MIGHTYFRAME™ SERIES
ADMINISTRATOR'S REFERENCE MANUAL

Specifications Subject to Change.

First Edition (May 1986) 09-00620-01

LIST OF FIGURES

LIST OF TABLES

The following conventions are used throughout this manual:

- "MightyFrame" is used generically to describe both the Mighty-C and MightyFrame computers, except where differences are noted in the text.

- Square brackets [ ] indicate an optional command argument.

- Braces { } indicate that there is a choice of required options.

- Ellipsis (pn ... pn) indicates that the previous parameter can be repeated any number of times.

- Boldface (**cat**) indicates that the item must be typed exactly as shown in the command line. Names of CTIX programs and CTIX file names are also shown in boldface.

  A name in boldface followed by a number enclosed by parentheses, such as **fsck**(1M), indicates that the item is described in your CTIX operating system manual; the number in parentheses is the section containing the item.

- The **Break** key is the key that sends a Break signal to the MightyFrame host. On a Programmable Terminal (PT) or Graphics Terminal (GT) the **Cancel** key sends a Break signal when the terminal is in emulate mode. On other terminals, the Break signal is normally sent by a key labeled "**Break**" or "**Attention.**"

- Underlining (input) indicates that the user must supply the appropriate information (such as file name or variable name).

- Complete examples are placed at the end of a chapter.

- In all examples, administrative input is in boldface; the system's responses are in normal type; ######## indicates unechoed input; and ^D indicates Control-D (**Code-D** or **Finish** on a PT or a GT).

1 OVERVIEW

This manual describes how to administer MightyFrame and Mighty-C computer systems: it explains your responsibilities as system administrator and gives procedures for you to follow. Used in conjunction with your CTIX operating system manual, this manual is intended to be a complete reference to MightyFrame CTIX administrative principles and practice. As such, it provides instruction on MightyFrame system administration without the aid of the CTIX administrative tools.

If you are using the CTIX administrative tools, you will need to use this manual to perform those functions you cannot perform with the tools alone. In this case, if you are new to CTIX system administration, you will probably also want to use this manual to gradually increase your knowledge of CTIX administration. This manual notes the functions that can be performed with the administrative tools.

If you are not using the administrative tools, you will need to become familiar with all of the CTIX programs and files described in this manual. If you are not already familiar with the principles of UNIX system administration, spend some time reading and studying this manual. Then get some "hands-on" experience with the system before you begin to support real MightyFrame users. A roadmap to the setup process is provided in "Using This Manual," later in this chapter.

NOTE

The version of the CTIX operating system manual is specific to a release or series of releases of CTIX software. For example, the operating system manual that corresponds to CTIX 5.1Ø is the CTIX Operating System Manual, Version B, with Update Notice 1.

Always consult the Release Notice for the version of the operating system manual that corresponds to your version of CTIX operating system software.

RESPONSIBILITIES OF MIGHTYFRAME ADMINISTRATION

Whether you spend all or only a small part of your time performing the role of MightyFrame system administrator, these are your specific responsibilities:

- allocating and denying other users access to the system
- specifying what disks and disk files users can access
- preparing new disks for use by the system
- changing CTIX to use new terminals and printers
- performing routine backup of disk files to prevent accidental loss of data
- making the system reasonably secure from tampering
- installing new releases of software
- maintaining the operating system software and troubleshooting problems when they arise

- optimizing system performance
- starting the system and turning the system off

To carry out these responsibilities, you access the system in ways that other users cannot, as the "superuser." Having superuser status allows you to control the way other users use the system and to make the system run as smoothly as possible for the benefit of all users.

CAUTION

Having superuser status makes it possible to accidentally destroy user and system files, and in other ways interfere with the proper functioning of the system! This is why your most important responsibility is to understand the principles and procedures of MightyFrame CTIX system administration.

USING THIS MANUAL

The MightyFrame Series Administrator's Reference Manual is organized as follows:

- This chapter, "Overview," describes your responsibilities as system administrator and explains how to use this manual.
- Chapter 2, "CTIX Modes and Superuser Status" describes multiuser, single-user and administrator modes. It also explains what "superuser" status means, how to become the superuser, and how to take CTIX to single-user mode and back to multiuser mode.

- Chapter 3, "CTIX System Directories," describes system directories and the kinds of files contained in each one. It also gives guidelines as to where to put new binary files, libraries, and shell scripts that are not part of the CTIX software distribution.

- Chapter 4, "MightyFrame Physical I/O Channels," describes how to associate entries in the **/dev** directory with physical channels on the MightyFrame and Mighty-C computer systems.

- Chapter 5, "Terminals and Modems," explains how to make CTIX support terminals and modems.

- Chapter 6, "Printers," provides information on how to make CTIX support printers.

- Chapter 7, "CTIX Device Drivers," explains how to configure CTIX for loadable and nonloadable device drivers.

- Chapter 8, "Disks," describes how disks are organized, how to initialize a disk and divide it into CTIX partitions, how to create a file system, and how to run **fsck**, the CTIX file system checker, and interpret its results.

