory structure, starting from the root directory, and calculating an actual link count for each i-node.

If the link count stored in the i-node and the actual link count determined by *fsck* do not agree, the reason may be

- stored count not 0, actual count 0

  No directory entry appears for the i-node.
  fsck can link the disconnected file to the lost+found directory.

- stored count not 0, actual count not 0, counts unequal

  Directory entry possibly removed without i-node update.
  fsck can replace the stored link count with the actual link count.

Duplicate Blocks

Each i-node contains a list of the extents of blocks claimed by the i-node. The *fsck* program compares each block number claimed by an i-node to a list of allocated blocks. If a block number claimed by an i-node is on the allocated-blocks list, it is put on a duplicate-blocks list. If the block number is not on the allocated-blocks list, it is put there. If a duplicate-blocks list develops, *fsck* makes a second pass of the i-node list to find the other i-node that claims the duplicated block. While there is not enough information available to determine which i-node is in error, most of the time the i-node with the latest modification time is correct. This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

Bad Block Numbers

The *fsck* program checks each block number claimed by an i-node 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, the block number is bad.

Note: A certain amount of semantic confusion is possible here. A bad block number in a file system is not the same as a bad (that is, unreadable) block on a hard disk.

I-Node Size

Each i-node contains a 32-bit (4-byte) size field. This size shows the number of characters in the file associated with the i-node.

For a regular file, a rough check of the consistency of the size field of an i-node can be performed by using the number of characters shown in the size field to calculate how many blocks should be associated with the i-node, and comparing that to the actual number of blocks claimed by the i-node.

The Algorithm

The *fsck* program calculates the number of blocks that should be in a file by dividing the number of characters in an i-node by 512 (the number of characters per block) and rounding up.

If the actual number of blocks does not match the computed number of blocks, *fsck* warns of a possible file-size error. This is only a warning. A check of the file would be required to tell if the error is real or not.

Indirect Extents

Indirect extents are owned by an i-node. Therefore, inconsistencies in an indirect extent directly affect the i-node that owns it. Inconsistencies that can be checked are:

- Blocks already claimed by another i-node
- Block numbers outside the range of the file system

The consistency checks described under "Duplicate Blocks" and "Bad Block Numbers" above are performed for indirect blocks as well as for the direct blocks of an i-node.

Directory Data Blocks

Directories are distinguished from regular files by an entry in the mode field of the i-node. Data blocks associated with a directory contain the directory entries. Directory data blocks are checked for inconsistencies involving:

- directory i-node numbers pointing to unallocated i-nodes
- directory i-node numbers greater than the number of i-nodes in the file system
- incorrect directory i-node numbers for "." and ".." directories
- directories disconnected from the file system

Directory Unallocated

If a directory entry i-node number points to an unallocated i-node, *fsck* can remove that directory entry. This condition occurs if the data blocks containing the directory entries are modified and written out while the i-node is not yet written out.

Bad I-Node Number

If a directory entry i-node number is pointing beyond the end of the i-node list, *fsck* can remove that directory entry. This condition occurs if bad data are written into a directory data block.

Incorrect "." and ".." Entries

The directory i-node number entry for "." should be the first entry in the directory data block. Its value should be equal to the i-node number for the directory data block.

The directory i-node number entry for ".." should be the second entry in the directory data block. Its value should be equal to the i-node number for the parent of the directory entry (or the i-node number of the directory data block if the directory is the root directory). If the directory i-node numbers for "." and ".." are incorrect, *fsck* can replace them with the correct values.

Disconnected Directories

The *fsck* program checks the general connectivity of the file system. If directories are found not to be linked into the file system, *fsck* links the directory back into the file system in the *lost+found* directory. This condition can be caused by i-nodes being written to the file system with the corresponding directory data blocks not being written to the file system. When a file is linked into the *lost+found* directory, the owner of the file needs to be told about it.

Regular Data Blocks

Data blocks associated with a regular file hold the file's contents. *fsck* does not attempt to check the validity of the contents of a regular file's data blocks.

8.6.6 Running fsck

The *fsck* program runs in phases. Each phase reports errors it detects. If an error is one that *fsck* can correct, the user is asked if the correction should be made. This section describes the messages that are produced by each phase.

The following abbreviations are used in the *fsck* error messages:

BLK	block number
DUP	duplicate block number
DIR	directory name
MTIME	time file was last modified
UNREF	unreferenced

The following single-letter abbreviations, used in the messages shown in the pages that follow, are replaced by the corresponding value when the message appears on your screen:

B	block number
F	file (or directory) name
I	i-node number
M	file mode
O	user-id of a file's owner
S	file size
T	time file was last modified
X	link count,
or	number of BAD, DUP, or MISSING blocks
or	number of files (depending on context)
Y	corrected link count number
or	number of blocks in file system (depending on context)
Z	number of free blocks

Figure 8-7. Error Message Abbreviations in *fsck*

Initialization Phase

Command line syntax is checked. Before the file system check can be performed, *fsck* sets up some tables and opens some files. The *fsck* program terminates on initialization errors.

General Errors

Three error messages may appear in any phase. While they seem to offer the option to continue, it is generally best to regard them as fatal, end the run, and investigate what may have caused the problem.

CAN NOT SEEK: BLK *B* (CONTINUE?)
: The request to move to a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

CAN NOT READ: BLK *B* (CONTINUE?)
: The request for reading a specified block number *B* in the file system failed. The occurrence of this error condition indicates a serious problem (probably a hardware failure) that may require additional help.

CAN NOT WRITE: BLK *B* (CONTINUE?)
: The request for writing a specified block number *B* in the file system failed. The disk may be write-protected.

Meaning of Yes/No Responses

An n(no) response to the CONTINUE? prompt says:

Terminate program.
: (This is the recommended response.)

A y(yes) response to the CONTINUE? prompt says:

Attempt to continue to run file system check.

Often, however, the problem persists. The error condition does not allow a complete check of the file system. A second run of *fsck* should be made to recheck this file system.

Phase 1: Check Blocks and Sizes

This phase checks the i-node list. It reports error conditions resulting from:

- checking i-node types
- setting up the zero-link-count table
- examining i-node block numbers for bad or duplicate blocks
- checking i-node size
- checking i-node format

Types of Error Messages—Phase 1

Phase 1 has three types of error messages:

1. information messages
2. messages with a CONTINUE? prompt
3. messages with a CLEAR? prompt

There is a connection between some information messages and messages with a CONTINUE? prompt. The meaning of the CONTINUE? prompt generally is that some limit of tolerance has been reached.

Meaning of Yes/No Responses—Phase 1

In Phase 1, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 1, a y(yes) response to the CONTINUE? prompt says:

Continue with the program.
This error condition means that a complete check of the file system is not possible. A second run of *fsck* should be made to recheck this file system.

In Phase 1, an n(no) response to the CLEAR? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 1, a y(yes) response to the CLEAR? prompt says:

Deallocate i-node *I* by zeroing its contents.
This may invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this i-node.

Phase 1 Error Messages

UNKNOWN FILE TYPE I=*I* (CLEAR?)
The mode word of the i-node *I* suggests that the i-node is not a pipe, special character i-node, regular i-node, directory i-node, symbolic link, or socket.

LINK COUNT TABLE OVERFLOW (CONTINUE?)
An internal table for *fsck* containing allocated i-nodes with a link count of zero has no more room.

B BAD I=*I*
I-node *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 tne last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if i-node *I* has too many block numbers outside the file system range. This error condition invokes the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLOCKS I=*I* (CONTINUE?)

There is more than a tolerable number (usually 50) of blocks 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 i-node *I*.

B DUP I=*I*

I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if i-node *I* has too many block numbers claimed by other i-nodes. This error condition invokes Phase 1B and the BAD/DUP error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=*I* (CONTINUE?)

There is more than a tolerable number (usually 50) of blocks claimed by other i-nodes.

DUP TABLE OVERFLOW (CONTINUE?)

An internal table in *fsck* containing duplicate block numbers has no more room.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR?)

I-node *I* is neither allocated nor unallocated.

RIDICULOUS NUMBER OF EXTENTS (*%d*) (max allowed *%d*)

The number of extents is larger than the maximum the system can set and is therefore ridiculous.

ILLEGAL NUMBER OF INDIRECT EXTENTS (*%d*)

The number of extents of pointers to extents (indirect extents) exceeds the number of slots in the inode for describing extents.

BAD MAGIC IN EXTENT

The ''pointer'' to an extent contains a ''magic number''. If this number is invalid, the pointer to the extent is probably corrupt.

EXTENT OUT OF ORDER

An extent's idea of where it is in the file is inconsistent with the extent pointer in relation to other extent pointers.

ZERO LENGTH EXTENT
An extent is at zero length.

ZERO SIZE DIRECTORY
It is erroneous for a directory inode to claim a size of zero. The corresponding inode is cleared.

DIRECTORY SIZE ERROR
A directory's size must be an integer number of blocks. The size is recomputed based on its extents.

DIRECTORY EXTENTS CORRUPTED
If the computation of size (above) fails, *fsck* will print this message and ask to clear the inode.

NUMBER OF EXTENTS TOO LARGE
The number of extents of pointers to extents (indirect extents) exceeds the number of slots in the inode for describing extents.

POSSIBLE DIRECTORY SIZE ERROR
The number of blocks in the directory computed from extent pointer lengths is inconsistent with the number computer from the inode size field.

POSSIBLE FILE SIZE ERROR
The number of blocks in the file computed from extent pointer lengths is inconsistent with the number computer from the inode size field. *fsck* gives the option of clearing the inode in this case.

Phase 1B: Rescan for More DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the i-node that previously claimed that block. When the duplicate block is found, the following information message is printed:

B DUP I=*I*

I-node *I* contains block number *B*, which is already claimed by another i-node. This error condition invokes the BAD/DUP error condition in Phase 2. I-nodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1.

Phase 2: Check Path Names

This phase removes directory entries pointing to bad i-nodes found in Phase 1 and Phase 1B. It reports error conditions resulting from

- Root i-node mode and status
- Directory i-node pointers out of range
- Directory entries pointing to bad i-nodes

Types of Error Messages—Phase 2

Phase 2 has 4 types of error messages:

1. Information messages
2. Messages with a FIX? prompt
3. Messages with a CONTINUE? prompt
4. Messages with a REMOVE? prompt

Meaning of Yes/No Responses—Phase 2

In Phase 2, an n(no) response to the FIX? prompt says:

Terminate the program since *fsck* will be unable to continue.

In Phase 2, a y(yes) response to the FIX? prompt says:

Change the root i-node type to "directory."
If the root i-node data blocks are not directory blocks, a very large number of error conditions are produced.

In Phase 2, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 2, a y(yes) response to the CONTINUE? prompt says:

Ignore DUPS/BAD error condition in root i-node and attempt to continue to run the file system check.
If root i-node is not correct, then this may result in a large number of other error conditions.

In Phase 2, an n(no) response to the REMOVE? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 2, a y(yes) response to the REMOVE? prompt says:

Remove duplicate or unallocated blocks.

Phase 2 Error Messages

ROOT INODE UNALLOCATED. TERMINATING
The root i-node (always i-node number 2) has no allocate mode bits. The occurrence of this error condition indicates a serious problem. The program stops.

ROOT INODE NOT DIRECTORY (FIX?)
The root i-node (usually i-node number 2) is not directory i-node type.

DUPS/BAD IN ROOT INODE (CONTINUE?)
Phase 1 or Phase 1B found duplicate blocks or bad blocks in the root i-node (usually i-node number 2) for the file system.

I OUT OF RANGE I=*I* NAME=*F* (REMOVE?)

A directory entry *F* has an i-node number *I* that is greater than the end of the i-node list.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* NAME=*F* (REMOVE?)

A directory entry *F* has an i-node *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 was not specified, the entry is removed automatically if the i-node it points to is character size 0.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory entry *F*, directory i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file entry *F*, i-node *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and filename *F* are printed.

BAD BLK *B* IN DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T*

This message only occurs when the *–D* option is used. A bad block was found in DIR i-node *I*. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and embedded slashes in the name field. This error message means that the user should at a later time either remove the directory i-node if the entire block looks bad or change (or remove) those directory entries that look bad.

Phase 3: Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. It reports error conditions resulting from

- Unreferenced directories
- Missing or full *lost+found* directories

Types of Error Messages—Phase 3

Phase 3 has 2 types of error messages:

1. Information messages
2. Messages with a RECONNECT? prompt

Meaning of Yes/No Responses—Phase 3

In Phase 3, an n(no) response to the RECONNECT? prompt says:

Ignore the error condition.
This invokes the UNREF error condition in Phase 4.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 3, a y(yes) response to the RECONNECT? prompt says:

Reconnect directory i-node *I* to the file system in directory for lost files (usually *lost+found*).
This may invoke a *lost+found* error condition if there are problems connecting directory i-node *I* to *lost+found*. This invokes CONNECTED information message if link was successful.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT?)

The directory i-node *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 i-node *I* are printed. The *fsck* program forces the reconnection of a nonempty directory.

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 directory in *lost+found.* This invokes the UNREF error condition in Phase 4. Possible problem with access modes of *lost+found.*

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 a directory in *lost+found.* This invokes the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*

This is an advisory message indicating a directory i-node *I1* was successfully connected to the *lost+found* directory. The parent i-node *I2* of the directory i-node *I1* is replaced by the i-node number of the *lost+found* directory.

Phase 4: Check Reference Counts

This phase checks the link count information seen in Phases 2 and 3. It reports error conditions resulting from:

- Unreferenced files
- 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
- Incorrect total free-i-node counts

Types of Error Messages—Phase 4

Phase 4 has 5 types of error messages:

1. Information messages
2. Messages with a RECONNECT? prompt
3. Messages with a CLEAR? prompt
4. Messages with an ADJUST? prompt
5. Messages with a FIX? prompt

Meaning of Yes/No Responses—Phase 4

In Phase 4, an n(no) response to the RECONNECT? prompt says:

Ignore this error condition.
This invokes a CLEAR error condition later in Phase 4.

In Phase 4, a y(yes) response to the RECONNECT? prompt says:

Reconnect i-node *I* to file system in the directory for lost files (usually *lost+found*).
This can cause a *lost+found* error condition in this phase if there are problems connecting i-node *I* to *lost+found*.

In Phase 4, an n(no) response to the CLEAR? prompt says:

Ignore the error condition.

A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the CLEAR? prompt says:

Deallocate the i-node by zeroing its contents.

In Phase 4, an n(no) response to the ADJUST? prompt says:

Ignore the error condition.

A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the ADJUST? prompt says:

Replace link count of file i-node *I* with *Y*.

In Phase 4, an n(no) response to the FIX? prompt says:

Ignore the error condition.

A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 4, a y(yes) response to the FIX? prompt says:

Replace count in super-block by actual count.

Phase 4 Error Messages

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT?)

I-node *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 i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty files are not cleared.

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 invokes the CLEAR error condition later in Phase 4. Possible problem with access modes of *lost+found*.

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 a file in *lost+found*. This invokes the CLEAR error condition later in Phase 4. Check size and contents of *lost+found*.

(CLEAR)

The i-node mentioned in the immediately previous UNREF error condition cannot be reconnected.

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)

The link count for i-node *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed.

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)

The link count for i-node *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory i-node *I* are printed.

LINK COUNT F I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST?)

The link count for *F* i-node *I* is *X* but should be *Y*. The filename *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)

I-node *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 i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty files are cleared automatically. Nonempty directories are not cleared.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)

I-node *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 i-node *I* are printed. If the **–n** option is omitted and the file system is not mounted, empty directories are cleared automatically. Nonempty directories are not cleared.

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with file i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR?)

Phase 1 or Phase 1B found duplicate blocks or bad blocks associated with directory i-node *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of i-node *I* are printed.

FREE INODE COUNT WRONG IN SUPERBLK (FIX?)

The actual count of the free i-nodes does not match the count in the super-block of the file system. If the **–q** option is specified, the count will be fixed automatically in the super-block.

Phase 5: Check Free List

This phase checks the free-block list. It reports error conditions resulting from

- Bad blocks in the free-block list
- Bad free-block count
- Duplicate blocks in the free-block list
- Unused blocks from the file system not in the free-block list
- Total free-block count incorrect

Types of Error Messages—Phase 5

Phase 5 has 4 types of error messages:

1. information messages
2. messages that have a CONTINUE? prompt
3. messages that have a FIX? prompt
4. messages that have a SALVAGE? prompt

Meaning of Yes/No Responses--Phase 5

In Phase 5, an n(no) response to the CONTINUE? prompt says:

Terminate the program.

In Phase 5, a y(yes) response to the CONTINUE? prompt says:

Ignore rest of the free-block list and continue execution of *fsck*.
This error condition will always invoke BAD BLKS IN FREE LIST error condition later in Phase 5.

In Phase 5, an n(no) response to the FIX? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 5, a y(yes) response to the FIX? prompt says:

Replace count in super-block by actual count.

In Phase 5, an n(no) response to the SALVAGE? prompt says:

Ignore the error condition.
A NO response is only appropriate if the user intends to take other measures to fix the problem.

In Phase 5, a y(yes) response to the SALVAGE? prompt says:

Replace actual free-block bitmap with a new free-block bitmap.

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE?)
The free-block list contains more than a tolerable number (usually 50) 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.

EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE?)
The free-block list contains more than a tolerable number (usually 50) of blocks claimed by i-nodes or earlier parts of the free-block list.

BAD FREEBLK COUNT
The count of free blocks is greater than the number of data blocks in the file system. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X BAD BLKS IN FREE LIST
X blocks in the free-block 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 later in Phase 5.

X DUP BLKS IN FREE LIST
X blocks claimed by i-nodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

X BLK(S) MISSING
X blocks unused by the file system were not found in the free-block list. This error condition will always invoke the BAD FREE LIST condition later in Phase 5.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX?)
The actual count of free blocks does not match the count in the super-block of the file system.

BAD FREE LIST (SALVAGE?)
This message is always preceded by one or more of the Phase 5 information messages. If the **–q** option is specified, the free-block list will be salvaged automatically.

Phase 6: Salvage Free List

This phase reconstructs the free-block bitmap. There are no error messages that can be generated in this phase.

Cleanup Phase

Once a file system has been checked, a few cleanup functions are performed. The cleanup phase displays advisory messages about the file system and status of the file system.

Cleanup Phase Messages

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.

SUPERBLOCK MARKED DIRTY
: A field in the superblock is queried by system utilties to decide if *fsck* must be run before mounting a file system. If this field is not "clean", fsck will report and ask if it should be cleaned.

PRIMARY SUPERBLOCK WAS INVALID
: If the primary superblock was too corrupt to use, and *fsck* could locate a secondary superblock, it will ask to replace the primary superblock with the backup.

SECONDARY SUPERBLOCK MISSING
: If there is no secondary superblock, and *fsck* finds space for one (after the last cylinder group), it will ask to create one.

CHECKSUM WRONG IN SUPERBLOCK
: An incorrect checksum will cause a file system to unmountable.

***** FILE SYSTEM WAS MODIFIED *****
: This is an advisory message indicating that the current file system was modified by *fsck*.

***** REMOUNTING ROOT... *****
: This is an advisory message indicating that *fsck* made changes to a mounted root file system. The automatic remount ensures that incore data structures and the file system are consistent.

8.7 Creating File Systems on Additional Disks

The first disk on your workstation already contains the *root* (*/*) and */usr* file systems. This procedure shows you how to define file systems on additional disks. Follow these steps:

1. To determine the type of disk in use on your system, use the *hinv* command. *hinv* displays the type (SCSI, ESDI, SMD, or IPI) of controllers that are present, as well as the number of drives per controller.

2. To proceed to the next steps, you need to know the disk device name and which partition(s) you plan to use to configure the disk. To help you understand disk device names, a brief discussion of disk naming conventions follows.

   You can add four types of disk to your IRIS: SCSI, ESDI, SMD, and IPI. Each type of disk has its own family of device names: **dks** for SCSI, **ips** for ESDI, **xyl** for SMD, and **ipi** for IPI.. Each family of device names has names for both raw and block devices. You use raw device names when you are accessing the physical disk; you use block device names when you are mounting file systems on a disk.

   The directory */dev/dsk* contains the block device names for disks; */dev/rdsk* contains the raw device names. The device names for disks have this form: `/dev/dsk/`*controller-type*`#d#s#` for block devices, and `/dev/rdsk/`*controller-type*`#d#s#` for raw devices.

   For example, */dev/dsk/ips0d1s0* refers to the ESDI device on controller 0, drive 1, and slice 0.

   The device name has these components:

   - The name of the directory that contains the disk names. There are two possibilities: *dev/dsk* contains block device names; *dev/rdsk* contains raw device names.
   - The controller type
   - The controller number (**0** in the example above).

- The disk number (**d1** in the example above). For ESDI and SMD disks, the disk number ranges between 0 and 3. By default, an ESDI or SMD root disk is controller 0, disk 0. For SCSI disks, the disk number ranges from 1 to 7. By default, a SCSI root disk is controller 0, disk 1. For IPI disks, the numbers range from 0 to 15. By default, an IPI root disk is controller 0, disk 0. (For more information, see *ipi*(7M), *xyl*(7M), *ips*(7M), and *dks*(7M).)
- The partition number (**s0** in the example above).

3. Decide whether to configure your disk as one or as several file systems.
4. Log in as *root*, or become the superuser with the *su* command.
5. Issue the *prtvtoc* command to find out the size of the partitions:

   ```
   prtvtoc raw_device_name
   ```

   For example, to find the partition layout for an ESDI root, type the command:

   ```
   prtvtoc /dev/rdsk/ips0d0s0
   ```

   (You can determine the device name of your system's root directory by typing the command `devnm /`; this returns the device name, not the raw device name.)

   For more information about the *prtvtoc* display, see *prtvtoc*(1M).
6. Decide which partition(s) you plan to use for mounting file systems.

   For example, suppose your IRIS has a SCSI root disk in the default configuration, and an additional SCSI disk. If you configure the additional disk as one file system, the device name is **dks0d2s7**. Another example is an IRIS with an ESDI root disk in the default configuration, and an additional ESDI disk. If you configure the additional disk as one file system, the device name is **ips0d1s7**.

Caution: On a standard disk, do not use partition 8, which contains volume header information, or partition 10 for mounting filesystems. On ESDI, SMD, and IPI disks, partition 9 also must not be used, as it contains the track replacement table. The largest partition, partition 10, represents the entire disk, including the header. These partitions contain valuable information which you must not overwrite. If you do, you will be unable to access information on your disk.