- Chapter 9, "Adding and Supporting Users," explains how to allocate and deny users access to the system, how to make CTIX reasonably secure, and how to use CTIX programs to communicate with users.

- Chapter 10, "UUCP," describes how to set up and administer UUCP, the UNIX-to-UNIX copy program.

- Chapter 11, "CTIX System Accounting," describes how to set up, manage, and interpret reports generated by the CTIX accounting system.

- Chapter 12, "Backups and Restores," explains how to efficiently preserve offline copies of CTIX files against accidental loss.

- Chapter 13, "System Maintenance and Troubleshooting," discusses how to keep the system running smoothly and how to detect the sources of difficulties that may arise. This chapter also explains how to use the System Activity Package to monitor CTIX operation and how to use the maintenance tape to restore the system.

- Appendix A, "File System Concepts," explains the data structures and concepts behind a CTIX file system.

- Appendix B, "System Initialization Programs and Administrative Files," describes the start-up programs **init**, **getty**, and **login**, and provides a reference to administrative files.

- Appendix C, "CTIX AutoConfiguration," describes the formulas used to automatically configure CTIX and the procedures to override autoconfiguration.

GETTING STARTED WITH THIS MANUAL

How you use this manual depends on your experience with UNIX and UNIX system administration, and on the specific configuration of your MightyFrame system.

The MightyFrame Series Administrator's Reference Manual assumes that you already know the following:

- basic information about the logical structure of the UNIX file system
- how to use the most common tools for manipulating files: **cat**, **grep**, **mv**, **cp**, and so forth
- how to use a UNIX editor
- how to use your CTIX operating system manual as a reference

If You Are New to UNIX

If you are new to UNIX, you will need some preparation before you are ready to set up and maintain your MightyFrame system. You will probably also want to rely heavily on the examples at the end of the setup chapters.

The following road map describes the process of (1) learning enough UNIX to get started and (2) setting up your MightyFrame system:

1. Have on hand one of the commercially-available introductory books on the UNIX operating system. Read the introductory material in your CTIX operating system manual, and learn how the information is organized.

2. Read Chapter 2 of this manual to learn how to turn the MightyFrame computer on and off and how to become the super-user.

3. Read the installation manual for your MightyFrame system: the MightyFrame Installation Manual or the Mighty-C Installation Manual. If the MightyFrame system is already installed, become familiar with the physical setup of the system. If it is your responsibility to install it, perform the installation.

4. Connect a 9600-baud terminal to Mighty-Frame Channel 0 on the main CPU board and power on the system.

5. Load operating system software according to the instructions in the Release Notice. Set the date according to the information in Chapter 13, "System Maintenance and Trouble-shooting."

6. Learn to use a UNIX editor by reading the appropriate pages of the introductory book you selected in Step 1 above.

7. Configure the terminal connected to Channel 0 (from Step 4 above) according to the information in Chapter 5, "Terminals and Modems." (Channel 0 is already partially configured; you need only make an entry in **/etc/ttytype**.)

   Read "Adding Users," in Chapter 9. Create a password file entry for yourself, give yourself and **root** a password, and give yourself a home directory.

Create text files and practice using a UNIX editor. Practice using basic UNIX tools to manipulate these files.

Although not essential, it is very helpful to know something about the shell programming language. It is recommended that you learn how to read simple shell scripts. Most commercially-available books on the UNIX operating system contain information about the shell language.

8. Read all of Chapters 4 through 9 and Chapters 12 and 13. Initially, devote special attention to Chapters 4 through 9 and Chapter 12.

9. Configure CTIX for the terminals, modems, printers, and drivers you will add initially; do this according to the information in Chapters 5 through 7 and the reference material in Appendix B. Configure the disks using the information in Chapter 8, "Disks."

10. Familiarize yourself with all the concepts and configurable files in Appendix B.

    Configure the files that apply to all users, such as **/etc/profile** and the **crontab** files.

    If you will be using UUCP, read Chapter 10, "UUCP," and configure UUCP. If you will be using the accounting system as soon as you add users, read Chapter 11, "CTIX System Accounting," and turn on the accounting system.

11. If you need to install additional software (for example, a FORTRAN compiler or an office applications software package), do that now.

12. Begin adding users according to the information in Chapter 9, "Adding and Supporting Users," and the reference material in Appendix B. Begin performing backups as soon as users are working on the MightyFrame system; information on backups is provided in Chapter 12, "Backups and Restores."

    Become familiar with the information in Appendix A, in case you experience file system problems.

If You Have a Basic Knowledge of the UNIX Operating System

If you are familiar with the UNIX operating system but are new to UNIX system administration, follow the road map above, making the obvious adjustments.

If You Are an Experienced UNIX System Administrator

If you have performed the role of system administrator on other UNIX or CTIX equipment, the transition to MightyFrame CTIX should be relatively easy. Become familiar with the few differences between the organization of your CTIX operating system manual and the standard AT&T documentation.

Read through this manual looking for differences between MightyFrame and other UNIX systems you have used. Then follow the steps below before you start to configure your system:

1. Read Chapter 8, "Disks," very carefully: there are important differences between MightyFrame CTIX and other versions of CTIX and UNIX where disks are concerned.