If you plan to configure your disk as several file systems, make sure to use partitions that do not overlap. If none of the default partitions suit your needs, you can re-partition the disk. See Chapter 7 for a detailed discussion.

The steps below tell you when to use raw device names; otherwise, use block device names.

7. Create a directory for each file system that you want to create on your disk. Although additional file systems are often mounted under *root* (*/*), you can mount them under any convenient directory. For example, if one user on your file system requires an entire disk, you can mount that disk under the directory */usr/people/<user_name>*.

   For example, to make a mount point named *moreusr*, type:

```
cd /
mkdir moreusr
```

8. Since the device names for the disk are long and cumbersome, you can link to simpler names. You can then refer to the disk by using the simpler names.

   Link the block and raw device names to the simpler device names you plan to use.

   Caution: When you issue *ln* commands, make sure that you type the device names correctly. Make sure that the device names you use address an additional disk and not the root disk. If you accidentally link the device name of a partition that already contains a file system, you will be unable to access information on that disk until you restore the proper device information. See *makedev*(7).

You must make links for both the block and raw device names for each file system you create; for example:

SCSI disk:

```
ln  /dev/dsk/dks0d2s7  /dev/moreusr
ln  /dev/rdsk/dks0d2s7  /dev/rmoreusr
```

ESDI disk:

```
ln  /dev/dsk/ips0d1s7  /dev/moreusr
ln  /dev/rdsk/ips0d1s7  /dev/rmoreusr
```

SMD disk:

```
ln  /dev/dsk/xyl0d1s7  /dev/moreusr
ln  /dev/rdsk/xyl0d1s7  /dev/rmoreusr
```

IPI disk:

```
ln  /dev/dsk/ipi0d1s7  /dev/moreusr
ln  /dev/rdsk/ipi0d1s7  /dev/rmoreusr
```

9. Make each new file system. Issue one *mkfs* command for each file system you want to make. To make the file system *moreusr*, issue this command:

```
mkfs  /dev/rmoreusr
```

mkfs reports some of the attributes of the file system and asks whether the file system is correct. If you don't interrupt, the system prompt appears after a few seconds.

10. Issue the *labelit* command to put the new file system name(s) into the superblock. Use raw devices. The *labelit* command has this form:

```
labelit  /dev/rmoreusr moreusr sgi
```

In this example, *moreusr* is the file system name and *sgi* is the volume name. The volume name is a way of labeling the disk. It can be up to six characters in length. Issue one *labelit* command for each file system you create. *labelit* reports the new file system and volume name, after which the system prompt appears.

11. Mount the new file system.

```
mount  /dev/moreusr  /moreusr
```

12. Optionally, edit the file *fstab* in the directory */etc*, adding a line for each file system. This line automatically mounts the new file system(s) when multi-user mode starts. */etc/fstab* is also a list of file systems that *fsck* checks as you enter multi-user mode. Entries in */etc/fstab* require both the block and raw device, in this format:

```
/dev/d  /d  efs  rw,raw=/dev/rd  0  0
```

13. Issue the *sync* command to flush the new information to the disk.

```
sync
```

8.7.1 Monitoring Disk Usage

This section describes utilities that allow you to make sure enough space is available on hard disk to accommodate users' needs. Follow these steps:

1. Enter the *sysadm diskuse* command. You see a display similar to this one:

```
FILE SYSTEM USAGE AS OF 08/28/87  13:23:30
File          Free        Total       Percent
System        Blocks      Blocks      Full
------        ------      ------      -------
/dev/root     11585        31815       64%
/dev/usr      20117       188650       89%
```

2. To determine which files are the oldest, use the *sysadm fileage* command. To determine which files are the largest, use the *sysadm filesize* command.

3. The *fileage* command prompts you for two pieces of information: the full pathname of the directory to search, and the number of days to go back. Be specific; if you select a high-level directory such as */usr*, the search might take a long time and you might get more information than

you want. The default number of days to go back is 90 days, which means that the system displays files in the directory you specify that have not changed since then.

4. *filesize* displays information on the *n* largest files (the default is 10) in a directory that you specify. For example, a list of the 10 largest files in */usr/etc* might produce a list like the following:

```
The 10 largest files in /usr/etc:

       file size   date of
owner (characters) last access  filename
----- ------------ ------------ --------
bin          486608 Aug 26  1989 dgld
root         176128 Apr 11 10:31 timed
root         151552 Aug 26  1989 timeslave
bin          122928 Sep  6  1989 rpc.bootparamd
bin          122880 Apr 18 15:41 bootp
bin          118832 Aug 26  1989 named
bin          118832 Apr  3 17:35 rpc.rstatd
bin          118784 Apr 11 10:30 routed
bin          114736 Dec 16 14:43 ftpd
bin          114736 Apr 11 10:30 route
```

9. Printer Use and Administration

The *lp* system allows information to be printed and is one of the main utilities that users need on a regular basis. This chapter discusses:

- The *lp* spooling system
- How to add and remove certain types of printers
- How to maintain the *lp* system
- How to troubleshoot *lp* system problems

9.1 Adding a Printer

To send documents to your printer, you must add your printer by registering it with the line printer (*lp*) spooler. This allows you to send print requests to the printer. The printer should be registered with the *lp* spooler of the computer to which the printer is directly connected (hardwired), and also with the *lp* spooler of any computer accessing the printer via a network. The procedures for registering a printer with *lp* are different depending on whether the printer is hardwired or accessed across a network; these procedures are described in the next sections. The last section describes how to remove a printer from the *lp* system.

9.2 Registering Hardwired Printers

To register printers connected directly to your computer, follow these steps:

1. Become the superuser.
2. Stop the print spooler:

   /usr/lib/lpshut

3. Assuming you have a printer attached to the parallel port, use the *mkcentpr*(1M) utility to install the printer in the *lp* system:

   mkcentpr *type printer-name*

   For *type*, enter `mits` if your printer is a Mitsubishi, `seiko` if it is a Seiko, `tek` if it is a Tektronix, or `vers` if it is a Versatec. *printer-name* must be fourteen characters or less and consist solely of alphanumerics and underscores. It must also be unique within the *lp* system.

4. To set up this printer as the default printer, type:

   /usr/lib/lpadmin -d*printer-name*

5. Restart the print spooler:

   /usr/lib/lpsched

Your printer is now registered with the *lp* system and is ready for printing.

9.3 Registering Network Printers

To configure a printer for use across a network, follow these steps:

1. If your computer does not use TCP/IP network protocols, skip to step 5.
2. Log in as the superuser to the workstation with the attached printer.
3. Replace *remote* in the command below with the name of the workstation that needs access to the printer. Type:

   addclient *remote*

   addclient grants permission for the specified *remote* workstation to access printers across the network. If you want all of the *remote* workstations to be able to use printers on your system, type:

   addclient -a

   Note: Printers must be configured on the system to which they are attached before remote workstations can configure them successfully across the network.

4. Both workstations must be able to communicate across the network. For additional information on network communications, see the *TCP/IP User's Guide*.
5. On the *local* machine, become superuser.
6. On the *local* machine, add the printer to the *lp* spooler with the script *mknetpr*:

   mknetpr *printer hostname netprinter*

   printer is the local name you want for the remote printer, and should be no more than 14 characters long. *hostname* is the name of the machine the remote printer is on, and *netprinter* is the name of the printer on that machine.

7. Stop the spooler:

   /usr/lib/lpshut

8. To set up this printer as the default printer, type this on the *local* machine:

   /usr/lib/lpadmin -d*printer-name*

9. Restart the spooler:

   /usr/lib/lpsched

9.4 Removing Printers from lp

Under some circumstances, you may want to remove one or more printers from the *lp* system. The *rmprinter*(1M) utility allows you to remove a specified printer. The *preset*(1M) utility allows you to reset your entire *lp* system to the way it was when you received your workstation from Silicon Graphics, Inc. To remove a specified printer, follow these steps:

1. Become the superuser.
2. Remove the printer by entering the command below. Replace *printer-name* with the name of the printer you wish to remove:

   rmprinter *printer-name*

Your printer is now removed from the *lp* system.

To remove all printers on your system:

Caution: Use *preset* with extreme care because it removes all printer configuration information.

1. Become the superuser.
2. Type:

   preset

Your *lp* system is now completely reset and all printers are removed.

9.5 Using the lp Spooler

The Line Printer (*lp*) Spooling Utilities are a set of eleven commands that allow you to *spool* a file that you want to print. Spooling is the name given to the technique of temporarily storing data until it is ready to be processed (in this case, by your printer). For *lp* spooling, a file (or group of files) to be printed is stored in a queue until a printer becomes available. When the printer is ready, the next file in the queue is printed.

lp spooling allows you to use your workstation without waiting for your file to print. *lp* spooling also lets you share printers among many users. The flow of printing throughout your system is regulated by the *lp* Spooling Utilities.

The *lp* Spooling Utilities allow:

- Customizing your system so that it will spool to a pool of printers. (These printers need not be the same type.)
- Grouping printers together into logical classes to maximize throughput.
- Queueing print requests, thus allowing a print request (job) to be processed by the next available printer.
- Cancelling print requests, so that an unnecessary job will not be printed.
- Starting and stopping *lp* from processing print requests.
- Changing printer configurations.
- Reporting the status of the *lp* scheduler.
- Restarting any printing that was not completed when the system was powered down.
- Moving print requests and queues from one printer or class of printers to another.

The eleven *lp* spooling commands are divided into two categories: *user* commands are for general use of the *lp* system; *administrative* commands are for system configuration and maintenance.

9.5.1 Definitions and Conventions

These terms represent important concepts used in this document:

printer A logical name that points to an interface file, which represents a physical device, i.e., the actual printer.

class The name given to an ordered list of one or more printers. A printer may be assigned to more than one class, but need not be a member of any class.

destination The place an *lp* request is sent to await printing. The destination may be a specific printer or a class of printers. An output request sent to a specific printer will be printed only by that printer; a request sent to a class of printers will be printed by the first available printer in its class.

9.5.2 User Commands

This section describes the five basic *lp* commands.

User Command Summary

lp Routes jobs to a destination and places them in a queue. The destination may be either a single printer or a class of printers.

cancel Cancels output requests.

disable Prevents a printer from processing jobs in the queue.

enable Allows a printer to process jobs in the queue.

lpstat Reports the status of all aspects of the *lp* Spooling system.

lp: Make an Output Request

The *lp* command routes a job request to a destination where it is placed in a queue to await printing. The destination may be a single printer or a class of printers. If you do not specify a destination, the request is routed to the default destination. For information on how to set the default printer destination, see the end of this chapter.

The form of the *lp* command is:

lp [*options*] *filenames*

Every time an *lp* request is made, a "request-ID" is assigned to the job, and a record of the request is sent to you. The request-ID has this form:

destination–*seqnum*

destination is the printer or class of printers to which the job has been routed. *seqnum* is an arbitrary sequence number assigned to the job by the *lp* system.

lp has three options which are particularly useful: **–n, –d,** and **–c** as shown in Figure 9-1.

Use **–n** to print more than one copy of a document:

lp **–n***number files(s)*

number is the number of copies to print.

Use **–d** to specify a printer or class of printers other than the default printer (assuming your system has more than one printer defined):

lp **–d***destination filenames*

Finally, use **–c** to ensure that no edits will be made to your files once you have issued a print request:

lp **–c** *files(s)*

You can combine these command options in any order. For a complete list of *lp* options, see the entry for *lp*(1) in the *IRIX User's Reference Manual.*

```
% lp myfile
request id is myprinter-12 (1 file)
% lp < myfile
request id is myprinter-13 (standard input)
% cat myfile | lp
request id is myprinter-14 (standard input)
% lp -n3 -dfoo -c myfile
request id is foo-15 (1 file)
```

Figure 9-1. *lp* Command Samples

There are several different ways to request a printout with the *lp* command. The first three examples in Figure 9-1 perform identical functions, sending a simple print request to the default printer. The fourth example prints three copies on printer *foo*, and creates a copy of the file for the printer to process, thus ensuring that no changes are made to the file after the print request.

cancel: Stop a Print Request

The *cancel* command removes a job from the queue. You can *cancel* a job either before or after it has started printing, but you can *cancel* only one at a time.

Any user may cancel any other user's job. If you cancel another user's print request, mail is sent to that user. Once you *cancel* a job, you can request again only with the *lp* command.

cancel *printer-name*

cancel *request-ID*

Cancelling using the printer name cancels the job currently being printed. Using the request-ID cancels the specified job whether or not it is currently being printed as shown in Figure 9-2.

```
% cancel myprinter
request "myprinter-16" cancelled

% cancel myprinter-17
request "myprinter-17" cancelled
```

Figure 9-2. *cancel* Command Samples

Note: Issuing a *cancel* command will not work when the job is being printed on a remote machine.

disable: Stop Printer from Processing Requests

The *disable* command prevents the printer from processing jobs in the queue. Possible reasons for disabling the printer include malfunctioning hardware, paper jams, running out of paper, or end-of-day shutdowns. If a printer is busy at the time it is disabled, the request it was printing is reprinted in its entirety when you re-enable the printer.

You can send job requests to a printer that has been disabled. The jobs are put on the queue but are not printed until the printer is enabled.

To *disable* a printer, type:

disable [-c] [-r"*reason*"] *printer(s)*

The **–c** option cancels the request currently being printed as well as disabling the printer. This is useful if the output is causing the printer to behave abnormally.

The **–r** option lets you tell other users why you disabled a printer. *reason* is a character string, and must be enclosed in double quotes ("). This string is reported to anyone trying to use the disabled printer.

enable: Allow Printer to Process Requests

The *enable* command permits a printer that has been disabled to begin processing jobs from the queue. Figure 9-3 contains examples of the *enable* command. To *enable* a printer, type:

enable *printer(s)*

```
% disable -r"paper jam" myprinter
printer "myprinter" now disabled
% enable myprinter
printer "myprinter" now enabled
```

Figure 9-3. *disable* and *enable* Command Samples

lpstat: Report lp Status

The *lpstat* command gives you a report on the status of various aspects of the *lp* system. To check *lp* status, type:

lpstat [*options*]

The most useful option is **–t**, which gives a complete report on the status of the *lp* system. For a complete list of options, see the entry for *lpstat*(1) in the *IRIX User's Reference Manual*. Figure 9-4 contains an example of the *lpstat* command.

```
%
scheduler is running
system default destination: myprinter
members of class foo:
myprinter
device for myprinter: /usr/spool/lp/etc/myprinter-log
myprinter accepting requests since Jul 31 21:40
foo accepting requests since Jul 30 12:23
printer myprinter now printing foo-18
enabled since Aug 5 15:34
foo-18 mylogin 3156 Aug 7 17:11  on myprinter
```

Figure 9-4. *lpstat* Command Samples

9.5.3 Administrative Commands

This section summarizes the commands that are used to administer the *lp* system. To execute the administrative commands, you must be logged in as either *root* (i.e., as the super-user) or as *lp*. Inexperienced users should not use the *lp* administrative commands.

Administrative Command Summary

lpsched	Starts the *lp* scheduler.
lpshut	Stops the *lp* scheduler.
reject	Prevents jobs from being queued at a particular destination.
accept	Permits job requests to be queued at a particular destination.
lpmove	Moves printer requests from one destination to another.
lpadmin	Configures the *lp* system.

lpsched : Start the lp Scheduler

The *lpsched* command starts the *lp* scheduler. *lp* prints jobs only when the scheduler is running. *lpsched* is executed automatically each time the computer is booted.

Every time *lpsched* is executed, it creates a file called *SCHEDLOCK* in */usr/spool/lp*. As long as this file exists, the system will not allow another scheduler to run. When the scheduler is stopped under normal conditions, *SCHEDLOCK* is automatically removed. If the scheduler stops abnormally, you must remove *SCHEDLOCK* before you use the *lpsched* command. This procedure may also be necessary to restart the scheduler after the system shuts down abnormally.

To start the *lp* scheduler, type:

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

There is no response from the system to acknowledge the *lpsched* command; to verify that the scheduler is running, use *lpstat*.

lpshut: Stop the lp Scheduler

The *lpshut* command stops the *lp* scheduler and ends all printing activity. All requests that are being printed when you issue the *lpshut* command are reprinted in their entirety when the scheduler is restarted.

To stop the *lp* scheduler, type:

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

reject: Prevent Print Requests

The *reject* command stops *lp* from routing requests to a destination queue. For example, if a printer has been removed for repairs, or has received too many requests, you may wish to prevent new jobs from being queued at that destination.

All requests that are in the queue when you issue the *reject* command are printed if the printer is enabled.

The *reject* command takes the form:

`/usr/lib/reject` [`-r`*"reason"*] *destination(s)*

The **–r** option lets you tell other users why print requests are being rejected. *reason* is a character string, and is enclosed in double quotes ("). This string is reported to anyone trying to use *lp* to send requests to the specified destination.

accept : Allow Print Requests

The *accept* command allows job requests to be placed in a queue at the named printer(s) or class(es) of printers. As shown in Figure 9-5, let a printer receive job requests:

/usr/lib/accept *destination(s)*

```
# usr/lib/accept myprinter
destination "myprinter" now accepting requests

# /usr/lib/reject -r"printer broken" myprinter
destination "myprinter" is no longer accepting requests
```

Figure 9-5. *reject* and *accept* Command Samples

lpmove: Move a Request to Another Printer

The *lpmove* command moves print requests from one destination to another. For example, if you have a printer removed for repairs, you may want to move all jobs pending on the queue to a destination with a working printer. You may also use *lpmove* to move specific requests from one destination to another, but only after you have halted the scheduler with the *lpshut* command. *lpmove* automatically rejects job requests re-routed to a destination without a printer. The *lpmove* command takes two forms:

`/usr/lib/lpmove` *dest1 dest2*

`/usr/lib/lpmove` *request(s) destination*

dest1, *dest2*, and *destination* are printers or classes of printers. *request* is a specific request-ID.

In the first form of the command, all requests are moved from *dest1* to *dest2*. After the move, the printer or printers at *dest1* will not accept requests until you issue an *accept* command. All re-routed requests are renamed *dest2–nnn*, where *nnn* is a new sequence number in the queue for destination *dest2*. In the second form, which you may issue only after you stop the scheduler, the re-routed requests are renamed *destination–nnn*. When you restart the scheduler, the original destinations will still accept new requests. Figure 9-6 contains examples of the *lpmove* command.

```
# /usr/lib/lpmove myprinter yourprinter
# /usr/lib/lpmove foo-19 foo-20 yourprinter
total of 2 requests moved to yourprinter
#
```

Figure 9-6. *lpmove* Command Samples

lpadmin : Configure Printers

The *lpadmin* command has two primary uses: adding new printers to the system, and changing printer classes and destinations. Since Silicon Graphics supplies routines to automatically add the printers supported for use with the computer, the options for adding printers are useful only in the case of dumb printers.

Unlike most IRIX commands, *lpadmin* requires an option. The *lpadmin* command takes three forms:

lpadmin –d[*destination*]

lpadmin –x*destination*

lpadmin –p*printer* [options]

The **–d** option sets the system default destination. The *destination* must already exist when you issue the command. For complete instructions on how to define the default destination, see the end of this chapter.

The **–x** option removes the specified *destination* from the *lp* system. This form of the *lpadmin* command will NOT work while the scheduler is running.

You cannot remove a destination (printer or class) if it has pending requests; you must first either remove all requests with the *cancel* command or move them to other destinations with *lpmove*.

Removing the last remaining member of a class deletes that class from *lp*. Removal of a class, however, does not imply the removal of printers assigned to that class.

The **–p** form of the *lpadmin* command has two options that let you re-assign printers to different classes as shown in Figure 9-7. With these options, the *lpadmin* command takes the form:

lpadmin -pprinter [–c*class*] [–r*class*]

The **–c** option assigns a *printer* to the specified *class*; the **–r** option removes a *printer* from the specified *class*.

```
# /usr/lib/lpadmin -xmyprinter
# /usr/lib/lpadmin -dmyprinter -rfoo -cboo
#
```

Figure 9-7. *lpadmin* Command Samples

The **-p** options will not work while the scheduler is running. For a complete list of options, see the entry for *lpadmin* (1M) in the *IRIX System Administrator's Reference Manual.*

9.5.4 Changing the Default Printer Destination

The *lp* command determines the destination of a request by checking for a **–d** option on the command line. If no **–d** is present, it checks to see if the environment variable LPDEST is set. If LPDEST is not set, then the request is routed to the default destination.

The system default destination can be a printer or a printer class. It is set by using the *lpadmin* command with the **–d** option. The system default must be set by the administrator as *root*. A destination must already exist on the *lp* system before you can designate it as the default destination.

Setting the environment variable LPDEST allows a user to have a default destination other than the system default.

Figure 9-8 illustrates examples of setting the system default with *lpadmin* and setting the user default with LPDEST.

```
# /usr/lib/lpadmin -dmyprinter
#
% setenv LPDEST yourprinter
```

Figure 9-8. Setting the Default Printer

9.6 Maintaining the lp System

This chapter contains procedures for changing your default printer, clearing printer *log* files, and printing over a network.

9.6.1 Changing the Default Printer Destination

The *lp* command determines the destination of a request by checking for a **–d** option on the command line. If no **–d** is present, it checks to see if the environment variable LPDEST is set. If LPDEST is not set, then the request is routed to the default destination.

The system default destination can be a printer or a printer class. It is set by using the *lpadmin* command with the **–d** option. The system default must be set by the user. A destination must already exist on the *lp* system before you can designate it as the default destination.

Setting the environment variable LPDEST allows a user to have a default destination other than the system default.

9.6.2 Clearing Out log Files

The purpose of a *log* file is to keep a record of all printing activity on a given printer. Each printer has a separate *log* file, located in */usr/spool/lp/transcript/log* if it is hardwired, and in */usr/spool/lp/etc/log* otherwise. The name of each printer's *log* file takes the form:

printer-name–log

Each file contains a running list of processed jobs, each of which includes the following:

- The *logname* of the user who made the request
- The request ID
- The name of the printer that processed the request
- The date and time that the printing started

Any *lpsched* error messages that occur are also recorded.

If there is a large number of *lp* requests for a given printer, that printer's *log* file will soon get very large. You can manually remove the contents of these files from time to time, or you can set up the computer to do it for you automatically at regular intervals.