2. Note any differences between the distributed configuration files described in Appendix B and those you are accustomed to. Read about the file **/etc/drvload** and about loading device drivers in Chapter 7, "CTIX Device Drivers."

3. Note the use of administrator mode (Chapter 2, Chapter 5, and Appendix B).

4. Pay particular attention to Chapter 4, "MightyFrame Physical I/O Channels." If you are using RS-422 terminals, note the peculiarities of their configuration as described in that chapter. Note also that **/dev/console** is not associated with a physical device and that console messages are saved in **/etc/log/confile**, the console log file.

5. Read the installation manual for your MightyFrame or Mighty-C computer system. Install the MightyFrame if that is your responsibility.

6. Connect a 9600-baud terminal to MightyFrame Channel 0 on the main CPU board and power on the system.

7. Load operating system software according to instructions in the Release Notice. Set the date.

Continue with the setup process in whatever fashion you think appropriate.

RELATED DOCUMENTATION

The documents described below provide additional information related to this manual.

Hardware
- MightyFrame Diagnostics Manual
- MightyFrame Hardware Manual
- MightyFrame Installation Manual
- MightyFrame VME Ethernet Card Manual
- Mighty-C Installation Manual

CTIX Operating System
- CTIX Operating System Manual, Version B

System Administration
- CTIX Administration Tools Manual

Program Development
- CTIX Programmer's Guide
- Graphics Terminal Programmer's Guide
- Programmable Terminal Programmer's Guide
- Writing MightyFrame Device Drivers

Communications and Networks
- CTIX Internetworking Manual
- CTIX SNA 3270 Terminal Emulator Manual
- CTIX SNA Network Gateway Manual

HARDWARE

The MightyFrame Diagnostics Manual explains how to execute MightyFrame diagnostic tests to ensure proper computer operation or to identify malfunctions.

The MightyFrame Hardware Manual contains the hardware description of the MightyFrame computer system.

The MightyFrame Installation Manual explains how to install the MightyFrame computer, its expansion cards, hard disk drives, and terminals.

The Mighty-C Installation Manual explains how to install the Mighty-C computer, its expansion cards, hard disk drives, and terminals.

The MightyFrame VME Ethernet Card Manual contains a hardware description of the Ethernet card and procedures for installing it.

CTIX OPERATING SYSTEM

The CTIX Operating System Manual, Version B, describes the CTIX Operating System, an operating system derived from the UNIX System V operating system. The manual describes CTIX commands, application programs, system calls, library subroutines, special files, file formats, games, miscellaneous facilities, and system maintenance procedures.

SYSTEM ADMINISTRATION

The CTIX Administration Tools Manual describes how to administer a MegaFrame, MiniFrame, or MightyFrame system using the CTIX administrative tools software package and without using CTIX shell commands.

PROGRAM DEVELOPMENT

The CTIX Programmer's Guide addresses the programmer's need for detailed explanations about the CTIX operating system. The guide discusses major tools, including basic interaction with the operating system, source code control, calculator languages, text editing, and C programming. Use the guide in conjunction with your CTIX operating system manual.

The Graphics Terminal Progammer's Guide and Programmable Terminal Programmer's Guide describe the programmable functions of the Graphics Terminal (GT) and the Programmable Terminal (PT).

Writing MightyFrame Device Drivers is a tutorial and reference manual for the experienced C programmer. It describes MightyFrame architecture, the CTIX kernel, and other information about developing device drivers, along with annotated sample source code for working CTIX drivers.

COMMUNICATIONS AND NETWORKS

The CTIX Internetworking Manual is both a tutorial and reference guide for users and programmers. The manual describes CTIX internetworking concepts, how to use CTIX network commands, and how to set up and manage a network. CTIX internetworking is a protocol-independent system that is compatible with many protocols and media (for example, TCP/IP, UUCP, and Ethernet).

The CTIX SNA 3270 Terminal Emulator Manual outlines the features and functions of the SNA 3270 Terminal Emulator. The manual provides detailed instructions for using, installing, configuring, and troubleshooting the Emulator software.

The CTIX SNA Network Gateway Manual provides technical information and operating procedures for the installation, configuration, operation, and maintenance of a CTIX SNA Network Gateway. The manual also defines IBM SNA general concepts and describes the Gateway components.

Users use the CTIX operating system in multiuser mode, which is the normal operating mode of CTIX.

In your role as MightyFrame administrator, you use the CTIX operating system in ways that other users cannot. This is necessary so that you can manage the system and its resources.

You have three special ways to interact with CTIX. Note that the first is not exclusive of the other two.

- Superuser status. When using CTIX as the superuser, you can ignore CTIX restrictions on file access and allowable commands.

- Single-user mode. When CTIX is in this mode, only one terminal is usable. Single-user mode is used for procedures that require an absence of normal disk activity. The single user in single-user mode has superuser status.

- Administrator mode. When CTIX detects a serious file system problem at system start-up, it automatically goes to administrator mode. Then, when you log in as **root** (superuser) on an eligible terminal, CTIX goes to single-user mode.

This chapter describes these interactions and the three CTIX modes: multiuser, single-user, and administrator.