Included in */usr/spool/lp/etc/lib* is a shell script *log.rotate* which will automatically rotate (clean out) your printers' log files once per day at 4:12 AM. To set up the script for your printer(s), you must edit *log.rotate* and */usr/lib/crontab* in the following manner:

1. Become the superuser and change directories to */usr/spool/lp/etc/lib*:

   ```
   cd /usr/spool/lp/etc/lib
   ```

2. In the file *log.rotate*, remove the comment marker (#) from the following line:

   ```
   #printers="PRINTER1 PRINTER2"
   ```

   In place of `PRINTER1` and `PRINTER2`, put the names of any parallel-interface (i.e., color) printers and any remote printers. Any number of printers may be included. If you have no color or remote printers, use the null string ("") in place of printer names.

3. In the file *log.rotate*, remove the comment marker (#) from the following line:

   ```
   #LocalPS="PRINTER1"
   ```

In place of `PRINTER1`, put the names of any hardwired (i.e, connected to the serial port) printers.

4. Change directories to */usr/lib*:

```
cd /usr/lib
```

5. Edit *crontab* by removing the comment marker from the line containing *log.rotate*.

9.6.3 Printing Over the Network

Remote printing on the computer allows you to send print jobs over the network with the same commands they use to send jobs to a printer connected directly to your computer. This is accomplished by giving a remote printer a local name so that the local *lp* scheduler is "fooled" into thinking it is sending the request to a local printer. After the local machine's *lp* spooler queues the print request, it is sent across the net to the remote machine, where it is processed by that machine's *lp* spooler. As a result of this, one cannot accurately determine the status of a remote print request by using the *lpstat* command on the local machine.

This section covers two aspects of remote printing:

- Checking the status of remote print requests
- Cancelling remote print requests

Checking Remote Printer Status

When you send a print request across the net to a remote machine, the local *lp* system will always report that the request is being printed, regardless of its actual status in the remote machine's *lp* system. To check the true status, you must remotely access (using *rsh* or *rlogin*) the machine whose printer is processing the job. The remote *lp* scheduler changes the request ID of any job sent to it over the net to reflect the actual name of the printer, and gives it a new sequence number corresponding to its place in the remote queue. The way to determine a specific job's status is to look in the remote printer's *log* file (i.e., the *log* file on the remote machine) with the *tail* command. The example below uses *rsh* to access the remote machine.

`rsh` *host* `tail` *logpath*

host is the name of the remote machine. *logpath* is the pathname of the remote printer's log file.

Cancelling Remote Print Requests

Once you know the remote printer status, you can use the *cancel* command on the remote machine to cancel any jobs on the printer's queue. You must cancel a remote print job from the remote machine once it has been sent over the net by the local *lp* system.

9.7 Troubleshooting

If you send a print request to the LaserWriter with *lp*, *psroff*, or *enscript* and do not receive any output, you should use the checklists below to make sure your system is ready for printing. Use these lists as a supplement to the troubleshooting information in the manufacturer's hardware manual.

9.7.1 Hardware Troubleshooting Checklist

1. **Is the printer turned on?** The on/off switch is at the bottom of the left side of the printer, behind the paper tray. If the printer is on and working properly, a green light will be lit on the LaserWriter's front panel.
2. **Does the printer have paper?** If the printer is out of paper, a yellow light will be lit on the front panel of the LaserWriter. To add paper, slide out the paper tray, insert paper under the metal clips on the tray, and slide the tray back in place. Be careful not to overfill the tray.
3. **Is there a paper jam?** If there is a paper jam, a red light will be lit on the LaserWriter's front panel. Follow the directions in the manufacturer's manual for clearing jammed paper from the printer.
4. **Is the printer set to 9600 baud?** If it is not, turn the printer off, set the switch next to the serial port on the back of the printer to **9600**, and turn the printer back on.
5. **Is the serial cable attached correctly?** One end should be attached to the RS-232 port on the back of the LaserWriter. Make sure that the other end of the cable is attached to the proper serial port (port 2 is recommended).

9.7.2 Software Troubleshooting Checklist

The *lp* scheduler is the program in charge of spooling your files to the printer, and is invoked whenever you use the *lp*, *psroff*, or *enscript* print commands. The scheduler can be in a number of states, and each printer registered with *lp* can be in a number of states as well.

To check on the complete status of the *lp* system, type:

```
lpstat -t
```

This gives you a complete description of the status of *lp*. Use this information to answer these questions:

1. **Is your printer registered with *lp*?** If you do not see the name of your printer in the list of information produced by *lpstat*, then you will have to register your printer with *lp*.

2. **Is the printer enabled?** If your printer is not enabled, the *lpstat* listing will contain this line:

   ```
   printer yourprinter disabled since...
   ```

   In order to enable the printer, type:

   enable *yourprinter*

   lp sometimes disables a printer automatically if it is unable to send a file to a remote printer, so a disabled printer is often an indication of a hardware problem, such as a host that is not communicating with the network.

3. **Is the printer accepting requests?** If the printer is not accepting requests, the *lpstat* listing will contain this line:

   ```
   yourprinter not accepting requests since...
   ```

   You will have to execute the *accept* command for that printer destination. Become the superuser (with *su*) and type:

   /usr/lib/accept *yourprinter*

4. **Is the *lp* scheduler running?** If the scheduler is not running, the *lpstat* listing will contain the message:

   ```
   scheduler is not running
   ```

   To restart the *lp* scheduler, become superuser (with *su*) and type:

   /usr/lib/lpsched

5. **Did you specify the right printer?** If your system has more than one printer, and you wish to send a job to a printer other than the default, remember to use the **–d** option:

```
lp -dotherprinter
psroff -dotherprinter
enscript -dotherprinter
```

9.7.3 Troubleshooting Network Printers

If you are having trouble with a printer you are accessing over a network, you should check the status of the *lp* scheduler both on your machine *and* the printer's host machine.

9.7.4 Emergency Measures

If none of the above procedures work, there are several "last resort" procedures you can try.

1. Stop the *lp* scheduler and then restart it. As *root*, type the following sequence:

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

2. Remove the offending printer destination from the *lp* scheduler, and then register it again. Before you can do this you will either have to cancel any print requests going to the printer or move them to another print destination (if you have more than one). See the sections describing the *cancel* and *lpmove* commands in
3. As an absolute last resort, remove all printers from the *lp* system, reboot the computer, and register them all once again.

9.8 lp Error Messages

This section provides a description of the error messages that are associated with *lp* commands. The following variables are used in the error messages:

file(s)	indicates the file or files that are to be printed.
dest	indicates the name of the destination printer.
printer-id	indicates the request identification number of the printout. For example, *myprinter-46* is the printer name followed by the request identification number.
printer-name	indicates the name of the printer.
program-name	indicates the program name that was executed.
user	indicates the user who requested the printout.

Following each message is an explanation of the probable cause of the error and the corrective action to take. If you are not able to correct all the error conditions you encounter, call your service representative for assistance.

Error Message	Description/Action
dest is an illegal destination name	The *dest* you used is not a valid destination name. Use the *lpstat -p* command to list valid destination names.
file is a directory	The file name you typed is a directory and cannot be printed.
xx is not a request id or a printer	The argument you used with the *cancel* command is not a valid request identification number or a printer name. Use the *lpstat -t* command to give you all the printers and requests waiting to get printed.
xx is not a request id	The request identification number you used with the *lpmove* command is not a valid request identification number. To find out what requests are valid, use the *lpstat -u* command.
xx not a request id or a destination	You used an invalid request identification number or destination with the *lpstat* command. To find out what is valid, use the *lpstat -t* command.
dest not accepting requests since *date*	Requests to the printer which you are trying to use have been stopped by the *reject* command.
Can't access FIFO	The named pipe file */usr/spool/lp/FIFO* is incorrect. The mode should be 600.
lp Administrator not in password file	You must have an entry in the */etc/passwd* file for "lp," and you must belong to the group "bin."
acceptance status of destination "printer-name" unknown	Use the *accept* command to enable the printer so that it will accept requests.
can't access file "xx"	The mode could be wrong on your directory or the file that you are trying to access.

Error Message	Description/Action
can't create class "xx"-it is an existing printer name	The class name you are trying to use has already been given to a printer. You will have to use another name or remove the printer to use the class name.
can't create new acceptance status file	The mode may be wrong on the */usr/spool/lp* directory. It should be 755 with the owner "lp" and the group "bin."
can't create new class file	The mode may be wrong on the */usr/spool/lp* directory. It should be 755 with the owner "lp" and the group "bin."
can't create new interface program	The mode may be wrong on the */usr/spool/lp/interface* directory. It should be 755 with the owner "lp" and the group "bin."
can't create new member file	The mode may be wrong on the */usr/spool/lp/member* directory. It should be 755 with the owner "lp" and the group "bin."
can't create new printer status file	The mode may be wrong on the */usr/spool/lp/pstatus*. It should be 644 with the owner "lp" and the group "bin."
can't create new request directory	The mode may be wrong on the */usr/spool/lp/request* directory. It should be 755 with the owner "lp" and the group "bin."
can't create printer "printer-name" -- it is an existing class name	The printer-name you are trying to use has already been used as a class name. You will have to assign another name for the printer.
can't create new output queue	The mode on the file */usr/spool/lp/seqfile* is incorrect. It should be 644, and the mode on the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.

Error Message	Description/Action
can't create new sequence number file	The mode on the file */usr/spool/lp/seqfile* is incorrect. The mode of the file should be 644, and the mode of the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.
can't create request file "xx"	The mode on the file */usr/spool/lp/request/printer-name/*r-id is incorrect. *Printer-name* is the name of the printer such as dqp10, and r-id is the request identification number. The mode of the file should be 444, and the mode of the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.
can't fork	You either have several processes running and are not allowed to run anymore, or the system has all the processes running that it can handle. You will have to rerun this command later.
can't lock acceptance status	This is a temporary file in */usr/spool/lp* that prevents more than one "lp" request from being taken at any given instant. You will have to rerun this command later.
can't lock output queue	The file */usr/spool/lp/QSTATLOCK* prevents more than one "lp" request from being printed on a printer at a time. You will have to rerun this command later.
can't lock printer status	The temporary file */usr/spool/lp/PSTATLOCK* prevents more than one "lp" request from being printed on a printer at a time. You will have to rerun this command later.
can't lock sequence number file	The file */usr/spool/lp/SEQLOCK* prevents more than one "lp" request from getting the next printer-id (request identification) number at a time. You will have to rerun this command later.

Error Message	Description/Action
can't move request printer-id	*Printer-id* is the request identification number that cannot be moved. You will probably have to change the modes on the files and directories in */usr/spool/lp/request*. Also, you will have to manually move the request from the disabled printer directory to the new destination after you shut down the *lp* scheduler.
can't open class file	The *lp* program is trying to access the list of classes for printers. One reason it may not be able to open the class file is that the system could have the maximum number of files open that are allowed at any time. This can be corrected by typing the command at a later time.
can't open member file	The *lp* program is trying to access the list of members in the directory */usr/spool/lp/member*. The system could have the maximum number of files open that are allowed at any time. This can be corrected by typing the command at a later time.
can't open xx file in MEMBER directory	There are a couple of reasons why file *xx* in the */usr/spool/lp/member* directory cannot be opened. The mode on the file could be incorrect. It should be 644. Another possibility is that the system could have the maximum number of files open that are allowed at any time. This can be corrected by typing the command at a later time.
can't open xx file in class directory	One possibility why file *xx* cannot be opened is that the mode on the file or directory is incorrect. The file mode should be 644, and the directory mode should be 755. Another possibility is that the system has the maximum number of files open that are allowed at any time. The latter problem can be corrected by typing the command at a later time.
can't open xx	You cannot print on printer *xx* because the mode is incorrect on **/dev/tty**. The mode should be 622.
can't open FIFO	The mode on the named pipe file */usr/spool/lp/FIFO* may be incorrect. It should be 600. Or, the system could have the maximum number of files open that are allowed at any time. The latter problem can be corrected by typing the command at a later time.

Error Message	Description/Action
can't open MEMBER directory	The mode on the directory */usr/spool/lp/member* could be incorrect. It should be 755. Another possibility is that the system could have the maximum number of files open that are allowed at any time. If the maximum number of files are open, try typing the command at a later time.
can't open acceptance status file	The mode on the file */usr/spool/lp/qstatus* may not be correct. It should be 644. Another possibility is that the system could have the maximum number of files open that are allowed at any time. The latter problem can be corrected by typing the command at a later time.
can't open default destination file	Check the mode on the file */usr/spool/lp/default*. The mode should be 644. If the mode is okay, it could be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the command at a later time.
can't open file filename	The *filename* was incorrectly typed or you don't have the correct modes set. The mode should be at least 400 if you are the owner.
can't open output queue file	Check the mode on the file */usr/spool/lp/outputq*. It should be 644. This error message could also be generated if the system has the maximum number of files open that are allowed at any one time. Try entering the command at a later time.
can't open printer status file	The mode on the file */usr/spool/lp/pstatus* is incorrect. The mode should be 644. It could also be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the command at a later time.
can't open request directory directory name	The mode on the directory */usr/spool/lp/request* is incorrect. The mode should be 655. It could also be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the command at a later time.

Error Message	Description/Action
can't open request file xx	The mode on the file */usr/spool/lp/member/request/xx* is incorrect. The mode should be 644. It could also be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the *lpmove* command at a later time.
can't open system default destination file	The mode on the file */usr/spool/lp/default* is incorrect. The mode should be 644. It could also be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the command again at a later time.
can't open temporary output queue	The mode on the file */usr/spool/lp/outputq* is incorrect. The mode should be 644. It could also be that the system has the maximum number of files open that are allowed at any one time. This can be corrected by trying the command at a later time.
can't proceed -- scheduler running	Many of the *lpadmin* command options cannot be executed while the scheduler is running. Stop the scheduler using the *lpshut* command and then try invoking the command again.
can't read current directory	The *lp* and *lpadmin* commands cannot read the directory containing the file to be printed. The directory name may be incorrect or you do not have read permission on that directory.
can't remove class file	The mode may be wrong on the */usr/spool/lp/class*. It should be 755. The owner should be "lp," and the group should be "bin." Another possibility is the file in that directory may have the wrong mode. It should be 644.
can't remove printer	The mode may be wrong on the */usr/spool/lp/member* directory. It should be 755, and the files in that directory should be 644. Both the directory and the files should be owned by "lp," and the group should be "bin."

Error Message	Description/Action
can't remove request directory	The mode may be wrong on the */usr/spool/lp/request* directory. It should be 755 and should be owned by "lp," and the group should be "bin." The directory may still have pending requests to be printed which will have to be removed before the directory can be removed.
can't set user id to *lp* Administrator's user id	The *lpsched* and *lpadmin* commands can only be used when you are logged in as "lp" or "root."
can't unlink old output queue	The *lpsched* program cannot remove the old output queue. You will have to remove it manually by using the command *rm /usr/spool/lp/outputq*.
can't write to xx	The *lpadmin* command cannot write to device *xx*. The mode is probably wrong on the /dev/ttyxx file. It should be 622 and owned by "lp."
cannot create temp file filename	The system may be out of free space on the */usr* file system. Use the command *df /usr* to determine the number of free blocks. Several hundred blocks are required to insure that the system will perform correctly.
class "xx" has disappeared!	Class *xx* was probably removed since the scheduler was started. The system may be out of free space on the /usr file system. Use the command **df /usr** to find out. Use the *lpshut* command to stop the scheduler and restore the class from a backup.
class "xx" non-existent	The class *xx* may have been removed because the system is out of free space on the */usr* file system. Use the command **df /usr** to find out how much free space is available. The class will probably have to be restored from a backup.
class directory has disappeared!	The */usr/spool/lp/class* directory has been removed. The system may be out of free space on */usr*; use the **df** */usr* command to find out. The class directory contains all the data for each printer class. To restore this directory, get these files and directory from a backup.

Error Message	Description/Action
corrupted member file	The */usr/spool/lp/member* directory has a corrupted file in it. You should restore the directory from backup.
default destination "dest" non-existent	Either the default destination is not assigned or the printer *dest* has been removed. Use the *lpadmin* to set up a default destination or set **LPDEST** to the value of the destination.
destination "dest" has disappeared!	A destination printer, *dest* has been removed since *lpsched* was started. Use the *lpadmin* command to remove the printer.
destination "printer-name" is no longer accepting requests	The printer has been disabled using the *reject* command. The printer can be re-enabled using the *accept* command.
destination dest non-existent	The destination printer you specified as an argument to the *accept* or *lpadmin* command is not a valid destination name, or it has been removed since the scheduler was started.
destination "printer-name" was already accepting requests	The destination printer was previously "enabled." Once a printer is accepting requests, any further *accept* commands are ignored.
destination "printer-name" was already not accepting requests	A *reject* command was already sent to the printer. Use the *accept* command to allow the printer to start accepting requests again.
destination printer-name is not accepting requests move in progress ...	The printer has been disabled by the *reject* command, and requests are being moved from the disabled printer to another printer. The printer can be enabled again by the *accept* command.
destinations are identical	When using the *lpmove* command, you need to specify a printer to move the print requests from and a different printer to move the requests to.
disabled by scheduler: login terminal	The login terminal has been disabled by the *lp* scheduler. The printer can be re-enabled by using the *enable* command.

Error Message	Description/Action
error in printer request printer-id	*Printer-id* is the actual request identification number. The error was most likely due to an error in the printer. Check the printer, and reset it if needed.
illegal keyletter "xx"	An invalid option, *xx*, was used. See the manual page for the correct options.
keyletters "-xx" and "-yy" are contradictory	This combination of options to the *lpadmin* program cannot be used together.
keyletter "xx" requires a value	The option *xx* requires an argument. For example, in the command line *lpadmin -m model* the argument to the **-m** option is the name of a model interface program.
keyletters -e, -i, and -m are mutually exclusive	These options to the *lpadmin* command cannot be used together. Refer to the manual page for more information on usage.
lp: xx	In this message the variable *xx* could be one of several arguments. Typically, it is a message telling you the default destination is not assigned.
member directory has disappeared!	The */usr/spool/lp/member* directory has been removed. The system is probably out of free disk space in the */usr* file system. You need to clean up the */usr* file system, and then install the *lp* commands or retrieve them from a backup.
model "xx" non-existent	The name that you are using for a model interface program is not a valid one. A list of valid models is in the */usr/spool/lp/model* directory.

Error Message	Description/Action
new printers require -v and either -e, -i, or -m	A printer must have an interface program, and this is specified by **-e**, **-i**, or **-m** options. The **-v** option specifies the device file for the printer. For more information on these options, refer to the *lpadmin* manual page.
no destinations specified	There are no destination printers specified. Use the *lpadmin* command to set one up.
no printers specified	There are no printers specified. Use the *lpadmin* command to set one up.
non-existent printer xx in PSTATUS	A printer with the name *xx* is in the */usr/spool/lp/pstatus* file but no longer exists. The printer should be removed using the *lpadmin* command.
non-existent printer printer-name in class xx	The printer that you are trying to address in class *xx* has been removed from that class.
out of memory	Implies the system is in trouble. The message implies that there is not enough memory to contain the text to be printed.
printer "printer-name" already in class "xx"	The printer you are trying to move to class *xx* is already in that class. You cannot move a printer to a class that it is already in.
printer "printer-name" has disappeared!	The printer has been removed, and the *enable* command cannot find it. The printer was most likely removed since the machine was rebooted or since the scheduler was started.
printer "printer-name" non-existent	*Printer-name* is the name of a printer that has been removed since the scheduler has been started. You must use the *lpadmin -xprinter-name*.
printer status entry for "printer-name" has disappeared	The */usr/spool/lp/pstatus* file must have been corrupted. You will have to resubmit the printer request.

Error Message	Description/Action
printer "printer-name" was not busy	The printer is not printing a request at this time. Either the request you wanted to cancel is finished printing or you have specified the wrong printer.
request "printer-id" non-existent	You are attempting to cancel a request that does not exist. You may have given the wrong printer name or wrong request id number or the request may have finished printing.
request not accepted	The request was not accepted by *lp*. The scheduler may not be running. Use the *lpstat -t* command to find out more information.
requests still queued for "printer-name" -- use lpmove	*Printer-name* is the printer that still has requests waiting to get printed. You need to use the *lpmove* command to get those requests moved to another printer.
scheduler is still running -- can't proceed	You cannot perform this command while the scheduler is running. You will have to use the *lpshut* command first.
spool directory non-existent	The directory */usr/spool* has been removed. You will have to use the **mkdir** command to restore the directory. This has probably removed some of the necessary *lp* files. You may have to reinstall the *lp* commands.
standard input is empty	You specified an invalid file name either by incorrectly typing a name or by specifying a nonexistent file. Nothing will be printed on the printers from this request.
this command for use only by *lp* Administrators	This command is restricted to someone logged in as root.
too many options for interface program	The *lp* command called the appropriate interface program with too many arguments. For more information on the options and arguments that can be used with the *lp* command, refer to the *lp* manual page.

10. Terminals, Modems, and Dumb Printers

The TTY system supports serial communication between users and serial devices, such as dumb printers, modems, and terminals. This chapter tells you how to administer the TTY system. It includes information about:

- Connecting an ASCII terminal, modem, or dumb serial printer
- Serial port cabling and pin signals for the standard serial ports

To connect peripherals that are not covered in this chapter, see the documentation that accompanies the peripheral.

10.1 Definition of Terms

The following terms are used in this chapter:

TTY Derived from the near-classic abbreviation for teletypewriter, the term covers the whole area of access between the IRIX system and peripheral devices, including the system console. It shows up in commands such as *getty*(1M) and *stty*(1), in the names of device special files such as */dev/ttyd1*, and in the names of files such as */etc/gettydefs*, which is used by *getty*.

TTY line The physical equipment through which access to the computer is made.

port A synonym for TTY line.

line settings A set of line characteristics.

baud rate The speed at which data is transmitted over the line. A part of line settings.

mode The characteristics of the terminal interface. A part of line settings. The TTY line and the terminal must be working in the same mode before communication can take place. Described in *termio*(7).

hunt sequence A circular series of line settings such as different baud rates. During the login sequence, a user looking for a compatible connection to the computer can go from one setting to the next by sending a BREAK signal.

terminal options Selectable settings that define the way a given terminal operates. Described in *termio*(7).

10.2 Administering the TTY System

10.2.1 Checking TTY Line Settings

For information on line settings, follow this sequence.

1. Issue the *sysadm ttymgmt* command, then select item 1 from the TTY Management menu; or, issue the *sysadm lineset* command at the system prompt to go directly to the lineset display. You see this display:

```
sysadm lineset
Running subcommand 'lineset' from menu 'ttymgmt',
TTY MANAGEMENT

Tty Line Settings and Sequences

co_1200     co_300     co_9600     co_4800     co_2400
console     (does not sequence)
du_1200     du_300     du_2400
dx_1200     (does not sequence)
dx_2400     (does not sequence)
dx_4800               (does not sequence)
dx_9600     (does not sequence)
```

   Each of the line settings is a name that identifies a set of TTY line characteristics. During the login process, the line settings on one line hunt from left to right, moving from one setting to the next when you press **<break>**. The setting on the far right of each line hunts to the first one again, forming a circular hunt sequence.

2. To look at a line setting in detail, specify any entry. Follow this sequence:

```
Select one line setting to see it in detail [?,q]: co_9600

Line Setting:                co_9600
        Initial Flags:       B9600
        Final Flags:         B9600 SANE TAB3
        Login Prompt:        \r\n\n$HOSTNAME login:
        Next Setting:        co_4800

B9600   9600 Baud
SANE    Set All Modes To "Traditionally Reasonable" Values
TAB3    Expand Horizontal-tab To Spaces

Select another line setting or
<RETURN> to see the original list [?, q]: du_2400

Line Setting:                du_2400
        Initial Flags:       B2400
        Final Flags:         B2400 SANE TAB3 HUPCL
        Login Prompt:        \r\n\n$HOSTNAME login:
        Next Setting:        du_1200

B2400   2400 Baud
HUPCL   Hang Up on Last Close
SANE    Set All Modes To "Traditionally Reasonable" Values
TAB3    Expand Horizontal-tab To Spaces

Select another line setting or
<RETURN> to see the original list [?, q]: q
```

10.2.2 Attaching an ASCII Terminal

This section describes the procedures for connecting and configuring an ASCII terminal. The ASCII terminal connected to the port labeled **1** on the I/O panel is the *diagnostics terminal.* The messages produced by the power-on diagnostics appear on the screen of this terminal.

Connecting the Hardware

Connect the ASCII terminal to the port labeled **1** on the I/O panel. See section 10.3 for information about port cabling and pin signals.

Configuring the Software

This section tells you how to configure IRIX software to use an ASCII terminal with your computer.

The */usr/lib/terminfo* directory contains files that describe different terminal models, their capabilities, and how they operate.

For most ASCII terminal models, you do not need to edit this database. Check to see whether the information on your terminal model appears in the directory */usr/lib/terminfo* by following the procedure below. If your terminal is not in the database, or if it does not work properly after you have configured the software, you need to write a terminal description. See Writing Terminal Descriptions in the chapter entitled curses/terminfo in the *IRIX Programmer's Guide*, *tset*(1) and *stty*(1) in the *IRIX User's Reference Manual*, and *terminfo*(4) in the *IRIX Programmer's Reference Manual.*

The directory */usr/lib/terminfo* is divided into numerical and alphabetical subdirectories. Each subdirectory contains entries for terminals whose names begin with that character. For example, */usr/lib/terminfo/v* contains the entry for the Visual 50. The entry name listed in the subdirectory is v50am.

To find the entry name for your terminal and to configure software for an ASCII terminal, follow these steps:

1. Log in as *root* or become the superuser by entering the *su* command.
2. Change directories to */usr/lib/terminfo.*

```
cd /usr/lib/terminfo
```

3. To find the name of your terminal, issue a *grep* command with a string you suspect could make up part of your terminal name. If this fails, examine the subdirectories of */usr/lib/terminfo*, which contain all the terminal entries.

   ls -R | fgrep *string*

4. Once you know the terminal name as it appears in */usr/lib/terminfo*, issue the *infocmp* command to find out the model name of your terminal. For example, for a Visual 50, issue this command:

   infocmp -I v50am

   You see a display which begins with this line:

   ```
   v50am|visual50 (v50 emulation) with automatic margins,
   ```

   The data in the first field (v50am) is the model name of your terminal. In the next step, you'll enter this model name in the */etc/ttytype* file.

5. Edit */etc/ttytype*. This file tells which type of terminal is connected to which port. In the line that contains the port you are using, replace du with the model name of your terminal. The question mark (?) at the beginning of the line in */etc/ttytype* causes *tset* to prompt for the kind of terminal you are using when you log on through that port. An */etc/ttytype* might look like this example:

   ```
   iris-ansi       systty
   ?v50am  ttyd1
   ?v50am  ttyd2
   ?v50am  ttyd3
   ?v50am  ttyd4
   ?v50am  ttyd5
   ?v50am  ttyd6
   ?v50am  ttyd7
   ?v50am  ttyd8
   ?v50am  ttyd9
   ?v50am  ttyd10
   ?v50am  ttyd11
   ?v50am  ttyd12
   ```

   You normally call *tset* in your login startup script (*~/.cshrc* or *~/.profile*). *tset* commands use information from */etc/ttytype* and */usr/lib/terminfo* to

initialize the terminal. These files also provide information on setting up environment variables so that editors and other programs know how to communicate with the terminal. See *tset*(1).

6. Edit */etc/inittab* so that you can log in to the computer ports. This is a sample from an */etc/inittab* file:

```
co:234:respawn:/etc/getty console console none
t1:23:respawn:/etc/getty -s console ttyd1 co_9600  # alt. console
t2:23:off:/etc/getty ttyd2 co_9600
t3:23:off:/etc/getty ttyd3 co_9600
t4:23:off:/etc/getty ttyd4 co_9600
```

Here are two example entries, with an explanation of each field in the entries:

```
t1:23:respawn:/etc/getty -s console ttyd1 co_9600
t2:23:off:/etc/getty ttyd2 co_9600
```

t1 uniquely identifies the entry.

23 defines the run level in which this entry is to be processed. A 23 means this entry is to be processed in run levels two and three.

off means never to perform the action on the process field of *init*.

respawn means to perform the action on the process field of *init*. See *inittab*(4) for a description of all possible actions.

/etc/getty ttyd2 co_9600

runs the *getty* process on the port labeled 1 at the baud rate and options specified in the co_9600 entry in the file */etc/gettydefs*.

To enable you to log in to the terminal you connected to the port labeled **2**, remove the `x` in the second field of the entry for the port labeled **2**, Find this line:

```
t2:x:off:/etc/getty ttyd2 co_9600
```

and change it to:

```
t2:23:respawn:/etc/getty ttyd2 co_9600
```

7. */etc/inittab* refers to */etc/gettydefs* for information about the terminal baud rate. In the example from */etc/inittab* above, `co_9600` refers to the name of an entry in */etc/gettydefs*. This entry defines a 9600 baud setting. If you don't plan to run the terminal at 9600 baud, replace `co_9600` in */etc/inittab* with the correct entry name from */etc/gettydefs*. To see which entries are defined in */etc/gettydefs*, examine the file, or see Section 4.6.1, Checking TTY Line Settings. To make a new entry, see Section 4.6.2, Making TTY Line Settings. The entries in */etc/gettydefs* look like this:

```
dx_9600# B9600 # B9600 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #dx_9600
dx_4800# B4800 # B4800 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #dx_4800
dx_2400# B2400 # B2400 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #dx_2400
dx_1200# B1200 # B1200 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #dx_1200
du_1200# B1200 # B1200 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_300
du_300#  B300  # B300  SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_2400
du_2400# B2400 # B2400 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #du_1200
```

The entries beginning with `dx` are typically used for terminals; those beginning with `du` typically for modems. See *gettydefs*(4) for more information on the fields of each entry.

When the terminal is powered on, the workstation sends a login prompt to the terminal screen. Press <enter> if the login prompt doesn't appear. If the default line speed set in */etc/inittab* is incorrect, the prompt may be garbled or may not appear.

8. Inform *init* of the change to */etc/inittab*, and start a *getty* process for the port:

```
telinit q
```

10.2.3 Attaching a Modem

This section describes the procedures for connecting the modem, configuring a port for a dial-in modem, and configuring a port for a dial-out modem. See section 10.3 for information about port cabling and pin signals.

Connecting the Hardware

The computer supports all Hayes compatable 110 through 2400 baud modems. Connect the modem to the port labeled **1** or the port labeled **2** on the computer with an RS-232 cable.

Both of these ports support *data carrier detect* and *data terminal ready*. See the tables in section 10.3 for information on the computer port signals.

Configuring the Software for a Dial-Out Modem

This section describes the configuration procedures for a dial-out modem. For more information, see Chapter 11, "UUCP".

The following example shows how to connect the necessary hardware and configure the system for a modem.

1. Log in as *root* or become the superuser by entering the *su* command.
2. Edit the file */usr/lib/uucp/Devices*. This file contains line speed entries for each port.

   The first field of this entry defines the connection as direct (Direct).

   The second field is the device name of the port. *ttym1* corresponds to the port labeled **1**, and *ttym2* corresponds to the port labeled **2**.

   The third field is a place holder (-).

   The fourth field is the baud rate of the connection. Use a baud rate that works for both the local modem and the modem on the other end of the telephone line.

   The fifth field contains a pair of dialers and tokens. It contains the word *direct* for a direct connection.

For example, to connect a modem that operates at 2400 baud to the port labeled **1**, add this line:

```
Direct ttym1 - 2400 direct
```

For more information on the *Devices* file, see Chapter 11, "UUCP".

3. Edit the file */usr/lib/uucp/Systems*. This file has information about sites that *uucp* can communicate with. Entries in this file have these fields:

   system_name time device class phone login

system_name	the name of the remote system; must be unique in the first seven characters
time	when the remote system can be accessed; ANY means any day of the week, any time
device	the device type used for the call
class	the baud rate for the device
phone	the dialer sequence the modem will use to call the remote system
login	the login password and special character sequence needed to complete the login

For more information on the file *Systems*, see chapter 11, "UUCP".

For example, if you are attaching a 1200–baud modem to the port labeled **2**, and you want it to automatically call a host named *omachine* at 415-555-1212, you might enter these lines in the */usr/lib/uucp/Systems* file:

```
# Name of company
#talk to (415)555-1212, Tom Jones.
#mail to omachine!him
#incoming=Upyramid, password cube
omachine Any ttyd2 1200 ttyd3 "" \rATS2=128\r\c \
OK atdt4155551212\r 1200 \
\r\c ogin:-\b\d-ogin:--ogin:--ogin: \
Uname ssword: PaExWoRd
```

Note: The line beginning with 'omachine' and ending with 'PaExWoRd' is actually one line. Do not type any carriage returns; allow the system to wrap the line. Or, precede each carriage return with a backslash, as above.

4. Edit the file */etc/inittab* to disable logins on the port connected to the modem. Find the line that corresponds to the port you chose for the modem. *ttyd1* corresponds to the port labeled **1**, and *ttyd2* corresponds to the port labeled **2**. For example, if you connected the Hayes modem to the port labeled **2**, find this line:

   ```
   t2:23:off:/etc/getty ttyd2 co_9600
   ```

5. Inform *init* of the change to */etc/inittab* by typing the command:

   ```
   telinit q
   ```

6. Test the serial line with *cu*:

   ```
   cu -sbaudrate -ldevice_name
   ```

   The *device_name* in the above command line should be replaced with the entire pathname of the device, for example, */dev/ttyml*... The computer should connect to the direct line specified by the **-l** option. See *cu*(1C) for more information.

Configuring the Software for a Dial-In Modem

This section describes the configuration procedures for a dial-in modem. For more information, see the chapter entitled UUCP.

1. Log in as *root* or become the superuser by entering the *su* command.
2. Edit the file */etc/inittab* to disable logins on the port connected to the modem, and to specify that the modem device driver be used with that port. Find the line that corresponds to the port you chose for the modem. *ttyd1* corresponds to the port labeled **1**, and *ttyd2* corresponds to the port labeled **2**. Edit the line, changing *ttyd1* to *ttym1* or *ttyd2* to *ttym2*. Make sure there is a *23* in the second field so you can log in to the port. Also, change *dx* to *du*, and enter the correct baud rate. For example, if you connected the Hayes modem to the port labeled **2**, find this line:

   ```
   t2:23:off:/etc/getty ttyd2 dx_9600
   ```

   and modify it to read:

   ```
   t2:23:respawn:/etc/getty ttym2 du_1200
   ```

3. Inform *init* of the change to */etc/inittab* by typing the command:

   ```
   telinit q
   ```

The *du* entries in */etc/gettydefs* toggle between 2400, 1200, and 300 baud. If *getty* is listening at the wrong baud rate when you log in, the prompt may be garbled or may not appear. Select another line speed by pressing the <break> key after the login message is displayed. See *gettydefs*(4) and *getty*(1M).

10.2.4 Attaching a Dumb Serial Printer

Silicon Graphics, Inc., supports the use of dumb printers with IRIS-4D Series workstations and servers. You can attach dumb printers to the computer through any of the serial ports, but you need to configure such a printer on your own. Only users who are familiar with printer interfacing should try to connect a dumb printer. See Chapter 13, "Kernel Configuration", if you plan to write a device driver for a dumb printer.

General Procedure

The following set of procedures gives a general outline for adding a dumb printer to the IRIX line printer (*lp*) spooler.

1. Log in as *root*, or become the superuser by entering the *su* command.
2. To avoid unwanted output from non-*lp* processes and to ensure that *lp* can write to the device, enter the following commands:

   ```
   chown lp  /dev/ttyd port
   chmod 600  /dev/ttyd port
   ```

   port is the number of the hardware port to which the printer is connected.
3. To prevent the computer from sending a login prompt to the printer, run the next command to kill unwanted gettys and then edit the file */etc/inittab*.

   ```
   telinit q
   ```

   Now, find the line corresponding to the serial port your printer is connected to. It should look something like this:

   ```
   dport:23:respawn:/etc/getty ttydport dx_9600
   ```

   and change `respawn` to `off` as follows:

   ```
   dport:x:off:/etc/getty ttydport dx_9600
   ```

   Putting an "x" in the initstate field accomplishes the same end.

4. Turn off the scheduler with the *lpshut* command.

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

5. Introduce the printer to the *lp* system with the *lpadmin* command. This command assigns a printer to a filter. You need to specify a *printer* name, a *device* file, and an interface program or *filter*.

   The *printer* name must conform to the following rules:

   - It must be no longer than ten characters.
   - It must consist solely of alphanumeric characters and underscores.
   - It must be unique within the *lp* system.

   device is the device file associated with the printer: */dev/ttyd1* or */dev/ttyd2*.

   Choose the *filter* with either the **–m** or **–i** option.

   Use **–m** if you want to use one of the "model" filters supplied with the *lp* system in */usr/spool/lp/model*.

   ```
   /usr/lib/lpadmin –pprinter –vdevice –mmodel
   ```

   See the following subsection, "Printer Filters", for more on the model filters.

   Use **–i** if you want to use a filter from the interface directory:

   ```
   /usr/lib/lpadmin –pprinter –vdevice –ifilter
   ```

   filter is your own filter.

   You may also want to use the **–h** option, which tells *lp* that the printer is hardwired to the computer. **–h** does not take any arguments, and may appear anywhere after the **–p** option on the command line.

6. Start the *lp* scheduler with the *lpsched* command.

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

7. Allow the printer to accept requests to its queue with the *accept* command.

   /usr/lib/accept

8. Enable the printer with the *enable* command.

   enable *printer*

Printer Filters

Printers that are used with the *lp* system must have a printer interface program, or *filter*. Every print request made with the *lp* command is routed through an appropriate filter before it is printed, as illustrated in Figure 10-1.

Figure 10-1. Printer Filter

A number of "model" interface programs in the form of shell scripts are provided in */usr/spool/lp/model*, including a generic dumb printer interface, *dumb*. Edit this script to meet the particular needs of your printer. The following information should help you to create your own printer filter.

When *lp* routes an output request to a printer, the filter for the printer is invoked by *lp* in the directory */usr/spool/lp* as follows:

```
interface printer id user title copies options file
```

The filter takes the following arguments:

printer	printer name
id	request ID returned by *lp*
user	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	list of class- and printer-dependent options specified by the user
file	full pathname of a file to be printed

When you invoke the filter, its standard input comes from */dev/null*, and both the standard output and standard error output are directed to the printing device.

Filters format their output by using command line arguments. Your filter must have the proper terminal characteristics (such as baud rate) included in its command lines. Add lines containing the *stty* command in the following form:

```
stty options <&1
```

This command takes the standard input for *stty* from the device. For more information on the *stty* command, see *stty*(1). Since different printers have different numbers of columns, make sure that the header and trailer for your filter correspond to your printer's specifications.

When printing is complete, it is the responsibility of your filter to exit with a code that shows the status of the print job.

Exit codes are interpreted by *lpsched* as follows:

0	The print job has completed successfully.
1–127	A problem was encountered in printing the request. *lpsched* notifies the sender by mail that there was a printing error. Subsequent jobs are not affected.
> 127	These codes are reserved for internal use by *lpsched.*

When problems occur that are likely to affect subsequent print jobs, you should have your filter disable the printer so that requests are not lost. When a busy printer is disabled, the filter exits with code 15.

This chapter covers the following topics:

- The terms used in discussing TTY management
- How the TTY system works
- How to tell what line settings are defined
- How to create new line settings and hunt sequences
- How to modify TTY line characteristics
- How to set terminal options

10.3 The TTY System

The remaining sections in this chapter describe how the TTY system operates, and how you can administer it.

10.3.1 How the TTY System Works

A series of four processes *init*(1M), *getty*(1M), *login*(1), *sh*(1), or *csh(1)* connects a user to the IRIX system. *init* is a general process spawner that is invoked as the last step in the boot procedure. It spawns a *getty* process for each line that a user may log in on, guided by instructions in */etc/inittab*. An argument required by the *getty* command is *line*. The TTY line argument is the name of a special file in the */dev* directory. For a description of other arguments that may be used with *getty* see the *IRIS-4D System Administrator's Reference Manual.*

A user attempting to make a connection generates a request-to-send signal that is routed by the hardware to the *getty* process for one of the TTY line files in */dev*. (We're omitting how the signal gets from the user's terminal to the workstation.) *getty* responds by sending an entry from file */etc/gettydefs* down the line. The *gettydefs* entry used depends on the *speed* argument used with the *getty* command. (In the SYNOPSIS of the *getty*(1M) command the argument name is *speed*, but it is really a pointer to the **label** field of a *gettydefs* entry.) If no *speed* argument is provided, *getty* uses the first entry in *gettydefs*. Among the fields in the *gettydefs* entry (described later in this chapter) is the login prompt.

On receiving the login prompt, the user enters a login name. *getty* starts *login*, using the login name as an argument. *login* issues the prompt for a password, evaluates the user's response, and assuming the password is acceptable, calls in the user's shell as listed in the */etc/passwd* entry for the login name. If no shell is named, */bin/sh* is furnished by default. *login* also executes */etc/profile*.

/bin/sh executes the user's *.profile*, if it exists. *.profile* or *.login* often contain *stty* commands that reset terminal options that differ from the defaults. The connection between the user and the IRIX system has now been made.

10.3.2 How to Tell What Line Settings Are Defined

You have two ways to check line settings.

1. Through the System Administration Menus, specifically the *sysadm*(1) *lineset* subcommand. *sysadm lineset* first shows the full range of line settings, then gives you the chance to examine a line in detail.
2. By looking directly in */etc/gettydefs*.

The */etc/gettydefs* file contains information used by the *getty*(1M) command to establish the speed and terminal settings for a line. The general format of the *gettydefs* file is:

```
label# initial-flags # final-flags #login-prompt #next-label
```

The following example shows a few lines from a *gettydefs* file:

```
console# B9600 # B9600 SANE TAB3 #\r\n\n$HOSTNAME console login: #console

co_9600# B9600 # B9600 SANE TAB3 #\r\n\n$HOSTNAME login: #co_4800

co_4800# B4800 # B4800 SANE TAB3 #\r\n\n$HOSTNAME login: #co_2400

co_2400# B2400 # B2400 SANE TAB3 #\r\n\n$HOSTNAME login: #co_1200

co_1200# B1200 # B1200 SANE TAB3 #\r\n\n$HOSTNAME login: #co_300

co_300# B300 # B300 SANE TAB3 #\r\n\n$HOSTNAME login: #co_9600

dx_9600# B9600 # B9600 SANE TAB3 HUPCL #\r\n\n$HOSTNAME login: #dx_9600
```

The entries shown in Figure 10-1 form a single, circular hunt sequence; the last field on each line is the label of the next line. The next-label field for the last line shown points back to the first line in the sequence. The object of the hunt sequence is to link a range of line speeds. If you see garbage characters instead of a clear login prompt, entering a BREAK causes *getty* to step to the next entry in the sequence. The hunt continues until the baud rate of the line matches the speed of the user's terminal.

The flag fields shown have the following meanings:

B300-B19200	The baud rate of the line.
HUPCL	Hang up on close.
SANE	A composite flag that stands for a set of normal line characteristics.
IXANY	Allow any character to restart output. If this flag is not specified, only DC1 (CTL-Q) will restart output.
TAB3	Send tabs to the terminal as spaces.

For a description of all *getty* flags, see *termio*(7).

10.3.3 How to Create New Line Settings and Hunt Sequences

You have two ways to do this.

1. Use the System Administration Menus, specifically the *sysadm mklineset*(1) subcommand. *sysadm mklineset* leads you through a series of prompts. Your responses make up the information for a new *gettydefs* entry.
2. By using *ed*(1) or *vi*(1) to edit */etc/gettydefs*.

Create new lines for the *gettydefs* file by following the example shown above. Each entry in the file is followed by a blank line. After editing the file run the command:

```
/etc/getty -c /etc/gettydefs
```

This causes *getty* to scan the file and print the results on your terminal. If there are any unrecognized modes or improperly constructed entries, they are reported.

10.3.4 How to Modify TTY Line Characteristics

You have two ways to do modify TTY line characteristics.

1. Use the System Administration Menus, specifically the *sysadm modtty*(1) subcommand. *sysadm modtty* leads you through a series of prompts. Your responses edit a "getty" entry in */etc/inittab.*
2. By using an editor, such as *vi*(1), to edit */etc/inittab.*

The */etc/inittab* file contains instructions for the */etc/init*(1M) command. The general format of a line entry in the */etc/inittab* file is as follows.

```
identification:level:action:process
```

The four colon-separated fields are as follows.

identification	A unique one- or two-character identifier for the line entry.
level	The run-level in which the entry is to be performed.
action	How */etc/init* treats the process field (refer to the *inittab*(4) manual page for complete information).
process	The shell command to be executed.