CAUTION

Do not reset or turn off a MightyFrame computer system unless CTIX is not running or is running in single-user mode.

SUPERUSER STATUS

Superuser status removes important CTIX restrictions. The administrative CTIX commands in this manual require superuser status. CTIX gives the superuser three exemptions from normal restrictions:

- File read and write permissions do not apply to the superuser. The superuser can write to or read from any ordinary or special file. The superuser can create a file in or delete a file from any directory.

- Certain commands are executable only by the superuser.

- Certain commands have built-in safeguards or restrictions on the way they are used. Some of these safeguards and restrictions do not apply to the superuser.

When CTIX is running normally, there are two ways to obtain superuser status:

- Log in as user **root**.

- Use the substitute user program, **su**.

Both accesses to superuser status require knowledge of root's user password. Consider root's password sensitive information; change it regularly.

When CTIX is running in single-user mode, the sole user normally has superuser status.

The shell prompt changes to remind you that you are the superuser. Normally the default Bourne shell prompt is a dollar sign ($). When the superuser runs the shell, the default prompt is a pound sign (#).

THE ROOT USER

In the password file, **/etc/passwd**, the user called root has numeric user ID 0; this identifies root as the superuser.

NOTE

Under no circumstances change the name, numeric user ID, or numeric group ID of root. The first user in the file should be root.

Root's password is a sensitive piece of information: anyone who knows it can become superuser. When your system is first booted, root has no password and anyone can become superuser. To provide an initial or changed password, log in as **root** and run **passwd**:

```
passwd
```

passwd prompts for the old password once (if there is one, and if you are not root) and the new password twice.

Root's home directory is /, but this directory should not have any more files in it than necessary.

THE su PROGRAM

To become superuser while logged in as an ordinary user, use **su**, the substitute user program. Type

```
su
```

su will prompt for a password; enter root's password. If the password is verified, **su** runs the shell with its numeric user ID set to Ø, giving the shell the same status as a shell run by root.

To return to normal user status, terminate the **su** shell with **Control-D** (**Finish** key on a Programmable Terminal and Graphics Terminal). You can also return to normal user status by using **su** with your own (or any other) user name, but this does not terminate execution of the superuser shell.

Example 2-1, at the end of this chapter, is an illustration of how to run the **su** program.

SINGLE-USER MODE

Single-user mode prevents ordinary users from communicating with the system. This prevents normal activity that might interfere with disk backup and maintenance.

There are two ways CTIX can go to single-user mode:

- by commands from you, the system administrator
- automatically on start up if CTIX decides it is not safe to go to multiuser mode

Both methods indirectly use the **telinit** command, a command that sends signals to **init**, the process initialization process. Do not change to single-user mode by using **telinit** directly: **telinit** does not give user programs a chance to terminate cleanly. (Appendix B contains information about the **init** program.)

When CTIX is in single-user mode, only one terminal is usable: the terminal that was used to take the system to single-user mode. The user on this terminal has superuser status.

TAKING CTIX TO SINGLE-USER MODE

To take CTIX to single-user mode:

1. Log in as **root**. (This is preferable to using the **su** program and changing your working directory to /, because a root login allows the proper unmounting of all file systems that should be unmounted.)

2. Run **shutdown**. Type

   /etc/shutdown grace

   where grace is the number of seconds that users get to log out by themselves; if grace is omitted, users get 60 seconds. **shutdown**(1M) runs **wall**(1M) to warn the users, **killall**(1M) to terminate the users, and **init**(1M) to change the system mode; this process takes about a minute.

Be sensitive to users' needs when you bring the system to single-user mode. If the system shutdown can be anticipated, notify users in advance. (See "Communicating with Users" in Chapter 9.) If the shutdown is unanticipated, allow a generous grace period if possible.

Example 2-2 is an illustration of how to take the system to single-user mode and how to give root a password.

AUTOMATICALLY GOING TO SINGLE-USER MODE

The CTIX start-up sequence is executed whenever the system is turned on or reset. The start-up sequence includes a check of CTIX file systems. The file system check can have three outcomes:

- Nothing is wrong with any CTIX file system. CTIX goes to multiuser mode.
- One or more file systems is corrupt but can be fixed without destroying any data. CTIX fixes the corrupt file systems, then goes to multiuser mode.

- One or more file systems is so corrupt that no automatic fix is evident. CTIX goes to administrator mode.

When CTIX is in administrator mode, designated terminals prompt for the administrator to log on. If an ordinary user logs on, CTIX promptly logs him or her off. If root logs in, CTIX switches to single-user mode. The single working terminal in single-user mode is the terminal on which root logged in.

You specify which terminals are to be active in administrator mode when you configure the terminals. (See Chapter 5, "Terminals and Modems.")

TAKING CTIX TO MULTIUSER MODE

To return to normal multiuser mode, terminate the shell by pressing **Finish** or **Control-D** in response the shell prompt. The system then prompts, "Run level?"

Type **2** and press **Return**.