/etc/inittab contains several entries that spawn *getty* processes. The following example is a selection of such entries *grep*'ped from a sample */etc/inittab.*

```
t1:23:respawn:/etc/getty -s console ttyd1 co_9600
t2:23:respawn:/etc/getty ttyd2 co_9600
```

There are at least three things you might want to do to an *inittab* entry for a TTY line:

1. Change the action. Two actions that apply to TTY lines are "respawn" and "off" (see the *inittab*(4) manual page for complete information on this field).
2. Add or change arguments to */etc/getty* in the process field. A frequently used argument is *–tnn*. This tells *getty* to hang up if nothing is received within *nn* seconds. It's good practice to use the **–t** argument on dial-up lines.
3. Add or change comments. Comments can be inserted after a semi-colon (;) to end the command, and a pound sign (#) to start the comments.

10.3.5 How to Set Terminal Options

The TTY system described thus far establishes a basic style of communication between the user's terminal and the IRIX operating system. Once the user has successfully logged in, there may be terminal options that would be preferable to ones in the default set.

The command that is used to control terminal options is *stty*(1). Many users add an *stty* command to their *.profile* so the options they want are automatically set as part of the *login* process. Here is an example of a simple *stty* command.

```
stty cr0 nl0 echoe -tabs erase ^H
```

The options in the example mean:

cr0 nl0	No delay for carriage return or new line. Delays are not used on a video display terminal, but are necessary on some printing terminals to allow time for the mechanical parts of the equipment to move.
echoe	Erases characters as you backspace.
–tabs	Expand tabs to spaces when printing.
erase ^H	Change the character-delete character to a ^H. The default character-delete character is the pound sign (#). Most terminals transmit a ^H when the **<backspace>** key is pressed. Specifying this option makes the **<backspace>** key useful.

10.4 Serial Ports

This section outlines the serial support on IRIS-4D Series workstations and servers and provides instructions for connecting peripheral devices to the serial ports. This section covers only serial support for the standard serial ports, **1** and **2**, IRIS-4D 60 series systems; and **1**, **2**, **3**, and **4** on IRIS-4D 20, 50, 60T, 70, 80, 85, 100, 200, and 300 series systems.

For information on the optional serial ports, see *Using the 6-Port RS-232 Option.*

10.4.1 Defining the Serial Interface

The IRIS workstation or server provides an RS-232-compatible serial interface. All serial data cables that you connect to the computer should be shielded. The computer can easily drive and receive signals on a 50-foot cable, and it typically drives and receives signals on a cable up to 200 feet long.

There are two types of serial interface equipment available: *Data Terminal Equipment* (DTE) and *Data Communications Equipment* (DCE). The primary difference between DTE and DCE is the designation of several pins on the connector. For example, DTEs transmit on pin 2 and receive on pin 3. DCEs transmit on pin 3 and receive on pin 2. You can connect a DTE interface directly to a DCE interface.

To connect either a DCE to a DCE, or a DTE to a DTE, use a null modem cable. A null modem cable has the wires to pins 2 and 3 swapped in one connector, and may have other swapped wires as well. A signal on pin 2 at one end appears on pin 3 at the other end, and vice-versa.

The serial ports for IRIS-4D Series workstations and servers are all configured as DTE. Most terminals are also configured as DTE. Therefore, to connect a terminal to the workstation, use a cable that has pins 2 and 3 swapped in one connector. To connect a modem to the workstation, use a cable that connects each pin of the serial port to the corresponding pin of the modem. No signals need to be swapped. Connect other peripheral devices according to the configuration data provided with the device.

Silicon Graphics, Inc., provides three kinds of special files, which determine which driver is used on each port. The special files *ttyd1* and *ttyd2* are used for devices such as terminals; the files *ttym1* and *ttym2* are used for modems; and *ttyf1* and *ttyf2* are used for flow control to devices that understand hardware flow control.

Table 10-1 shows the signals supported on each port, as well as the standard 25-pin configuration.

9-Pin	Standard 25-Pin	Abbreviation	Signal
2	2	TD	Transmit Data
3	3	RD	Receive Data
4	4	RTS	Request To Send
5	5	CTS	Clear To Send
7	7	GND	Signal Ground
8	8	DCD	Data Carrier Detect
9	20	DTR	Data Terminal Ready

Table 10-1. Pins Supported on Serial Ports

Table 10-2 shows the signals and devices supported on each serial port.

Port Number	Software Port	Signals Supported (Standard 25-Pin)
1	ttyd1	TD, RD, GND
2	ttyd2	
3	ttyd3	
4	ttyd4	
1	ttym1	2,3,4†,7,8,20
2	ttym2	
3	ttym3	
4	ttym4	
1	ttyf1	2,3,4‡,5,7,8,20
2	ttyf2	
3	ttyf3	
4	ttyf4	

Table 10-2. Device Types and Pins

Note: DTR is positive when the device is open. †RTS is the same as DTR for *ttyd[1,2,3,4]* and *ttym[1,2,3,4]*. ‡RTS means "o.k. to send" for *ttyf[1,2,3,4]*.

10.4.2 Cabling the Serial Ports

This chapter describes the cables typically used to connect the computer to terminals, printers, and modems.

The serial ports are designed to connect directly to *Data Communications Equipment* (DCE) devices such as modems, via a modem cable. Each wire on this cable connects the same pin of the computer to the modem, i.e., it has a pin-to-pin correspondence. Table 10-3 shows the pin definitions for the computer and for the modem cable.

9-Pin	Modem	Signals
1	-	Data Set Ready
2	2	Transmit Data
3	3	Receive Data
4	4	Request To Send
5	5	Clear To Send
7	7	Signal Ground
8	8	Data Carrier Detect
9	20	Data Terminal Ready

Table 10-3. Pin Definitions for a Modem Cable

Note: Note that on a standard DB-25, pin 1 is chassis ground. Pin 5 is not supported on 4D60 series systems.

Connecting to *Data Terminal Equipment* (DTE) devices, such as terminals and printers, requires a different cable arrangement: a *null modem cable.* A null modem cable has the wires for pin 2 and pin 3 swapped in one connector, and may have other wires swapped as well. Table 10-4 lists the pin definitions for an example of a null modem cable. Connect the pins that are shown separated by commas in the 'DTE Device' column. Then connect these pins to the pin shown in the '9-Pin' column.

9-Pin	DTE Device	Signal
1	-	Data Set Ready
2	3	Transmit Data
3	2	Receive Data
4	5	Request To Send, Clear To Send
5	4	Clear To Send, Request To Send
7	7	Signal Ground
9	6,8*	Data Terminal Ready, Data Set Ready + Data Carrier Detect
8	20	Data Carrier Detect, Data Terminal Ready

Table 10-4. Pin Definitions for a Null Modem Cable

Note: *This connection may be necessary on the terminal side. Pin 5 is not supported on 4D60 systems.

Most terminals do not require the various handshaking lines such as *Clear To Send* or *Data Set Ready*, and work with a *three-wire null modem cable.* You need to swap the signals for pins 2 and 3, and you need to connect pin 7 of the computer to pin 7 of the terminal. Table 10-5 lists the pin definitions for a three-wire null modem cable.

Computer	Terminal	Signals
2	3	Transmit Data
3	2	Receive Data
7	7	Signal Ground

Table 10-5. Sample of a Three-wire Null Modem Cable for Terminals

Table 10-6 shows pin signals for the computer and printer. Table 10-7 shows the pins typically used by printers. Refer to your printer manual for pin specifications for printer cabling, since many printers are different.

9-Pin	Standard 25-Pin	Abbreviation	Signal
1	1	FG	Chassis Ground
2	2	TD	Transmit Data
3	3	RD	Receive Data
4	4	RTS	"I'm caught up" for *ttyf[1,2]* "I'm here" for *ttym[1,2]*
5*	5	CTS	"O.K. to send"
7	7	SG	Signal Ground
8	8	DCD	The other end is alive (if it's not connected, assumes the other end is alive)
9	20	DTR	"I'm here"

Table 10-6. Pin Signals for the Computer and Printer

Note: *Pin 5 is not supported on 4D60 series systems.

Standard 25-Pin	Signal
2	Transmit data to printer
3	Optional XON/XOFF flow control
4,6,8, or 20	"I'm alive"
5,6, or 20	"I'm caught up"
7	Ground

Table 10-7. Pin Signals Typically Used by Printers

11. UUCP

UUCP, also called the Basic Networking Utilities, lets computers using the UNIX/IRIX operating systems communicate with each other and with remote terminals. These utilities range from those used to copy files between computers (*uucp* and *uuto*) to those used for remote login and command execution (*cu*, *ct*, and *uux*).

As an administrator, you need to be familiar with the administrative tools, logs, and database files used by the Basic Networking Utilities. This chapter goes into detail about the Basic Networking Utilities files, directories, daemons, and commands.

11.1 Networking Hardware

Before your computer can communicate with other computers, you must set up the hardware to complete the communications link. The cables and other hardware you will need depend on how you want to connect the computers: direct links, telephone lines, or local area networks.

Direct Links

You can create a direct link to another computer by running cables between serial ports on the two computers. Direct links are useful where two computers communicate regularly and are physically close—within 50 feet of each other. You can use a limited distance modem to increase this distance somewhat. Transfer rates of up to 19200 bits per second (bps) are possible when computers are directly linked.

Telephone Lines

Using an Automatic Call Unit (ACU), your computer can communicate with other computers over standard phone lines. The ACU dials the telephone number requested by the networking utilities. The computer it is trying to contact must have a telephone modem capable of answering incoming calls.

Local Area Network

A Local Area Network (LAN) can be the communication medium for basic networking. Once your computer is established as a node on a LAN, it will be able to contact any other computer connected to the LAN.

11.2 Networking Commands

Basic networking programs can be divided into two categories: user programs and administrative programs. The following paragraphs describe the programs in each category.

11.2.1 User Programs

The user programs for basic networking are in */usr/bin*. No special permission is needed to use these programs. These commands are all described in the *IRIX User's Reference Manual.*

cu — Connects your computer to a remote computer so you can be logged in on both at the same time, allowing you to transfer files or execute commands on either computer without dropping the initial link.

ct — Connects your computer to a remote terminal so the user of the remote terminal can log in. The user of a remote terminal can call the computer and request that the computer call it back. In this case, the computer drops the initial link so that the remote terminal's modem will be available when it is called back.

uucp — Lets a user copy a file from one computer to another. It creates work files and data files, queues the job for transfer, and calls the *uucico* daemon, which in turn attempts to contact the remote computer.

uuto — Copies files from one computer to a public spool directory on another computer (*/usr/spool/uucppublic/receive*). Unlike *uucp*, which lets you copy a file to any accessible directory on the remote computer, *uuto* places the file in an appropriate spool directory and tells the remote user to pick it up with *uupick.*

uupick — Retrieves the files placed under */usr/spool/uucppublic/receive* when files are transferred to a computer using *uuto.*

uux Creates the work, data, and execute files needed to execute commands on a remote computer. The work file contains the same information as work files created by *uucp* and *uuto*. The execute files contain the command string to be executed on the remote computer and a list of the data files. The data files are those files required for the command execution.

uustat Displays the status of requested transfers (*uucp*, *uuto*, or *uux*). It also provides you with a means of controlling queued transfers.

11.2.2 Administrative Programs

Most of the administrative programs are in */usr/lib/uucp*, along with basic networking database files and shell scripts. The only exception is *uulog*, which is in */usr/bin*. These commands are described in the *IRIX System Administrator's Reference Manual.*

You should use the *uucp* login ID when you administer the Basic Networking Utilities because it owns the basic networking and spooled data files. The home directory of the *uucp* login ID is */usr/lib/uucp*. (The other basic networking login ID is *uucp*, used by remote computers to access your computer. Calls from *nuucp* are answered by *uucico*.)

uulog Displays the contents of a specified computer's log files. Log files are created for each remote computer your computer communicates with. The log files contain records of each use of *uucp*, *uuto*, and *uux*.

uucleanup Cleans up the spool directory. It is normally executed from a shell script called *uudemon.cleanup*, which is started by *cron*.

Uutry Tests call processing capabilities and does a moderate amount of debugging. It invokes the *uucico* daemon to establish a communication link between your computer and the remote computer you specify.

uucheck Checks for the presence of basic networking directories, programs, and support files. It can also check certain parts of the *Permissions* file for obvious syntactic errors.

11.3 Daemons

There are three daemons in the Basic Networking Utilities. A daemon is a routine that runs as a background process and performs a system-wide public function. These daemons handle file transfers and command executions. They can also be run manually from the shell.

uucico — Selects the device used for the link, establishes the link to the remote computer, performs the required login sequence and permission checks, transfers data and execute files, logs results, and notifies the user by *mail* of transfer completions. When the local *uucico* daemon calls a remote computer, it "talks" to the *uucico* daemon on the remote computer during the session.

The *uucico* daemon is executed by *uucp*, *uuto*, and *uux* programs, after all the required files have been created, to contact the remote computer. It is also executed by the *uusched* and *Uutry* programs.

uuxqt — Executes remote execution requests. It searches the spool directory for execute files (always named X.**file**) that have been sent from a remote computer. When an X.**file** file is found, *uuxqt* opens it to get the list of data files that are required for the execution. It then checks to see if the required data files are available and accessible. If the files are present and can be accessed, uuxqt checks the *Permissions* file to verify that it has permission to execute the requested command. The *uuxqt* daemon is executed by the *uudemon.hour* shell script, which is started by *cron.*

uusched — Schedules the queued work in the spool directory. Before starting the *uucico* daemon, *uusched* randomizes the order in which remote computers will be called. uusched is executed by a shell script called *uudemon.hour*, which is started by *cron.*

11.3.1 Internal Programs

uugetty This program is very similar to the getty program except it permits a line (port) to be used in both directions. *uugetty* is executed as a function of the *init* program and is described in the *IRIX System Administrator's Reference Manual.*

11.4 Supporting Data Base

The Basic Networking Utilities support files are in the /usr/lib/uucp directory. Most changes to these files can be made using the System Administration Menu commands. The descriptions below, however, provide details on the structure of these files so you can edit them manually.

Devices — Contains information concerning the location and line speed of the automatic call unit, direct links, and network devices.

Dialers — Contains character strings required to negotiate with network devices (automatic calling devices) in the establishment of connections to remote computers (non 801-type dialers).

Systems — Contains information needed by the *uucico* daemon and the *cu* program to establish a link to a remote computer. It contains information such as the name of the remote computer, the name of the connecting device associated with the remote computer, when the computer can be reached, telephone number, login ID, and password.

Dialcodes — This file contains dial-code abbreviations that may be used in the phone number field of *Systems* file entries.

Permissions — This file defines the level of access that is granted to computers when they attempt to transfer files or remotely execute commands on your computer.

Poll — This file defines computers that are to be polled by your system and when they are polled.

Sysfiles — This file is used to assign different or multiple files to be used by uucico and *cu* as *Systems*, *Devices*, and *Dialers* files.

11.4.1 Devices File

The *Devices* file (*/usr/lib/uucp/Devices*) contains information for all the devices that may be used to establish a link to a remote computer, devices such as automatic call units, direct links, and network connections.

Note: This file works closely with the *Dialers*, *Systems*, and *Dialcodes* files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to a related entry in another file.

Each entry in the *Devices* file has the following format:

```
Type Line Line2 Class Dialer-Token-Pairs
```

Each of these fields is defined in the following section.

Type This field may contain one of two keywords (**Direct** or **ACU**), the name of a Local Area Network switch, or a system name.

Direct This keyword indicates a Direct Link to another computer or a switch (for *cu* connections only).

ACU This keyword indicates that the link to a remote computer is made through an automatic call unit (Automatic Dial Modem). This modem may be connected either directly to your computer or indirectly through a Local Area Network (LAN) switch.

LAN_Switch
This value can be replaced by the name of a LAN switch. **micom** and **develcon** are the only ones for which there are caller scripts in the *Dialers* file. You can add your own LAN switch entries to the *Dialers* file.

Sys-Name
This value indicates a direct link to a particular computer. (**Sys-Name** is replaced by the name of the computer.) This naming scheme is used to convey the fact that the line associated with this *Devices* entry is for a particular computer in the *Systems* file.

The keyword used in the **Type** field is matched against the third field of *Systems* file entries as shown below:

```
Devices:   ACU tty11 - 1200 penril

Systems: eagle Any ACU 1200 3251 ogin: nuucp \
                ssword: Oakgrass
```

You can designate a protocol to use for a device within this field. See the Protocols section at the end of the description of this file.

Line This field contains the device name of the line (port) associated with the *Devices* entry. For instance, if the Automatic Dial Modem for a particular entry was attached to the */dev/tty11* line, the name entered in this field would be **tty11**.

Line2 If the keyword **ACU** was used in the **Type** field and the ACU is an 801 type dialer, **Line2** would contain the device name of the 801 dialer. (801 type ACUs do not contain a modem. Therefore, a separate modem is required and would be connected to a different line, defined in the **Line** field.) This means that one line would be allocated to the modem and another to the dialer. Since non-801 dialers will not normally use this configuration, the **Line2** field will be ignored by them, but it must still contain a hyphen (-) as a placeholder.

Class If the keyword **ACU** or **Direct** is used in the Type field, **Class** may be just the speed of the device. However, it may contain a letter and a speed (for example, C1200, D1200) to differentiate between classes of dialers (Centrex or Dimension PBX). This is necessary because many larger offices may have more than one type of telephone network: one network may be dedicated to serving only internal office communications while another handles the external communications. In such a case, it becomes necessary to distinguish which line(s) should be used for internal communications and which should be used for external communications.

The keyword used in the **Class** field of the *Devices* file is matched against the fourth field of *Systems* file entries as shown below:

```
Devices: ACU ttyl1 - D1200 penril

Systems: eagle Any ACU D1200 3251 ogin: nuucp \
          ssword: Oakgrass
```

Some devices can be used at any speed, so the keyword Any may be used in the **Class** field. If **Any** is used, the line will match any speed requested in a *Systems* file entry. If this field is **Any** and the *Systems* file **Class** field is **Any**, the speed defaults to 1200 bps.

Dialer-Token-Pairs:
This field contains pairs of dialers and tokens. The *dialer* portion may be the name of an automatic dial modem, a LAN switch, or it may be **direct** for a Direct Link device. You can have any number of Dialer-Token-Pairs. The **token** portion may be supplied immediately following the *dialer* portion or if not present, it will be taken from a related entry in the *Systems* file.

This field has the format:

dialer token dialer token

where the last pair may or may not be present, depending on the associated device (dialer). In most cases, the last pair contains only a *dialer* portion and the **token** portion is retrieved from the Phone field of the *Systems* file entry. A valid entry in the *dialer* portion may be defined in the *Dialers* file.

The *Dialer-Token-Pairs* (**DTP**) field may be structured four different ways, depending on the device associated with the entry:

1. If an automatic dialing modem is connected directly to a port on your computer, the **DTP** field of the associated Devices file entry will only have one pair. This pair would normally be the name of the modem. This name is used to match the particular *Devices* file entry with an entry in the *Dialers* file. Therefore, the *dialer* field must match the first field of a Dialers file entry as shown below:

```
Devices:  ACU tty11 - 1200 ventel

Dialers:  ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

Notice that only the *dialer* portion (**ventel**) is present in the DTP field of the *Devices* file entry. This means that the **token** to be passed on to the dialer (in this case the phone number) is taken from the **Phone** field of a Systems file entry. (\T is implied, see below.) Backslash sequences are described below.

2. If a direct link is established to a particular computer, the **DTP** field of the associated entry would contain the keyword **direct**. This is true for both types of direct link entries, **Direct** and *System-Name* (refer to discussion on the **Type** field).

3. If a computer with which you wish to communicate is on the same local network switch as your computer, your computer must first access the switch and the switch can make the connection to the other computer. In this type of entry, there is only one pair. The *dialer* portion is used to match a *Dialers* file entry as shown below:

```
Devices:  develcon tty13 - 1200 develcon \D

Dialers: develcon "" "" \pr\ps\c est:\007 \E\D\e \007
```

As shown, the **token** portion is left blank, which indicates that it is retrieved from the *Systems* file. The *Systems* file entry for this particular computer will contain the token in the **Phone** field, which is normally reserved for the phone number of the computer (refer to *Systems* file, **Phone** field). This type of **DTP** contains an escape character (**\D**), which ensures that the contents of the **Phone** field will not be interpreted as a valid entry in the Dialcodes file.

4. If an automatic dialing modem is connected to a switch, your computer must first access the switch and the switch will make the connection to the automatic dialing modem. This type of entry requires two *dialer-token-pairs*. The *dialer* portion of each pair (fifth and seventh fields of entry) will be used to match entries in the *Dialers* file as shown below:

```
Devices:  ACU tty14 - 1200 develcon vent ventel

Dialers:  develcon "" "" \pr\ps\c est:\007 \E\D\e \007
Dialers:  ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

   In the first pair, **develcon** is the dialer and **vent** is the token that is passed to the Develcon switch to tell it which device (ventel modem) to connect to your computer. This token would be unique for each LAN switch since each switch may be set up differently. Once the ventel modem has been connected, the second pair is accessed, where ventel is the dialer and the token is retrieved from the *Systems* file.

There are two escape characters that may appear in a **DTP** field:

\T Indicates that the **Phone (token)** field should be translated using the *Dialcodes* file. This escape character is normally placed in the *Dialers* file for each caller script associated with an automatic dial modem (penril, ventel, etc.). Therefore, the translation will not take place until the caller script is accessed.

\D Indicates that the **Phone (token)** field should not be translated using the *Dialcodes* file. If no escape character is specified at the end of a *Devices* entry, the **\D** is assumed (default). A **\D** is also used in the *Dialers* file with entries associated with network switches (develcon and micom).

Protocols

You can define the protocol to use with each device. In most cases it is not needed since you can use the default or define the protocol with the particular System you are calling (see Systems file, type field). If you do specify the protocol, you must do in the form **Type,Protocol**. Available protocols are:

g This protocol is slower and more reliable than **e**. It is good for transmission over noisy telephone lines.

e This protocol is faster than **g**, but it assumes error-free transmission.

For reliable local area networks, you should use the **e** protocol.

11.4.2 Dialers File

The *Dialers* file (/usr/lib/uucp/Dialers) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

As shown in the above examples, the fifth field in a Devices file entry is an index into the *Dialers* file or a special dialer type. Here an attempt is made to match the fifth field in the *Devices* file with the first field of each *Dialers* file entry. In addition, each odd numbered *Devices* field starting with the seventh position is used as an index into the *Dialers* file. If the match succeeds, the *Dialers* entry is interpreted to perform the dialer negotiations. Each entry in the *Dialers* file has the following format:

```
dialer substitutions expect-send...
```

The *dialer* field matches the fifth and additional odd numbered fields in the *Devices* file. The **substitutions** field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for "wait for dialtone" and "pause."