Example 2-1

USING THE su COMMAND;
GIVING ROOT A PASSWORD

In this example, you change root's password while logged in as an ordinary user.

```
$ su
Password: ########
# passwd root
Changing password for root
New Password: ########
Retype new password: ########
# ^D
```

Example 2-2

TAKING CTIX TO SINGLE-USER MODE
(Page 1 of 2)

In this example, you are logged in as an ordinary user and you take the system to single-user mode. You give users two minutes to log out.

```
$ su
Password:########
# cd /
# /etc/shutdown 120

SHUTDOWN PROGRAM

Fri Nov 15  10:21:30 PST 1985

Do you want to send your own message? (y or n):
y
Type your message followed by ctrl d....

Taking system down for weekly backups.  You
have 2 minutes to log out.
^D

Broadcast message from root (tty002) Fri
Nov 15 10:22:27...
Taking system down for weekly backups.  You
have 2 minutes to log out.

Broadcast message from root (tty002) Fri
Nov 15 10:22:57...
SYSTEM BEING BROUGHT DOWN NOW ! ! !
All processes being killed.


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

Error logging stopped.

****     SYSCON CHANGED TO /dev/tty002     ****
```

Example 2-2

TAKING CTIX TO SINGLE-USER MODE
(Page 2 of 2)

```
INIT: New run level: S

INIT: SINGLE-USER MODE
Entered single user mode on Fri Nov 15 10:24:17
PST 1985
Ok To Stop Or Reset Processor
#
```

There is a two-minute delay after the first broadcast message and a one-minute delay after the second.

This chapter describes CTIX conventions for system directory organization. It describes the kinds of files that reside in particular system directories.

NOTE

CTIX expects that certain programs and files are located in the conventional places and that the system directory organization has not been altered. Therefore, it is important that you abide by the conventions described here and that you do not rearrange system directories.

Categories of programs and files reside in directories as follows:

- Binary programs that are transported from UNIX System V and are not administrative reside in **/bin** and **/usr/bin**, and in **/usr/lib**. Programs that are needed when the system is in single-user mode and the **/usr** file system is unmounted are in **/bin**. The rest of the binary programs that are not administrative are in **/usr/bin** and **/usr/lib**.

  Binary programs that are not transported from UNIX System V reside in **/usr/local/bin**. Thus, **/usr/local/bin** contains programs from the Berkeley release and programs developed at Convergent Technologies. It is recommended that you place locally-implemented programs in **/usr/local/bin**.

Certain miscellaneous UNIX System V binary programs are in **/usr/lib**.

- Administrative programs and files reside in **/etc**. Some of the programs in **/etc** are binary and some are executable text files (shell programs).

- Special files, files corresponding to input or output devices, reside in **/dev**.

- System library routines reside in **/lib** and **/usr/lib**.

- Data files used by system processes (for example, object files used by the system accounting system) reside in **/usr/lib**.

- Spool files used by **lp** (line printer spooler) and **uucp** reside in **/usr/spool**.

- C language header files reside in **/usr/include** and **/usr/include/sys**. The **/usr/include/sys** directory contains system header files, several of which are specific to MightyFrame CTIX.

NOTE

If your MightyFrame is being used for cross-development (that is, for developing programs to run on a MiniFrame or MegaFrame computer system), you have a **/cross** directory on your system. This directory holds subdirectories containing header files and libraries for use in cross-compilation. Refer to the current Release Notice for information about establishing a cross-development environment.

This chapter describes how to associate entries in the **/dev** directory with physical channels on MightyFrame and Mighty-C hardware.

DIFFERENCES BETWEEN MIGHTYFRAME AND MIGHTY-C

A Mighty-C computer system differs from a MightyFrame computer system with respect to the number of 10-channel or 20-channel expansion boards that can be installed:

- A Mighty-C computer system has two slots for RS-232-C expansion boards. Thus a Mighty-C can have 10 or 20 serial channels in addition to the two serial channels on the main CPU board.

- A MightyFrame computer system has four slots for RS-232-C expansion boards. Thus a MightyFrame can have 10, 20, 30, or 40 serial channels, in addition to the two serial channels on the main CPU board.

plp, tty, AND tp ENTRIES IN THE /dev DIRECTORY

CTIX treats all peripheral devices as files. Each type of device has a driver that manages the device and handles communication between the device and the CTIX kernel. A device's filename is a symbolic name for an index to a kernel table entry that references the driver.

Each CTIX device has a filename in the **/dev** directory. Devices are parallel and serial channels, RS-422 channels, disks, and so forth. This chapter describes three categories of **/dev** entries: **plp**, **tty**, and **tp**.

plp, **tty**, and **tp** entries in the **/dev** directory correspond to the following:

entries for parallel devices

There are two such entries, **/dev/plp** and **/dev/plp1**.

entries for RS-232-C and RS-422 devices

On a Mighty-C computer, there are 22 **/dev/tty** entries for RS-232-C devices and 32 **/dev/tty** entries for RS-422 devices. On a MightyFrame computer (not a Mighty-C), there are 42 **/dev/tty** entries for RS-232-C devices and 32 **/dev/tty** entries for RS-422 devices. RS-232-C devices are printers operating in serial mode, terminals, modems, and other serial devices. RS-422 devices are Programmable Terminals and Graphics Terminals (PTs and GTs).

entries for PT and GT local devices

There are two **/dev/tp** entries for each RS-422 terminal. A PT/GT local device is a serial device attached to an RS-232-C channel on a PT or a GT. To accommodate a local device, the PT or GT must communicate with the MightyFrame via an RS-422 connection.

NOTE

Certain important system messages are sent to **/dev/console**, the console file. The console is not associated with a physical device. (See "Checking the System Console File" in Chapter 13.)

RS-232-C AND RS-422 TERMINALS

It is important to distinguish between RS-232-C terminals and RS-422 terminals. These terms actually describe the kind of communication link between the terminal and a MightyFrame computer. Each RS-232-C line supports a single terminal. Each RS-422 line supports up to eight terminals. Most terminals can only be used on an RS-232-C line. Programmable Terminals and Graphics Terminals can be used on either kind of line.

tty NUMBERS

Each terminal or other device that is connected to a tty line has a three-digit decimal number. tty numbers start from **000**.

tty numbers **000** through **041** designate RS-232-C devices. The terminal number indicates the line used.

tty numbers **256** through **287** designate RS-422 terminals.

NOTE

An RS-422 terminal does not automatically get a certain number; the number is assigned when you turn on the terminal.

Turning the terminal on gives it the lowest RS-422 terminal number not already in use on that RS-422 line. Turning the terminal off frees its number; this number may then be appropriated by some other terminal. Thus, the only way to make sure that an RS-422 terminal gets a specific number is to control the order in which the RS-422 terminals are booted. Normally it is not necessary to make sure that a terminal has a specific terminal number. But note how many RS-422 terminals are in use so you will know what terminal numbers will be allocated.

PT/GT Local Device Numbers

A PT can accommodate one local serial device; a GT, two local serial devices. A PT/GT local device number depends on the tty number of the PT or GT. Thus, PT/GT local device numbers are assigned "on the fly" along with the numbers of their accommodating RS-422 terminals. (If the local device is a printer configured for the **lp** spooling system, you can use two CTIX programs, **mktpy** and **mvtpy**. These programs help with **lp** configuration problems caused by the unpredictability of the terminal number. See "The **lp** Spooling System," in Chapter 6, "Printers.")

Each RS-232-C device attached to a PT or GT has an entry in the **/dev** directory of the form

/dev/tp/aXXX

or

/dev/tp/bXXX

where

a specifies that the device is attached to Channel A on a PT or GT

b specifies that the device is attached to Channel B on a GT

XXX specifies the RS-422 number of the accommodating terminal.

PHYSICAL I/O CHANNELS ON MIGHTYFRAME AND MIGHTY-C COMPUTER SYSTEMS

Entries in the **/dev** directory correspond to physical I/O channels on a MightyFrame, as follows:

- **/dev/plp** corresponds to the parallel channel on the main CPU board.
- **/dev/plp1** corresponds to the parallel channel on either the RS-422 expansion board or the IOP board. (Both MightyFrame and Mighty-C computers can have either an RS-422 expansion board or an IOP board.)
- **/dev/tty000** and **/dev/tty001** correspond to Channel 0 and Channel 1 on the main CPU board, respectively. Channel 0 and Channel 1 support both synchronous and asynchronous RS-232-C devices.

- There are 10 **/dev/tty** numbers for each slot that can hold an RS-232-C expansion board. The slot adjacent to the main CPU board has a designated row of connectors whose tty numbers correspond to **/dev/tty002** through **/dev/tty011**; the slot next to it has the next 10 tty numbers, and so forth.

  The important thing to remember about tty numbers on the RS-232-C expansion boards is that tty numbers depend on the slot number, rather than the board number. For example, if the first 10-channel expansion board is in slot I02 rather than slot I01, the tty numbers on that board begin at **012**.

- **/dev/tty256** through **/dev/tty287** are available for RS-422 terminals. Up to eight terminals can be configured for each cluster channel. The number of RS-422 cluster terminals (called "drops") on each channel and the number of lines in use are specified in the **/etc/system** file. You must edit **/etc/system** to assign tty numbers to each of the channels.

  For example, if there are three drops specified for Channel 0, then the tty numbers on Channel 1, if that line is in use, begin with **/dev/tty259**. If, however, there are eight drops specified for Channel 0, then the tty numbers on Channel 1, if that line is in use, begin with **/dev/tty264**. (See Appendix B and "Tuning Cluster Line Communications" in Chapter 5 for information about the **/etc/system** file.)

Figure 4-1 shows the main CPU board with the parallel channel and two RS-232-C channels on a MightyFrame computer. Figure 4-2 shows the same main CPU board on a Mighty-C computer. Figure 4-3 shows a MightyFrame with a 10-channel RS-232-C expansion board in slot IO1. Figure 4-4 shows a MightyFrame with a 10-channel expansion board in slot IO1 and a 20-channel expansion board in slots IO2 and IO3. Figure 4-5 shows a MightyFrame with a 10-channel expansion board in slot IO1 and an RS-422 expansion board in slot x5.