The remaining **expect-send** fields are character strings. Below are some character strings distributed with the Basic Networking Utilities in the *Dialers* file.

```
penril =W-P "" \d > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
ventel =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
hayes  =,-, "" \dAT\r\c OK\r \EATDT\T\r\c CONNECT
rixon  =&-% "" \d\r\r\c $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
vadiac =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
develcon "" "" \pr\ps\c est:\007 \E\D\e \007
micom ""     "" \s\c NAME? \D\r\c GO
direct
att2212c        =+-,     "" \r\c :--: atol2=y,T\T\r\c red
att4000 =,-,    "" \033\r\r\c DEM: \033s0401\c \006 \033s0901\c \
        \006 \033s1001\c \006 \033s1102\c \006 \033dT\T\r\c \006
att2224 =+-,    "" \r\c :--: T\T\r\c red
nls     ""      "" NLPS:000:001:1\N\c
```

The meaning of some of the escape characters (those beginning with "\") used in the *Dialers* file are listed below:

`\p`	pause (approximately ¼ to ½ second)
`\d`	delay (approximately 2 seconds)
`\D`	phone number or token without *Dialcodes* translation
`\T`	phone number or token with *Dialcodes* translation
`\K`	insert a BREAK
`\E`	enable echo checking (for slow devices)
`\e`	disable echo checking
`\r`	carriage return
`\c`	no new-line or carriage return
`\n`	send new-line
`\nnn`	send octal number.

Additional escape characters that may be used are listed in the section discussing the *Systems* file.

The penril entry in the *Dialers* file is executed as follows. First, the phone number argument is translated, replacing any = with a W (wait for dialtone) and replacing any - with a **P** (pause).

The handshake given by the remainder of the line works as follows:

`""`	Wait for nothing. (In other words, proceed to the next thing.)
`\d`	Delay for 2 seconds.
`>`	Wait for a **>**.
`s\p9\c`	Send an **s**, pause for ½ second, send a **9**, send no terminating new-line
`)-W\p\r\ds\p9\c-)`	Wait for a). If it is not received, process the string between the - characters as follows. Send a **W**, pause, send a carriage-return, delay, send an **s**, pause, send a **9**, without a new-line, and then wait for the **)**.
`y\c`	Send a **y**.
`:`	Wait for a **:**.
`\E\TP`	Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the *Dialcodes* translation and the modem function translation specified by field 2 of this entry. Then send a **P**.
`>`	Wait for a **>**.
`9\c`	Send a **9** without a new-line.
`OK`	Waiting for the string **OK**.

11.4.3 Systems File

The *Systems* file (*/usr/lib/uucp/Systems*) contains the information needed by the *uucico* daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that can be called by your computer. In addition, the basic networking software can be configured to prevent any computer that does not appear in this file from logging in on your computer (refer to the section "Other Networking Files" in this chapter for a description of the **remote.unknown** file). More than one entry may be present for a particular computer. The additional entries represent alternative communication paths that will be tried in sequential order. The management of this file is supported by the System Administration Menu subcommand *systemmgmt*.

Using the *Sysfiles*, you can define several files to be used as "Systems" files. See the description of the *Sysfiles* file for details. Each entry in the *Systems* file has the following format:

System-Name Time Type Class Phone Login

Each of these fields is defined in the following section.

System-name
This field contains the node name of the remote computer.

Time This field is a string that indicates the day-of-week and time-of-day when the remote computer can be called. The format of the **Time** field is:

daytime[*;retry*]

The day portion may be a list containing some of the following, possibly compounded with comma delimiters:

Su Mo Tu We Th Fr Sa for individual days

Wk for any week-day (Mo Tu We Th Fr)

Any for any day

Never for a passive arrangement with the remote computer. If the **Time** field is **Never**, your computer will never initiate a call to the remote computer. The call must be initiated by the remote computer. In other words, your computer is in a passive mode in respect to the remote computer (see discussion of *Permissions* file).

Here is an example:

```
7Wk 1700-0800, Sa, Su
```

This example allows calls from 5:00 p.m. to 8:00 am, Monday through Thursday, and calls any time Saturday and Sunday. The example would be an effective way to call only when phone rates are low, if immediate transfer is not critical.

The **time** portion should be a range of times such as 0800-1230. If no **time** portion is specified, any time of day is assumed to be allowed for the call. A time range that spans 0000 is permitted. For example, **0800-0600** means all times are allowed other than times between 6 a.m. and 8 a.m. An optional subfield, **retry**, is available to specify the minimum time (in minutes) before a retry, following a failed attempt. The default wait is 60 minutes. The subfield separator is a semicolon (;). For example, **Any;9** is interpreted as call any time, but wait at least 9 minutes before retrying after a failure occurs.

Type This field contains the device type that should be used to establish the communication link to the remote computer. The keyword used in this field is matched against the first field of *Devices* file entries as shown below:

```
Systems:  eagle Any ACU,g D1200 3251 ogin: nuucp \
           ssword: Oakgrass

Devices:  ACU tty11 - D1200 penril
```

You can define the protocol used to contact the system by adding it on to the **Type** field. The example above shows how to attach the protocol **g** to the device type **ACU**. See the information under the Protocols section in the description of the *Devices* file for details.

Class This field is used to indicate the transfer speed of the device used in establishing the communication link. It may contain a letter and speed (for example, C1200, D1200) to differentiate between classes of dialers (refer to the discussion on the *Devices* file, Class field). Some devices can be used at any speed, so the keyword **Any** may be used. This field must match the **Class** field in the associated *Devices* file entry as shown below:

```
Systems:  eagle Any ACU D1200 NY3251 ogin: nuucp \
           ssword: Oakgrass

Devices:  ACU tty11 - D1200 penril
```

If information is not required for this field, use a - as a place holder for the field.

Phone This field is used to provide the phone number (token) of the remote computer for automatic dialers (LAN switches). The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the *Dialcodes* file. For example:

```
Systems:  eagle Any ACU D1200 NY3251 ogin: nuucp \
           ssword: Oakgrass

Dialcodes:  NY 9=1212555
```

In this string, an equal sign (=) tells the ACU to wait for a secondary dial tone before dialing the remaining digits. A dash in the string (-) instructs the ACU to pause 4 seconds before dialing the next digit.

If your computer is connected to a LAN switch, you may access other computers that are connected to that switch. The *Systems* file entries for these computers will not have a phone number in the **Phone** field. Instead, this field will contain the token that must be passed on to the switch so it will know which computer your computer wishes to communicate with. (This is usually just the

system name.) The associated Devices file entry should have a **\D** at the end of the entry to ensure that this field is not translated using the *Dialcodes* file.

Login This field contains login information given as a series of fields and subfields of the format:

```
expect send
```

where **expect** is the string that is received and **send** is the string that is sent when the **expect** string is received.

The **expect** field may be made up of subfields of the form:

```
expect[-send-expect]...
```

where the **send** is sent if the prior **expect** is not successfully read and the **expect** following the **send** is the next expected string. For example, with *login--login*, UUCP will expect *login*. If UUCP gets *login*, it will go on to the next field. If it does not get *login*, it will send nothing followed by a new line, then look for *login* again. If no characters are initially expected from the remote computer, the characters "" (null string) should be used in the first **expect** field. Note that all **send** fields will be sent followed by a new-line unless the **send** string is terminated with a **\c**.

Here is an example of a *Systems* file entry that uses an expect-send string:

```
owl Any ACU 1200 Chicago6013 "" \r ogin:-BREAK-ogin: \
uucpx word: xyzzy
```

This example says send a carriage return and wait for *login:* (for `Login:`). If you don't get `ogin`, send a **BREAK**. When you do get `ogin:` send the login name *uucpx*, then when you get word: (for Password:), send the password **xyzzy**.

There are several escape characters that cause specific actions when they are a part of a string sent during the login sequence. The following escape characters are useful in UUCP communications:

`\N` Send or expect a null character (ASCII NUL).

`\b`	Send or expect a backspace character.
`\c`	If at the end of a string, suppress the new-line that is normally sent. Ignored otherwise.
`\d`	Delay two seconds before sending or reading more characters.
`\p`	Pause for approximately ¼ to ½ second.
`\E`	Start echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.)
`\e`	Echo check off.
`\n`	Send a new-line character.
`\r`	Send or expect a carriage-return.
`\s`	Send or expect a space character.
`\t`	Send or expect a tab character.
`\\`	Send or expect a \ character.
`EOT`	Send or expect EOT new-line twice.
`BREAK`	Send or expect a break character.
`\K`	Same as BREAK.
`\ddd`	Collapse the octal digits (ddd) into a single character.

11.4.4 Dialcodes File

The *Dialcodes* file (*/usr/lib/uucp/Dialcodes*) contains the dial-code abbreviations that can be used in the **Phone** field of the *Systems* file. Each entry has the format:

```
abb dial-seq
```

where **abb** is the abbreviation used in the *Systems* file Phone field and *dial-seq* is the dial sequence that is passed to the dialer when that particular *Systems* file entry is accessed.

The entry

```
jt 9=847-
```

would be set up to work with a **Phone** field in the *Systems* file such as `jt7867`. When the entry containing `jt7867` is encountered, the sequence 9=847-7867 would be sent to the dialer if the token in the dialer-token-pair is `\T`.

11.4.5 Permissions File

The *Permissions* file (*/usr/lib/uucp/Permissions*) specifies the permissions that remote computers have with respect to login, file access, and command execution. There are options that restrict the remote computer's ability to request files and its ability to receive files queued by the local site. Another option is available that specifies the commands that a remote site can execute on the local computer.

How Entries are Structured

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name/value pair in the following format:

```
name=value
```

Note that no white space is allowed within an option assignment.

Comment lines begin with a "#" and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of *Permissions* file entries:

LOGNAME Specifies the permissions that take effect when a remote computer logs in on (calls) your computer.

MACHINE Specifies permissions that take effect when your computer logs in on (calls) a remote computer.

LOGNAME entries will contain a LOGNAME option and MACHINE entries will contain a MACHINE option.

Considerations

The following items should be considered when using the Permissions file to restrict the level of access granted to remote computers:

- All login IDs used by remote computers to login for UUCP communications must appear in one and only one LOGNAME entry.
- Any site that is called whose name does not appear in a MACHINE entry, will have the following default permissions/restrictions:

  1. Local send and receive requests will be executed.
  2. The remote computer can send files to your computer's */usr/spool/uucppublic* directory.
  3. The commands sent by the remote computer for execution on your computer must be one of the default commands; usually *rmail.*

Options

This section describes each option, specifies how they are used, and lists their default values.

REQUEST When a remote computer calls your computer and requests to receive a file, this request can be granted or denied. The REQUEST option specifies whether the remote computer can request to set up file transfers from your computer. The string

```
REQUEST=yes
```

specifies that the remote computer can request to transfer files from your computer. The string

```
REQUEST=no
```

specifies that the remote computer cannot request to receive files from your computer. This is the default value. It will be used if the REQUEST option is not specified. The REQUEST option can appear in either a LOGNAME (remote calls you) entry or a MACHINE

(you call remote) entry. A note on security: When a remote machine calls you, unless you have a unique login and password for that machine you don't know if the machine is who it says it is.

SENDFILES When a remote computer calls your computer and completes its work, it may attempt to take work your computer has queued for it. The SENDFILES option specifies whether your computer can send the work queued for the remote computer.

The string

```
SENDFILES=yes
```

specifies that your computer may send the work that is queued for the remote computer as long as it logged in as one of the names in the LOGNAME option. This string is mandatory if your computer is in a "passive mode" with respect to the remote computer.

The string

```
SENDFILES=call
```

specifies that files queued in your computer will be sent only when your computer calls the remote computer. The call value is the default for the SENDFILE option. This option is only significant in LOGNAME entries since MACHINE entries apply when calls are made out to remote computers. If the option is used with a MACHINE entry, it will be ignored.

READ and WRITE

These options specify the various parts of the file system that *uucico* can read from or write to. The READ and WRITE options can be used with either MACHINE or LOGNAME entries.

The default for both the READ and WRITE options is the *uucppublic* directory as shown in the following strings:

```
READ=/usr/spool/uucppublic
WRITE=/usr/spool/uucppublic
```

The strings

```
READ=/ WRITE=/
```

specify permission to access any file that can be accessed by a local user with "other" permissions.

The value of these entries is a colon separated list of pathnames. The READ option is for requesting files, and the WRITE option for depositing files. One of the values must be the prefix of any full pathname of a file coming in or going out. To grant permission to deposit files in /usr/news as well as the public directory, the following values would be used with the WRITE option:

```
WRITE=/usr/spool/uucppublic:/usr/news
```

It should be pointed out that if the READ and WRITE options are used, all pathnames must be specified because the pathnames are not added to the default list. For instance, if the */usr/news* pathname was the only one specified in a WRITE option, permission to deposit files in the public directory would be denied.

You should be careful what directories you make accessible for reading and writing by remote systems. For example, you probably wouldn't want remote computers to be able to write over your */etc/passwd* file so */etc* shouldn't be open to writes.

NOREAD and NOWRITE

The NOREAD and NOWRITE options specify exceptions to the READ and WRITE options or defaults. The strings

```
READ=/ NOREAD=/etc WRITE=/usr/spool/uucppublic
```

would permit reading any file except those in the /etc directory (and its subdirectories—remember, these are

prefixes) and writing only to the default /usr/spool/uucppublic directory. NOWRITE works in the same manner as the NOREAD option. The NOREAD and NOWRITE can be used in both LOGNAME and MACHINE entries.

CALLBACK The CALLBACK option is used in LOGNAME entries to specify that no transaction will take place until the calling system is called back. There are two examples of when you would use CALLBACK. From a security standpoint, if you call back a machine you can be sure it is the machine it says it is. If you are doing long data transmissions, you can choose the machine that will be billed for the longer call.

The string

```
CALLBACK=yes
```

specifies that your computer must call the remote computer back before any file transfers will take place.

The default for the COMMAND option is

```
CALLBACK=no
```

The CALLBACK option is very rarely used. Note that if two sites have this option set for each other, a conversation will never get started.

COMMANDS The COMMANDS option can be hazardous to the security of your system. Use it with extreme care.

The *uux* program will generate remote execution requests and queue them to be transferred to the remote computer. Files and a command are sent to the target computer for remote execution.

The COMMANDS option can be used in MACHINE entries to specify the commands that a remote computer can execute on your computer. Note that COMMANDS is not used in a LOGNAME entry; COMMANDS in MACHINE entries define command permissions whether we call the remote system or it calls us.

The string

```
COMMANDS=rmail
```

indicates the default commands that a remote computer can execute on your computer. If a command string is used in a MACHINE entry, the default commands are overridden. For instance, the entry

```
MACHINE=owl:raven:hawk:dove \
COMMANDS=rmail:rnews:lp
```

overrides the COMMAND default so that the computers owl, raven, hawk, and dove can now execute rmail, *rnews*, and *lp* on your computer.

In addition to the names as specified above, there can be full pathnames of commands. For example,

```
COMMANDS=rmail:/usr/bin/rnews:/usr/local/lp
```

specifies that command *rmail* uses the default path. The default paths for your computer are */bin*, */usr/bin*, and */usr/bin*. When the remote computer specifies *rnews* or */usr/bin/rnews* for the command to be executed, */usr/bin/rnews* will be executed regardless of the default path. Likewise, */usr/local/lp* is the *lp* command that will be executed.

Including the ALL value in the list means that any command from the remote computer(s) specified in the entry will be executed. If you use this value, you give the remote computer full access to your computer. BE CAREFUL. This allows far more access than normal users have.

The string

```
COMMANDS=/usr/bin/rnews:ALL:/usr/local/lp
```

illustrates two points: The ALL value can appear anywhere in the string, and the pathnames specified for *rnews* and *lp* will be used (instead of the default) if the requested command does not contain the full path names for *rnews* or *lp*.

The VALIDATE option should be used with the COMMANDS option whenever potentially dangerous commands like *cat* and *uucp* are specified with the COMMANDS option. Any command that reads or writes files is potentially dangerous to local security when executed by the UUCP remote execution daemon (*uuxqt*).

VALIDATE The VALIDATE option is used in conjunction with the COMMANDS option when specifying commands that are potentially dangerous to your computer's security. It is used to provide a certain degree of verification of the caller's identity. The use of the VALIDATE option requires that privileged computers have a unique login/password for UUCP transactions. An important aspect of this validation is that the login/password associated with this entry be protected. If an outsider gets that information, that particular VALIDATE option can no longer be considered secure. (VALIDATE is merely an added level of security on top of the COMMANDS option, though it is a more secure way to open command access than ALL.)

Careful consideration should be given to providing a remote computer with a privileged login and password for UUCP transactions. Giving a remote computer a special login and password with file access and remote execution capability is like giving anyone on that computer a normal login and password on your computer. Therefore, if you cannot trust someone on the remote computer, do not provide that computer with a privileged login and password.

LOGNAME The LOGNAME entry

```
LOGNAME=uucpfriend VALIDATE=eagle:owl:hawk
```

specifies that if one of the remote computers that claims to be eagle, owl, or hawk logs in on your computer, it must have used the login *uucpfriend.* As can be seen, if an outsider gets the *uucpfriend* login/password, marauding is trivial.

But what does this have to do with the COMMANDS option, which only appears in MACHINE entries? It links the MACHINE entry (and COMMANDS option) with a LOGNAME entry associated with a privileged login. This link is needed because the execution daemon is not running while the remote computer is logged in. In fact, it is an asynchronous process with no knowledge of what computer sent the execution request. Therefore, the real question is how does your computer know where the execution files came from?

Each remote computer has its own "spool" directory on your computer. These spool directories have write permission given only to the UUCP programs. The execution files from the remote computer are put in its spool directory after being transferred to your computer. When the *uuxqt* daemon runs, it can use the spool directory name to find the MACHINE entry in the Permissions file and get the COMMANDS list, or if the computer name does not appear in the *Permissions* file, the default list will be used.

The following example shows the relationship between the MACHINE and LOGNAME entries:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
COMMANDS=rmail:/usr/bin/rnews \
READ=/  WRITE=/

LOGNAME=uucpz VALIDATE=eagle:owl:hawk \
REQUEST=yes SENDFILES=yes \
READ=/  WRITE=/
```

The value in the COMMANDS option means that remote mail and */usr/bin/rnews* can be executed by remote users.

In the first entry, you must make the assumption that when you want to call one of the computers listed, you are really calling either *eagle*, *owl*, or *hawk*. Therefore, any files put into one of the *eagle*, *owl*, or *hawk* spool directories is put there by one of those computers. If a remote computer logs in and says that it is one of these three computers, its execution files will also be put in the privileged spool directory. You therefore have to validate that the computer has the privileged login *uucpz*.

You may want to specify different option values for the computers your computer calls that are not mentioned in specific MACHINE entries. This may occur when there are many computers calling in, and the command set changes from time to time. The name "OTHER" for the computer name is used for this entry as shown below:

```
MACHINE=OTHER \
COMMANDS=rmail:rnews:/usr/bin/Photo:/usr/bin/xp
```

All other options available for the MACHINE entry may also be set for the computers that are not mentioned in other MACHINE entries.

Combining MACHINE and LOGNAME Entries

It is possible to combine MACHINE and LOGNAME entries into a single entry where the common options are the same. For example, the two entries

```
MACHINE=eagle:owl:hawk REQUEST=yes \
  READ=/  WRITE=/

LOGNAME=uucpz REQUEST=yes SENDFILES=yes \
  READ=/  WRITE=/
```

share the same REQUEST, READ, and WRITE options. These two entries can be merged as shown below:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
LOGNAME=uucpz SENDFILES=yes \
  READ=/  WRITE=/
```

11.4.6 Poll File

The *Poll* file (*/usr/lib/uucp/Poll*) contains information for polling remote computers. Each entry in the *Poll* file contains the name of a remote computer to call, followed by a <tab> character (a space won't work), and finally the hours the computer should be called. The format of entries in the *Poll* file are:

```
sys-name hour ...
```

For example the entry:

```
eagle            0 4 8 12 16 20
```

will provide polling of computer *eagle* every four hours.

The *uudemon.poll* script does not actually perform the poll. It merely sets up a polling work file (always named C.**file**), in the spool directory that will be seen by the scheduler, which is started by *uudemon.hour*.

11.4.7 Sysfiles File

The */usr/lib/uucp/Sysfiles* file lets you assign different files to be used by *uucp* and *cu* as *Systems*, *Devices*, and *Dialers* files. Here are some cases where this optional file may be useful.

- You may want different *Systems* files so requests for login services can be made to different addresses than *uucp* services.
- You may want different *Dialers* files to use different handshaking for *cu* and *uucp*.
- You may want to have multiple *Systems*, *Dialers*, and Devices files. The *Systems* file in particular may become large, making it more convenient to split it into several smaller files.

The format of the *Sysfiles* file is

```
service=w systems=x:x dialers=y:y devices=z:z
```

where *w* is replaced by *uucico*, *cu*, or both separated by a colon; *x* is one or more files to be used as the *Systems* file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the *Dialers* file; and *z* is one or more files to be used as the *Devices* file. Each file is assumed to be relative to the */usr/lib/uucp* directory, unless a full path is given. A backslash-carriage return (**\<return>**) can be used to continue an entry on to the next line.

Here's an example of using a local *Systems* file in addition to the usual *Systems* file:

```
service=uucico:cu     systems=Systems:Local_Systems
```

If this is in */usr/lib/uucp/Sysfiles*, then both *uucico* and *cu* will first look in */usr/lib/uucp/Systems*. If the system they're trying to call doesn't have an entry in that file, or if the entries in the file fail, then they'll look in */usr/lib/uucp/Local_Systems*.

When different *Systems* files are defined for *uucico* and *cu* services, your machine will store two different lists of Systems. You can print the *uucico* list using the uuname command or the *cu* list using the *uuname –c* command.

11.4.8 Other Networking Files

There are three other files that impact the use of basic networking facilities. In most cases, the default values are fine and no changes are needed. If you want to change them, however, use any standard IRIX system text editor (**ed** or **vi**).

Maxuuxqts This file defines the maximum number of *uuxqt* programs that can run at once.

Maxuuscheds This file defines the maximum number of *uusched* programs that can run at once.

remote.unknown This file is a shell script that executes when a machine that is not in any of the *Systems* starts a conversation. It will log the conversation attempt and fail to make a connection. If you change the permissions of this file so it cannot execute (*chmod 000 remote.unknown*), your system will accept any conversation requests.

11.5 Administrative Files

The basic networking administrative files are described below. These files are created in spool directories to lock devices, hold temporary data, or keep information about remote transfers or executions.

TM (temporary data file)

These data files are created by Basic Networking processes under the spool directory (i.e., */usr/spool/uucp/*X) when a file is received from another computer. The directory X has the same name as the remote computer that is sending the file. The names of the temporary data files have the format:

`TM.`*pid.ddd*

where *pid* is a process-ID and *ddd* is a sequential three digit number starting at 0.

When the entire file is received, the *TM.pid.ddd* file is moved to the pathname specified in the C.*sysnxxxx* file (discussed below) that caused the transmission. If processing is abnormally terminated, the *TM.pid.ddd* file may remain in the X directory. These files should be automatically removed by *uucleanup*.

LCK (lock file)

Lock files are created in the /usr/spool/locks directory for each device in use. Lock files prevent duplicate conversations and multiple attempts to use the same calling device. The names of lock files have the format:

`LCK..`*str*

where *str* is either a device or computer name. These files may remain in the spool directory if the communications link is unexpectedly dropped (usually on computer crashes). The lock files will be ignored (removed) after the parent process is no longer active. The lock file contains the process ID of the process that created the lock.

C. (work file) Work files are created in a spool directory when work (file transfers or remote command executions) has been queued for a remote computer. The names of work files have the format:

`C.`*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the ASCII character representing the grade (priority) of the work, and *xxxx* is the four digit job sequence number assigned by UUCP. Work files contain the following information:

- Full pathname of the file to be sent or requested
- Full pathname of the destination or userle name
- User login name
- List of options
- Name of associated data file in the spool directory. If the *uucp* **–c** or *uuto –p* option was specified, a dummy name (**D.0**) is used
- Mode bits of the source file
- Remote user's login name to be notified upon completion of the transfer

D. (Data file) Data files are created when it is specified in the command line to copy the source file to the spool directory. The names of data files have the following format:

`D.`*systmxxxxyyy*

where *systm* is the first five characters in the name of the remote computer, *xxxx* is a four-digit job sequence number assigned by *uucp*. The four digit job sequence number may be followed by a sub-sequence number, *yyy* that is used when there are several **D.** files created for a work (**C.**) file.

X. (Execute file)

Execute files are created in the spool directory prior to remote command executions. The names of execute files have the following format:

`X.`*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the character representing the grade (priority) of the work, and *xxxx* is a four digit sequence number assigned by UUCP. Execute files contain the following information:

- Requester's login and computer name.
- Name of file(s) required for execution.
- Input to be used as the standard input to the command string.
- Computer and file name to receive standard output from the command execution.
- Command string.
- Option lines for return status requests.

11.6 uucp Error Messages

11.6.1 ASSERT Error Messages

When a process is aborted the system records ASSERT error messages in */usr/spool/uucp/.Admin/errors*. These messages include the file name, *sccsid*, line number, and the text listed below. In most cases, these errors are the result of file system problems. Use the "errno" (when present) to investigate the problem. If "errno" is present in a message, it is shown as () in the following list.

Error Message	Description/Action
CAN'T OPEN	An open() or fopen() failed.
CAN'T WRITE	A write(), fwrite(), fprint(), etc. failed.
CAN'T READ	A read(), fgets(), etc. failed.
CAN'T CREATE	A create() call failed.
CAN'T ALLOCATE	A dynamic allocation failed.
CAN'T LOCK	An attempt to make a LCK (lock) file failed. In some cases, this is a fatal error.
CAN'T STAT	A stat() call failed.
CAN'T CHMOD	A chmod() call failed.
CAN'T LINK	A link() call failed.
CAN'T CHDIR	A chdir() call failed.
CAN'T UNLINK	An unlink() call failed.
WRONG ROLE	This is an internal logic problem.

Error Message	Description/Action
CAN'T MOVE TO CORRUPTDIR	An attempt to move some bad C. or X. files to the */usr/spool/uucp/.Corrupt* directory failed. The directory is probably missing or has wrong modes or owner.
CAN'T CLOSE	A close() or fclose() call failed.
FILE EXISTS	The creation of a C. or D. file is attempted, but the file exists. This occurs when there is a problem with the sequence file access. This usually indicates a software error.
No uucp server	A TCP/IP call is attempted, but there is no server for UUCP.
BAD UID	The uid cannot be found in the */etc/passwd* file. The file system is in trouble, or the /etc/passwd file is inconsistent.
BAD LOGIN_UID	The uid cannot be found in the **etc/passwd** file. The file system is in trouble, or the /etc/passwd file is inconsistent.
ULIMIT TOO SMALL	The ulimit for the current user process is too small. File transfers may fail, so transfer is not attempted.
BAD LINE	There is a bad line in the *Devices* file; there are not enough arguments on one or more lines.
FSTAT FAILED IN EWRDATA	There is something wrong with the ethernet media.
SYSLST OVERFLOW	An internal table in gename.c overflowed. A big/strange request was attempted. Contact your service representative.
TOO MANY SAVED C FILES	An internal table in gename.c overflowed. A big/strange request was attempted. Contact your service representative.
RETURN FROM fixline ioctl	An **ioctl**, which should never fail, failed. There is a system driver problem.

Error Message	Description/Action
BAD SPEED	A bad line speed appears in the Devices/*Systems* files (Class field).
PERMISSIONS file: BAD OPTION	There is a bad line or option in the Permissions file. Fix it immediately!
PKCGET READ	The remote machine probably hung up. No action is required.
PKXSTART	The remote machine aborted in a non-recoverable way. This can generally be ignored.
SYSTAT OPEN FAIL	There is a problem with the modes of */usr/lib/uucp/.Status*, or there is a file with bad modes in the directory.
TOO MANY LOCKS	There is an internal problem!
XMV ERROR	There is a problem with some file or directory. It is likely the spool directory, since the modes of the destinations were suppose to be checked before this process was attempted.
CAN'T FORK	An attempt to fork and exec failed. The current job should not be lost, but will be attempted later (*uuxqt*). No action need be taken.

11.7 STATUS Error Messages

Status error messages are messages that are stored in the */usr/spool/uucp/.Status directory. This directory contains a separate file for each remote machine that your workstation attempts to communicate with. These individual machine files contain status information on the attempted communication, whether it was successful or not. What follows is a list of the most common error messages that may appear in these files.*

Error Message	Description/Action
OK	Things are OK.
NO DEVICES AVAILABLE	*There is currently no device available for the call. Check to see that there is a valid device in the Devices* file for the particular system. Check the *Systems* file for the device to be used to call the system.
WRONG TIME TO CALL	A call was placed to the system at a time other than what is specified in the *Systems* file.
TALKING	Self explanatory.
LOGIN FAILED	The login for the given machine failed. It could be a wrong login/password, wrong number, a very slow machine, or failure in getting through the Dialer-Token-Pairs script.
CONVERSATION FAILED	The conversation failed after successful startup. This usually means that one side went down, the program aborted, or the line (link) was dropped.
DIAL FAILED	The remote machine never answered. It could be a bad dialer or the wrong phone number.
BAD LOGIN/MACHINE COMBINATION	The machine called us with a login/machine name that does not agree with the Permissions file. This could be an attempt to maraude!
DEVICE LOCKED	The calling device to be used is currently locked and in use by another process.

Error Message	Description/Action
ASSERT ERROR	An ASSERT error occurred. Check the */usr/spool/uucp/Admin/errors* file for the error message and refer to the section ASSERT Error Messages.
SYSTEM NOT IN Systems	The system is not in the *Systems* file.
CAN'T ACCESS DEVICE	The device tried does not exist or the modes are wrong. Check the appropriate entries in the *Systems* and *Devices* files.
DEVICE FAILED	The open of the device failed.
WRONG MACHINE NAME	The called machine is reporting a different name than expected.
CALLBACK REQUIRED	The called machine requires that it calls your 3B2 Computer.
REMOTE HAS A LCK FILE FOR ME	The remote site has a LCK file for your 3B2 Computer. They could be trying to call your machine. If they have an older version of Basic Networking, the process that was talking to your machine may have failed leaving the LCK file. If they have the new version of Basic Networking, and they are not communicating with your 3B2 Computer, then the process that has a LCK file is hung.
REMOTE DOES NOT KNOW ME	The remote machine does not have the node name of your 3B2 Computer in its *Systems* file.
REMOTE REJECT AFTER LOGIN	The login used by your 3B2 Computer to login does not agree with what the remote machine was expecting.
REMOTE REJECT, UNKNOWN MESSAGE	The remote machine rejected the communication with your 3B2 Computer for an unknown reason. The remote machine may not be running a standard version of Basic Networking.
STARTUP FAILED	Login succeeded, but initial handshake failed.
CALLER SCRIPT FAILED	This is usually the same as "DIAL FAILED." However, if it occurs often, suspect the caller script in the *dialers* file. Use *uutry* to check.

12. The fx Disk Utility

fx is an interactive, menu-driven disk utility that lets you detect and map bad blocks on your disk. It also lets you display information stored on the label of the disk, including partition sizes, disk drive parameters and the volume directory.

An *extended* mode, which can be selected when *fx* starts running, provides additional functions normally used during factory set-up or servicing of disks, such as formatting the disk and creating or modifying the disk label.

Warning: Unless you are very familiar with the parameters and partitions of your disks, you are **strongly** advised not to invoke the extended mode of *fx*. A mistake in extended mode can destroy all the data on the disk.

12.1 Using fx

fx exists in two forms: as a regular IRIX system command, and as a standalone utility. The functions available in the two forms are the same, but the method of invoking them is different.

12.1.1 Using fx as an IRIX Command

You must be logged in as superuser to use *fx*, since it accesses the device files for the disks. You can use *fx* interactively by typing *fx* and pressing **<enter>** at the system prompt. When you do this, *fx* prompts for a disk controller type, with a default of the root disk controller type. Recognized controller types are *dksc* for SCSI drives, *dkip* for ESDI drives, *xyl* for SMD drives, and *ipi* for IPI drives.

Command Format

The format for using *fx* is:

fx [–x] ["*drive_spec*" ["*drive_type*"]]

–x invokes the extended mode. Include the quotation marks if *drive_spec* or *drive_type* contain spaces or shell special characters.

You can invoke *fx* interactively, in which case *fx* prompts you for each successive piece of information that your selection requires. Alternatively, you can invoke *fx* with a selection of command line parameters that specify the details of operation. Both methods of operation are described in this chapter.

To invoke *fx* interactively, type *fx* followed by **<enter>**. *fx* then prompts for controller number and drive number. Controller number is normally 0 unless your system has more than one controller and you wish to work on disks attached to the second controller.

Drive number depends on controller type. ESDI, SMD, and IPI drives are numbered starting from 0, 0 to 1 for two drive controllers, and 0 to 15 for 16 drive controllers. Drive 0 on controller 0 is normally used as the root disk. SCSI drives are numbered from 1 to 6, with drive 1 on controller 0 normally used as the root disk.

fx next prompts for the drive type. The default is the drive type stored in the disk label.

Command Line Parameters

You can also use command line parameters to specify controller type, controller number, and drive number as well as drive type, bypassing the interactive questions just described. The format is:

```
fx "controllertype(controller_number, drive_number)" "drive_type"
```

For example:

```
fx "dkip(0,1)" "Hitachi 512-17"
```

The quotes are essential in the first argument since parentheses are shell special characters, and in the second because the drive name contains a space.

12.1.2 Using fx as a Standalone Utility

You use the standalone version of *fx* when IRIX is not running. To invoke *fx* from the PROM monitor prompt, boot the standalone shell, *sash*, as described elsewhere. Once the `sash:` prompt appears, type one of the following commands:

For ESDI drives:

```
boot -f dkip()/stand/fx
```

For SCSI drives:

```
boot -f dksc(0,1,0)/stand/fx
```

For SMD drives:

```
boot -f xyl()/stand/fx
```

For IPI drives:

```
boot -f ipi()/stand/fx
```

You can also boot *fx* from the network. At the PROM monitor prompt, type:

```
boot -f bootp()host:pathname
```

In the preceding example, *host* represents the name of a system that has a copy of *fx* in its file system, and *pathname* represents the pathname of the *fx* file on the remote host.

fx starts in non-extended (normal) mode, and prints a message asking you whether you wish to use extended mode. You should answer yes to this only if you need to make significant changes to a disk. You can map bad blocks in non-extended mode.

fx then prompts for a device name to identify the disk controller type. Recognized names are `dksc` for SCSI drives, `dkip` for ESDI drives, `xyl` for SMD drives, and *ipi* for IPI drives.

fx then prompts you to provide the controller number and the drive number. Normally, you use controller 0 unless your system has more than one controller and you wish to run *fx* on disks attached to the additional controller. ESDI, SMD, and IPI drives use numbers starting at 0: 0 to 3 for four drive controllers, 0 to 7 for eight drive controllers, and so on. SCSI drives use numbers 1 through 6.

fx then prompts for the drive type. By default, *fx* uses the drive type stored in the disk label. You should select this unless you know that the drive type stored in the disk label differs from the actual drive type.

12.2 The fx Main Menu

Once you have selected controller type, controller number, drive number, and drive type, *fx* enters its main menu. You can select menu items either by *name* or by the *number* shown on the menu. A menu item can be an action (for example, *exit*), or the name of a submenu (for example, *badblock*).

If you select a submenu name, *fx* displays that submenu and lets you select from the options that the submenu displays.

To return to a parent menu from a submenu, enter two dots (..).

To obtain a "help" display giving more information about the items on the current menu, enter a question mark (?) at the prompt.

fx catches interrupts: an interrupt stops any operation in progress but does not terminate *fx* itself.

To exit from *fx*, select *exit* at the main menu. To go back to controller and drive type selections, enter two period characters at the *fx* prompt.

12.3 Bad Block Management

Most disks have a number of defective spots where data cannot be stored. The disk controller is able to get around this by dynamically replacing a bad block with a good block from a pool reserved for this purpose. The system keeps a list on the disk of defects and their replacements. If new bad blocks develop during the life of the system, you must add these new bad blocks to the bad block list. Typically, the disk driver prints error messages on the console when it encounters a bad block. These error messages might give the location of the bad block, either as a single block number or as cylinder, head and sector in a form such as:

```
chs:  123/4/5
```

Sometimes, the error message only contains the block number. The form depends on the controller type. The disk is identified by its special file name: see *dkip*(7) or *dksc*(7).

Note: For ESDI SMD, and IPI drives, *fx* will ask you whether you want to attempt to save data when mapping out bad blocks. *fx* does *not* save data for SCSI disks. In all cases, it is strongly recommended that you make a backup of the disk before proceeding with any bad block operations.

To map out a bad block:

1. Invoke *fx* with the **–x** and **–r0** options (**–r0** means zero retries). Select the appropriate disk, and then go to the *badblock* menu.
2. For non-SCSI disks, select the *readinbb* item to read in the existing bad block list from the disk. (**Note**: this step is not necessary for SCSI

disks, and the *readinbb* item does not appear on the menu in this case).

3. Select the *showbb* item to display the existing bad block list. Typically, there is a small number of entries. If there are no entries, it is possible that the bad block list has been lost or corrupted. In this case, for ESDI or SMD drives, attempt to recover the manufacturer's original defect list by entering *readdefects*. However, this option does not work on SMD drives that have been previously formatted. Once an SMD drive is formatted, the manufacturer's original defect list is lost. (Note that no corresponding function is available for SCSI drives, and the *readdefects* menu item does not appear).

4. To enter new bad blocks, select the *addbb* item. Then enter the location of the bad block (*fx* accepts either a single block number or a cylinder/head/sector specification). You can enter more than one bad block at a time; when you have finished, terminate the entries by entering two dots (..).

5. For non-SCSI disks, you must then save the updated bad block list to to disk and map out the new bad blocks. If you did not use the *fx* **–r0** option, select the *forward* option on the *badblock* menu to do this.

6. For SCSI disks, bad blocks are mapped out as soon as they are entered via the *addbb* function, and nothing more need be done; the *forward* item does not appear on the menu.

You can also automatically scan the disk surface for bad blocks:

1. Select the *exercise* option on the main *fx* menu.

2. Select *sequential* on the *exercise* menu and use the defaults for exercise type and range: this causes a read-only scan of the entire disk surface.

Defects detected during the scan are automatically added to the bad block list. Note that all exercises in the normal (non-extended) mode of *fx* are read-only for safety; this means that some defects might escape detection.

In addition, the *exercise* function marks as bad only unrecoverable blocks: blocks that can be accessed but need one or more retries are not marked as the **–r0** option makes *fx* map out any error it detects. For this reason, it is advisable to keep notes of persistent soft error messages (retries) which the disk driver prints on the console during normal operations, and add these bad blocks manually. It is best to replace a block that is going bad before it becomes unreadable.

12.4 fx Display Functions

fx can display the information in the various parts of the disk label. To do this, select the *label* option at the main menu. Then select the *readin* function, and select the part(s) of the label you wish to display; this reads the information from the disk. Return to the *label* menu and select *show*; you can then select the various parts of the label for display.

12.5 fx Debug Functions

fx has a menu of disk debug functions. For safety reasons, these are somewhat restricted in the normal (non-extended) mode. However, a function that you might find useful is the ability to read directly and display the contents of any block on the disk.

In the *debug* menu, select *seek*. You can then enter the block number you wish to read, either as a single block number or as a cylinder/head/sector specification. Note that this is an absolute block number, starting at the beginning of the disk: if you wish to read the *n*th block of a given partition you must convert this to an absolute block number by adding the starting block of the partition. (You can obtain partition information from the label display function mentioned above.)

Once you have selected the block, use *readbuf* to read it. (*readbuf* can read up to 100 consecutive blocks from the selected starting block.) Use *dumpbuf* to display data read from the disk; *dumpbuf* displays the data in hex and (if printable) in character format.

12.6 Menu Descriptions

The top level *fx* menu contains the following choices (you can select menu options by name or by number):

1) *exit* — Exits from *fx*. If changes have been made to the copy *fx* keeps of the disk label and these changes have not been written to the disk, a prompt appears, giving the option to write the changes to disk.

2) *badblock* Selects the menu of operations dealing with bad block handling.

3) *debug* Selects the menu of debug functions.

4) *exercise* Selects the menu of functions for analyzing the disk surface to find bad blocks.

5) *label* Selects the menu of functions for reading (and, in extended mode, modifying) the disk label.

The remaining options appear only in extended mode.

Caution: Extended mode makes it possible to write directly to the disk. Improper use of the extended mode can result in loss of data or damage to your file system structure.

6) *auto* Initializes a new disk. This formats the disk, creates a label and writes it to the disk, and exercises the disk to detect and map out bad blocks. Performing each of the above mentioned tasks manually takes less time. *auto* is not the best option.

7) *format* — Formats the disk. With SCSI disks, the whole disk is formatted. With ESDI, SMD, or IPI disks, it is possible to format a range of cylinders; *fx* displays prompts for the starting cylinder and the number of cylinders, the defaults being the entire disk. Note that with SMD disks, the defect information placed on the disk by the manufacturer is destroyed by formatting, so when formatting an SMD disk, *fx* automatically attempts to read and save this information first.

If the disk has been previously formatted, *fx* finds no defect information on the disk. In this case, it prints a message saying that no defect information was found after trying a few tracks, and offers the possibility of abandoning the search for it or continuing. If you know that an SMD disk has been previously formatted, answering "no" to this question shortens the time required to reformat the disk.

8) *restore* — Permits you to copy information from tape onto a disk partition. (*fx* does not provide disk-to-disk copying in the standalone version.)

12.7 badblock Menu

Bad blocks are handled differently by SCSI and ESDI/SMD/IPI controllers. In the case of SCSI controllers, the SCSI controller/formatter hardware maintains the list of bad blocks. Although you can read and write to the list of bad blocks, this list does not not appear in the user-readable part of a SCSI disk.

For ESDI/SMD/IPI disks, *fx* explicitly maintains the bad block list as a structure in the part of the disk label specific to Silicon Graphics equipment. ESDI/SMD bad blocks are managed on a track basis: if a track has one or more bad blocks, the entire track is replaced with one from the track replacement area reserved on the disk.

On IPI disks, defects are handled slightly differently. If there is only one bad block, that block is slipped to the next one, and each succeeding block is also slipped one. This practice is known as sector slipping. If there are two or more bad blocks on a track, the entire track is slipped as with SMD drives.

The *badblock* menu contains the following choices:

1) addbb — Adds new bad blocks to the bad block list. You can identify blocks either by a single block number or as cylinder/head/sector. When you have finished adding bad blocks, enter two dots (..); this returns to the *badblock* menu. For SCSI disks, *fx* immediately inserts a bad block in the on-disk list maintained by the SCSI controller/formatter. For ESDI/SMD/IPI disks, the list goes into the in-core copy of the bad block list maintained by *fx*, and does not become effective until you give the the *forward* command.

2) deletebb — Deletes a block from the bad block list. Essentially this is just to allow any mistake made during entry to be corrected; once a block has been identified as bad, it will remain so. Note that this function does not appear for SCSI disks; there is no way to remove an added bad block from a SCSI disk without reformatting the disk.

3) readdefects — Reads the defect information placed on the disk by the disk manufacturer, and adds bad blocks to the bad block list. This function appears in the menu only for ESDI, SMD, and IPI disk types. (If the blocks identified by the manufacturer's defect information are already in the bad block list, nothing is altered). This function is useful for retrieving original defect information if the bad block list in the disk label has become lost or corrupted for some reason. Note that formatting an SMD disk overwrites the manufacturer's defect information.

4) readinbb — Reads the bad block list in the disk label into memory. (Note that *fx* does not keep an in-core bad block list for SCSI drives, so this function appears only for ESDI, SMD, or IPI drive types.) Use this option before adding any bad block to ensure that you are working with the current bad block list.

5) showbb — Displays the current bad block list. For SCSI disks, *fx* reads the list maintained by the SCSI controller/formatter. For ESDI, SMD, and IPI disks, *fx* displays its in-core copy, originally read in with *readinbb* and possibly modified with *addbb*.

6) forward — For ESDI, SMD, and IPI disks, saves the bad block list to disk, and causes the disk controller to map out any newly added bad blocks. (It does not appear for SCSI disks since bad blocks added to a SCSI disk are effective immediately).

The remaining option appears only in extended mode.

7) createbb — Clears out any existing bad block list that *fx* holds for the disk. It appears only for ESDI, SMD, or IPI disks, and is normally used only when a disk is being completely reformatted.

12.8 Debug Menu

This menu gives access to miscellaneous debug functions, mostly for reading and writing disk blocks. *fx* provides an internal memory buffer as a source or destination for data; you can edit and display the contents of this buffer. In the normal (non-extended) mode of *fx*, only nondestructive functions are available. In the extended mode, you can write disk blocks as well as read them. Options are:

1) cmpbuf — Compares blocks of data in different areas of the buffer, for example, to compare written and read-back data. It prompts for the starts of the two areas to be compared (relative to the beginning of the internal buffer), and for the length of comparison.