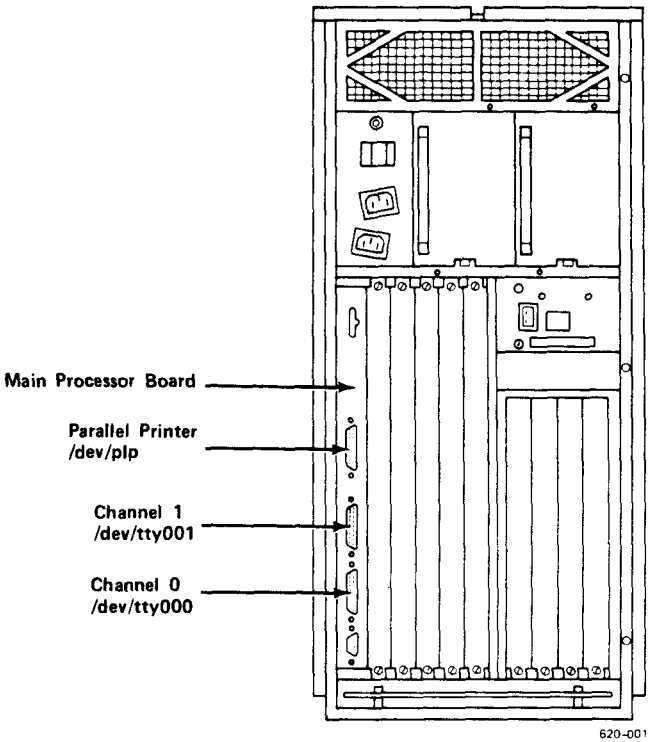


Figure 4-1. The Main CPU Board on a MightyFrame Computer

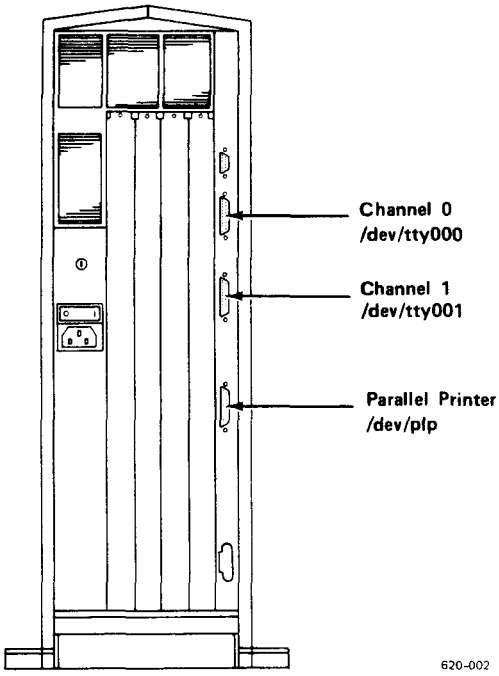


Figure 4-2. The Main CPU Board on a Mighty-C Computer

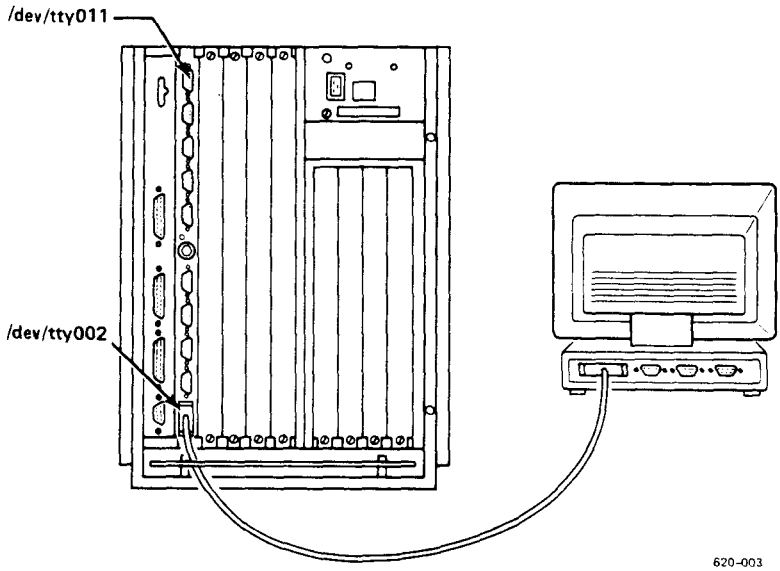


Figure 4-3. A 10-Channel Expansion Board in Slot 101 on a MightyFrame Computer

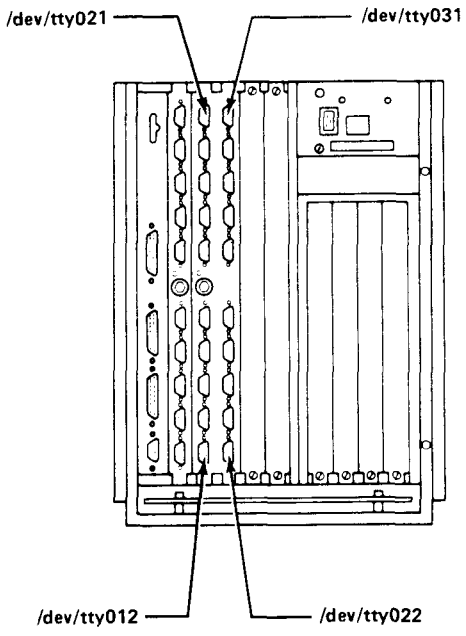


Figure 4-4. A 10-Channel Expansion Board in Slot I01 and a 20-Channel Expansion Board in Slots I02 and I03 on a MightyFrame Computer