2) dumpbuf — Displays the contents of the buffer. It prompts for start address (relative to the beginning of the buffer), length to display, and display format: bytes, (2-byte) words, or (4-byte) longwords. Data is displayed in the hex format selected, and also in character format with non-printable characters represented by dots.

3) editbuf — Modifies individual buffer locations in byte, 2-byte, or 4-byte units.

4) fillbuf — Fills sections of the buffer with a pattern. It prompts for start location and length to fill, and for a string of data to use as the fill pattern. (Only a string is accepted; you cannot enter hex data.) To clear the buffer, enter a null string.

5) number — Accepts a decimal number, and prints it in octal and hex. For ESDI, SMD, and IPI drive types only, it also prints the input, interpreted as a block number, in the form cylinder/head/sector for the current drive. It also accepts input in the form cylinder/head/sector (for example, 123/4/5) and prints the corresponding block number. (SCSI drives are always accessed by a single block number, so this translation is not performed if the current drive type is SCSI.)

6) readbuf — Reads disk blocks into the internal buffer. It prompts for buffer address (relative to start of buffer), and number of blocks to read. Up to 100 blocks can be read in one operation. *fx* stores the disk block address in an internal variable that you can set with the *seek* function.

7) seek — Sets the internal *fx* variable that holds the source or destination block number on disk for transfers between disk and the internal buffer. *seek* prompts you by displaying the current value of this internal variable.

The remaining functions appear only in extended mode, since they are either potentially destructive (for example, *writebuf*) or of little interest to the typical user.

8) writebuf — Writes blocks from the internal buffer to the disk. It prompts for source buffer address and number of blocks to write. The disk address block for the write is taken from the internal *fx* variable set by *seek*, as for *readbuf*.

9) showuib — Prints the unit initialization parameters used by the disk controller for the current drive. This function appears only for ESDI, SMD, and IPI drives.

10) showstatus — Prints the firmware ID code of the disk controller, and the status of the controller and drive. This function appears only for ESDI, SMD, or IPI drives.

11) rawreadbuf — Invokes the disk controller command intended for reading ESDI, SMD, or IPI manufacturer's flaw maps at the start of tracks. It prompts for destination buffer address, disk address, and length. It should be noted that for ESDI and SMD disks, the sector part of disk address is irrelevant, since the controller always reads at start of track for this function. Also, the length parameter is ignored. The controller always transfers one sector for this operation. This function appears only for ESDI or SMD drives.

12) rawreadcdc — Compensates for the different format of flaw maps on CDC disks by using a function of the Interphase controller. Appears only for Interphase controllers.

13) passthru — Permits ESDI command codes to be sent directly to an ESDI disk drive. This is a very low-level debugging interface for engineers familiar with the ESDI specification.

12.9 Exercise Menu

The exercise menu accesses functions intended for surface analysis of the disk to find bad blocks. Only read-only tests are possible in normal (non-extended) mode; destructive read-write tests are allowed in extended mode.

1) butterfly — Invokes a test pattern in which successive transfers cause seeks to widely separated areas of the disk. This is intended to stress the head positioning system of the drive, and will sometimes find errors that do not show up in a sequential test. It prompts for the range of disk blocks to exercise, number of scans to do, and a test modifier (described below). Each of the available test patterns can be executed in a number of different modes (read-only, read-write, etc.), which are described below.

2) errlog — Prints the total number of read and write errors detected during a previous exercise.

3) random — Invokes a test pattern in which the disk location of successive transfers is selected at random; intended to simulate a multiuser load. As with the *butterfly* test, *random* prompts for a range of blocks to exercise, the number of scans, and a modifier (described below).

4) sequential — Invokes a test pattern in which the disk surface is scanned sequentially. As with the *butterfly* test, it prompts for a range of blocks to exercise, the number of scans, and a modifier (described below).

The following items appear only in extended mode, since they are concerned with destructive (write) tests.

5) settestpat — Creates the pattern of data to be used in tests that write to the disk. You can initialize one sector of data byte by byte, entering each byte as a decimal or a hex value.

6) showtestpat — Displays the pattern of data to be used in tests that write to the disk. The default is a repeating pattern of 0xdb 0x6d 0xb6. Use *settestpat* to change this pattern.

7) complete — Runs a write-and-compare sequential test on the entire disk area.

The *butterfly*, *random*, and *sequential* tests prompt for a modifier. The modifier determines the type of transfer that will occur during the test patterns. Possible modifiers in non–extended mode are:

rd-only — Reads one track of the disk at each location in the test pattern. The value of read data is ignored; the test detects only the success or failure of the read operation.

rd-cmp — Reads the disk track twice at each location in the test pattern. The data obtained in the two reads is compared.

seek — Reads one sector at each step of the test pattern to confirm that the issued seek executed correctly, without checking readability of a complete track.

The following modifiers are available only in extended mode because they destroy existing data by specifying disk write operations.

wr-only — Writes data to one track of the disk at each location in the test pattern. Written data is not re-examined; the test detects only the success or failure of the write operation.

wr-cmp — Writes data to one track of the disk at each location in the test pattern. The written data is then read back and compared with its expected value.

12.10 Label Menu

This menu gives access to functions for displaying and (in extended mode) modifying information contained in the disk label. It contains the following items:

1) readin — Allows part or all of the label to be read in from the disk. Selecting this item brings up a menu of the accessible parts of the label. (These are described in detail below.) Selecting a part causes that part to be read in from disk; there is also an *all* option, to read in all parts at once.

2) show — Displays parts of the label. As with *readin*, *show* brings up a menu of the label parts, allowing selection of the part to be displayed.

The remaining items appear only in extended mode, since they offer the possibility of changing data on the disk.

3) sync — Writes the in-core copy of the disk label back to disk.

4) set — Modifies parts of the label. As with *readin*, *set* brings up a menu of the label parts and lets you choose the part to be modified. See "Parts of the Disk Label," below, for more information.

5) create — Discards existing label information, and creates new label information. If the label on disk is valid, the created label information is based on this, otherwise default label information based on the drive type is created. This would normally be used only for attempting to repair a damaged disk label (or to recover from major errors during *set*). As for *readin*, it brings up a menu of the label parts, allowing selection of the part to be worked on.

12.11 Parts of the Disk Label

A disk label contains the following parts:

1) parameters

This is information used by the disk controller, such as disk geometry (for example, number of cylinders), and format information (for example, interleave). The *show* submenu of the *label* menu displays this information. In extended mode, you can change the parameters with the *set* submenu, or reset them to default values for the drive type with the *create* submenu.

The parameters depend on the type of controller. These values do not need to be changed in normal use; refer to the manufacturer's documentation for the disk controller and disk drive.

2) partitions

The disk surface is divided for convenience into a number of different sections ("partitions") used for various purposes. (See *intro* (7m) for more details.) When the operating system is accessing the disk, its drivers make the connection between the special file name and the physical disk partition using information from the partition table in the disk label.

You can find this information under the *show* submenu of the *label* menu. In extended mode, you can change the partitions through the *set* submenu, or reset them to default values for the drive type through the *create* submenu. There can be up to 16 partitions on a disk, numbered 0 to 15 (though not all need be present). Each partition is described by its starting block on the disk, its size in blocks, and a type indicating its expected use (for example, file system, disk label, swap, etc).

3) sgiinfo

Contains information kept for administrative purposes: the type of disk drive and its serial number. As for other parts of the disk label, use the *show* submenu of the *label* menu to view *sgiinfo*. In extended mode, use the *set* submenu to change *sgiinfo*, or the *create* submenu to reset it to default values for the drive type.

4) bootinfo — Contains information used by the system PROMs during a normal system boot. It specifies the root partition, the name of the file on the root partition to boot, and the swap partition. Normal defaults for these are root partition 0, filename */unix* and swap partition 1. To view the current setting, you can use the *show* submenu of the *label* menu; in extended mode, you can change the setting by using the *set* submenu. The defaults appear as prompts, and can be changed by inputting different values.

5) directory — Some system files are normally kept in the label area on the disk. These files are used in standalone operations such as the standalone shell *sash* and the standalone version of *fx*. The directory is a table in the label; this table enables these files to be located. The *show* submenu of the *label* menu allows the directory of these files to be displayed. The files in the disk label are manipulated by *dvhtool*(1M) and *fx* does not provide facilities for adding or deleting them.

12.12 Initializing New Disks

You can use *fx* to initialize disk drives that have not been previously formatted.

The new drive to be initialized should be physically connected to the system. Do not change disk connections while the system's power is on.

Take care that termination of the new drive is correct and that its drive id does not conflict with that of any other drive connected to the same controller. With the new drive connected, restore system power and invoke *fx* in *extended* mode as described earlier in this chapter.

When *fx* prompts you, enter the controller type and number, and the drive number for the new drive. For SCSI drives, the drive type is determined automatically by an inquiry operation on the drive. For ESDI, SMD, or IPI drives, a menu of known drive types appears; it is important to know the exact model number of the drive you are adding.

Once drive type is identified, select the *auto* item on the main menu; this formats the drive, scans it for bad blocks, and places a label on it. On completion of this, exit from *fx*; the drive is now ready for use.

Note that for the new drive to be useful in the system, you must create partitions, optionally add the partitions to logical volumes, make the filesystems, then mount them on the disk. See *mkfs* (1M), *growfs*(1M), *lv*(1M) and *mount* (1M).

For more information, see *dvhtool*(1M), *dkip*(7), *dksc*(7), *vh*(7), and the following example of *fx*.

Appendix A: Device Files

This appendix contains a list of device files and directories that reside in the */dev* directory.

dsk/	Directory containing block device files for disks; see ips(7), dks(7) and xyl(7) for disk partition device names.
rdsk/	Directory containing raw device files for disks; see ips(7), dks(7) and xyl(7) for disk partition device names.
root	Generic root partition (block device).
rroot	Generic root partition (raw device).
usr	Generic usr partition (block device).
rusr	Generic usr partition (raw device).
swap	Generic swap partition (block device).
rswap	Generic swap partition (raw device).
vh	Generic root volume header (block device).
rvh	Generic root volume header (raw device).
mt/	directory containing block device files for tapes; see ts(7) for ISI quarter-inch tape drive device names; see tps(7) for SCSI quarter-inch tape drive device names; see xmt(7) for Xylogics half-inch tape drive names.
rmt/	directory containing raw device files for tapes; see ts(7) for ISI quarter-inch tape drive device names; see tps(7) for SCSI quarter-inch tape drive device names; see xmt(7) for Xylogics half-inch tape drive names.

tape	Generic tape device; bytes are swapped in order to be compatible with the IRIS series 2000 and 3000 workstations; see mtio(7).
nrtape	Generic no-rewind tape device; bytes are swapped in order to be compatible with the IRIS series 2000 and 3000 workstations; see mtio(7).
tapens	Generic tape device; bytes are not swapped; see mtio(7).
nrtapens	Generic no-rewind tape device; bytes are not swapped; see mtio(7).
mem	Memory; see mem(7).
mmem	Mappable memory; see mmem(7).
kmem	Kernel memory; see kmem(7).
null	Null device (zero length on input, data sink on output); see null(7).
SA/	Block devices used by System Administration tools; see sysadm(1M) and sa(7).
rSA/	Raw devices used by System Administration tools; see sysadm(1M) and sa(7).
audio	Audio interface (series 20 only).
dn_ll	File used to create 4DDN logical links; see dn_ll(7).
dn_netman	File used by 4DDN network management software; see dn_netman(7).
cent	Centronics color graphics printer device.
tek	Tektronics color graphics printer device.
vers	Versatek color graphics printer device.
vp0	Hard link to vers.
gpib*	GPIB (IEEE-488) device; see gpib(7).
gse	Spectragraphics coax device; see gse(7).
plp	Parallel line printer interface; see plp(7).

prf	File used by operating system profiler; see prf(7).
t3270	Raw device file for IBM 3270 Cluster Controller; see t3270(7).
hl/	Directory containing files used by GTX series machines hardware spinlock driver; see usnewlock(3P).
log	Named pipe that is read by the system logging daemon; see syslogd(1M).
ptc	Clonable pseudo-tty controller; see clone(7), ptc(7).
grconc	Master pseudo-teletype for the graphics console; see pty(7).
grcons	Slave pseudo-teletype for the graphics console; see pty(7).
gm	Logical console device for the Graphics Manager on the GT and GTX model machines. Messages from the software running on the 68020 on the GM board will appear as output on this device.
grin/	Directory containing the individual logical graphics input devices.
gro/	Directory containing the individual logical graphics output devices for 4D 50, 60, 70, and 80 series, model G machines.
console	System console device.
syscon	Hard link to /dev/console.
systty	Hard link to /dev/console.
queue	Graphics queue device. Graphics programs call 'select' on this device in order to be notified when there is input in their graphics queue. This device can't be actually read or written.
dials	Device for serial port connected to dial and button box.
keybd	Device for serial port connected to keyboard.
mouse	Device for serial port connected to mouse.
tablet	Device for serial port connected to digitizing tablet.
ttyd[1-12]	Serial ports 1-12.
ttyf[1-12]	Serial ports 1-12 for devices that understand hardware flow control.

ttym[1-12]	Serial ports 1-12 for modems.
ttyq*	Pseudo tty devices; see pty(7).
zero	Zero device (infinite zeros on reads); see zero(7).

Appendix B: Error Messages

This appendix lists error messages you may receive from the IRIX operating system. Where a message is unique to either IRIX version 3.2 or 3.3, you will see a reference to that version, for example:

`Swap failed (page still in memory)` (IRIX version 3.3)

The operating system handles error conditions according to their degree of severity, and divides the error messages into three classes: NOTICE, WARNING, and PANIC.

- NOTICE messages provide information on the system status. These messages can sometimes help you to anticipate problems before trouble occurs.
- WARNING messages indicate that the system might stop functioning unless you take corrective action.
- PANIC messages indicate that a problem is severe enough that the system comes to a halt. The cause is usually a hardware problem or a problem in the kernel software. This type of message occasionally occurs; it should not cause much concern. If a particular PANIC message occurs repeatedly or predictably, however, contact your service representative.

The error messages in this appendix are listed alphabetically in their respective classes. The messages that follow relate to tunable parameters. For other error messages, see */usr/include/sys/errno.h*, the *IRIX Programmer's Reference Manual* section titled "intro(2)," and the Owner's Guide for your system.

B.1 NOTICE Messages

```
Can't allocate message buffer (IRIX version 3.2)
```

The product of the tunable parameters MSGSSZ and MSGSEG was too high. When convenient, correct the msg master file, reconfigure, and reboot.

```
Configured value of NOFILES (#) is greater than min (#)
```

The tunable parameter NOFILES was found to be greater than the system-imposed limit. Modify this parameter in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

```
Configured value of NOFILES (#) is less than min (#)
```

The tunable parameter NOFILES was found to be less than the system-imposed limit. Modify this parameter in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

```
diskname: Swap out failed  blkno x (page still in memory)
```

A swap write failed.

```
File table overflow
```

The number of systemwide open files has exceeded a tunable limit. If this happens frequently, increase the NFILE parameter in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

```
Insufficient memory to lock/allocate # pages -
system call failed for process name
```

There is insufficient free memory to execute the system call; reduce the system load and try again, or add more memory.

```
I/O error in swap, [name] -- dev #, blkno # (IRIX version 3.3)
```

A hard error occurred during a transfer to or from swap space. Record the block number and map it out using the *fx*(1M) utility.

`"str" - swpuse count overflow`

More than 256 processes are sharing the same page of swap. A copy has been made. No action is required.

`Setting v.v_syssegsz down to 0x4000 (up to 0x800)`
(IRIX version 3.3)

The configured value SYSSEGSZ in */usr/sysgen/master.d/kernel* was either too small or too large. It has been changed.

`swapdel - too few free pages` (IRIX version 3.2)

There is not enough free main memory to delete the swap device at this time. Reduce the system load and try again.

`tune.t_maxfc reduced to #` (IRIX version 3.2)

The tunable parameter MAXFC was found to be greater than the system-imposed limit. MAXFC specifies the maximum number of pages that are added to the free list in a single operation. It has been automatically reduced to that number. When convenient, correct the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`tune.t_maxsc reduced to #` (IRIX version 3.2)

The tunable parameter MAXSC was found to be greater than the system-imposed limit. MAXSC specifies the maximum number of pages that are swapped out in a single operation. It has been automatically reduced to that limit. When convenient, correct the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`useracc - no memory to lock page`

There is insufficient free memory to lock a user data page in memory to service a read or write system call to a raw device. Reduce the system load, reduce the size of the raw I/O buffer in the user program, or add more memory.

B.2 WARNING Messages

`iget - inode table overflow`

Each active file in the system requires an entry in the inode table, and this table has overflowed. Reduce the system load. If this happens frequently, increase the tunable parameter NINODE in the /usr/sysgen/master.d/kernel file and reconfigure.

`mfree map overflow #`

Fragmentation of some resource (such as message queues) led to some of the resource being lost. No action necessary.

`out of dynamic memory, nbytes=#` (IRIX version 3.2)

No more pages are free to be allocated for kernel data structures. Reduce the system load. If this happens frequently, increase KHEAP_MAXBYTES in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`out of memory making pipe` (IRIX version 3.2)

Can't allocate an inode for an incore pipe. Reduce the system load.

`out of memory cloning` (IRIX version 3.2)

Can't allocate an inode for a clone device. Reduce the system load.

`out of physical memory, nbytes=#`

No more physical pages are available. Reduce the system load or add more memory.

`Paging Daemon (vhand) not running - NFS server down?`

The system has determined that *vhand* is not executing, possibly because it is waiting for an I/O transfer to complete to an NFS server (especially if the NFS file system is hand mounted).

`Process [name] pid # killed due to insufficient memory/swap` (IRIX version 3.3)

The named process would not fit into available memory. This may be caused by a run-away process or too little available swap for a process.

`Process name (#) killed due to bad page read (error #)` (IRIX version 3.3)

When attempting to fault in a page, the read failed. This may be either a bad disk block or an NFS server failure.

`Region table overflow` (IRIX version 3.2)

Each text, data, stack, shared memory, and mapped file segment requires one entry in the region table. The system call that tried to allocate another region failed. Reduce the number of processes (NPROC), or in- crease the number of region table entries (NREGION) in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`Stray VME interrupt on level # at PC=# VME vector=#` (IRIX version 3.3)

A VME device has interrupted but no configured driver exists. This is either a hardware problem with one of the devices or an incorrectly con- figured kernel.

`Swap space running out` (IRIX version 3.2)

Can't get a contiguous chunk of swap space of the required size, so do swaps one page at a time. Reduce the system load by reducing the num- ber of large processes, or add another swap partition.

`unswap - I/O error in swap (continuing)` (IRIX version 3.3)

When attempting to delete a swap area, a stored page could not be read. The swap space will not be deleted until the read is successful.

B.3 PANIC Messages

`bumpcnt - region count list overflow`

Indicates a problem with the operating system.

`getpages - pbremove`

Indicates a problem with the operating system.

`init died` (IRIX version 3.3)

Process 1 died. This could be due to a memory error; otherwise it is a problem with the operating system.

`invalid upg/kstk` (IRIX version 3.3)

A problem with the operating system. Contact customer support.

`IRIX killed due to Memory Error at Physical Address #`
(IRIX version 3.3)

A physical memory error has occurred (either parity or double bit ECC). Contact customer support.

`IRIX killed due to VME Bus timeout` (IRIX version 3.3)

A VME device did not respond. Contact customer support.

```
KERNEL FAULT: (IRIX version 3.3)
PC:  #    ep:  #
EXC code:  #
Bad_Addr:  #
```

The system detected a bad object access. The object accessed is at "Bad_Addr" and the program counter is at "PC." This is either a problem with a device driver or the operating system. Write down all information. If there is a system core dump, use *dbx* **-k** to determine what part of the operating system caused the fault.

`kern_free: null pointer` (IRIX version 3.3)

The caller of *kern_free* attempted to free a null pointer.

`kern_malloc: invalid size -- #` (IRIX version 3.3)

The caller of *kern_malloc* attempted to allocate 0 bytes.

`kern_realloc:`
`kern_free: free ptr ~ # block ~ # already free`
(IRIX version 3.3)

The caller of *kern_free* attempted to free an already freed block.

`kseg - ptmemall failed` (IRIX version 3.2)

No physical memory was available for the kernel or driver when needed.

`kvpalloc: size # illegal with VM.DIRECT` (IRIX version 3.3)

The caller of *kupalloc* attempted to allocate more than one page.

`no root found` (IRIX version 3.3)

The configured *root* (*/*) file system could not be mounted. Check the root prom environment variable, the disk drive cabling, or make sure that the specified drive partition has a valid file system on it.

`not enough vm for buffer cache` (IRIX version 3.2)

The tunable parameter for the maximum size of the buffer cache, NBUF, was specified too high. Decrease this value, reconfigure, and reboot.

`not enough vm for malloc` (IRIX version 3.2)

The tunable parameter for dynamic kernel memory, KHEAP_MAXBYTES, was specified too high. Decrease this value in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`nsplock (#) too large`

The number of kernel hardware locks configured exceeded the system-imposed limit. Lower NSPLOCKS in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`out of spinlocks`

The multiprocessor kernel ran out of hardware locks for synchronization. If this happens frequently, increase the tunable parameter NSPLOCKS in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`piinsert dup # and #` (IRIX version 3.3)

A problem with the operating system. Contact customer support.

`pinsert dup # and #` (IRIX version 3.3)

A problem with the operating system. Contact customer support.

`Ran out of action blocks`

A resource used by the multiprocessor kernel for inter-CPU interrupts has run out. If this happens frequently, increase the parameter NACTIONS in the kernel file, reconfigure, and reboot.

`Read error in swap` (IRIX version 3.3)
`swapseg - i/o error in swap` (IRIX version 3.2)

A hard error occurred during a transfer to or from swap space. Record the block number (bn) and map it out using the *fx*(1M) utility.

`Timeout table overflow`

Increase NCALL in the */usr/sysgen/master.d/kernel* file, reconfigure, and reboot.

`vfault P dbd_type DBD_MFILE` (IRIX version 3.3)
`vfault - bad dbd_type`

The page being faulted in is not a recognized type: either demand fill, demand zero, in file or on swap (operating system error).

Index

I

K

L

M

N

O

P

R

S

T

U

V

W

Y