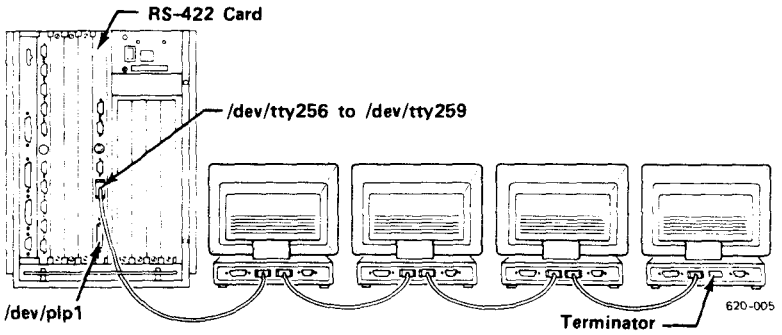


Figure 4-5. A 10-Channel Expansion Board in Slot I01 and an RS-422 Expansion Board in Slot x5 on a MightyFrame Computer

This chapter describes the operating system changes required when you

- configure a new terminal
- remove a terminal from an existing configuration
- configure a new modem

These operating system changes consist of modifying certain configuration files that CTIX uses and ensuring that CTIX responds to the change.

Appendix B provides general reference information for the configuration files discussed here. This chapter gives step-by-step procedures for modifying these files and for making any other required changes to the operating system.

Information regarding cables for peripheral devices is contained in the MightyFrame Installation Manual and the Mighty-C Installation Manual.

NOTE

There is an administrative tool to configure a new terminal. See the CTIX Administration Tools Manual for a description of the tool.

CONFIGURING A NEW TERMINAL

Configuring each new terminal requires the following actions:

1. Determine the new terminal's number or range. If it is an RS-422 terminal, edit **/etc/system**(4) to specify the number of drops on the line. If you change **/etc/system**, reboot the MightyFrame.

2. Make sure there is a suitable entry or entries for the terminal in **/etc/gettydefs**(4), the configuration file for the **getty**(1M) program. (If it is an RS-422 terminal, you do not need to modify **/etc/gettydefs**.)

3. Create or modify an entry in **/etc/inittab**(4), the configuration file for the **init**(1M) program.

4. Create an entry in **/etc/ttytype**(4), the file that lists terminal types.

5. Make sure that **init** rereads its configuration file.

NOTE

It is strongly recommended that you use the following procedure when you edit an important system file that CTIX may read while you are working on it (for example, **/etc/gettydefs**, **/etc/inittab**, and **/etc/ttytype**):

- Make two copies of the current version of the file, using **cp**; one of the copies will be the working copy, the other an "oldfile" for reference.
- Edit the working copy of the file, not the current version of the file.
- Test the working copy of the file, if there is a way to test it (for example, in the case of **/etc/gettydefs**).
- Replace the current file with the working copy, using **mv**.

Refer to Appendix B, "System Initialization Programs and Administrative Files," for information about **init**(1M), **getty**(1M), and the configuration files used by these initialization programs.

Example 5-1, at the end of this chapter, is an illustration of how to configure terminals.

NOTE

The RS-422 driver is loaded by default if the system recognizes the presence of an RS-422 expansion board. (See "Loading Loadable CTIX Device Drivers" in Chapter 7, "CTIX Device Drivers.")

The CTIX kernel configuration that was installed when you loaded the operating system assumes that you are not using RS-422 terminals in administrator mode. To use RS-422 terminals connected to the RS-422 cluster board in administrator mode, you must "hard-code" the RS-422 driver into the kernel configuration. (See "Hard-Coding Loadable Drivers" in Chapter 7, "CTIX Device Drivers.")

The following paragraphs describe the procedural steps in more detail.

Step 1

Determining the Terminal Number

If the terminal is an RS-232-C terminal, select an RS-232-C channel and be sure you know the entry in the **/dev** directory that corresponds to it. (Refer to Chapter 4, "MightyFrame Physical I/O Channels.")

If the terminal is an RS-422 terminal, you cannot know in advance what the terminal number will be unless it is the only RS-422 terminal on a particular channel. You can, however, be sure that the entire range of RS-422 terminals on a channel is configured. Determine which channel it is or will be connected to, and know how many RS-422 terminals are on that line.

If the terminal is an RS-422 terminal, it is recommended that you edit **/etc/system** to reflect the correct number of lines and drops in use: this will improve system performance. Specifying lines and drops affects the number of RS-422 tty numbers available for a particular channel. This is described immediately below in "Tuning Cluster Line Communication." (Refer also to Example 5-1 and the description of **/etc/system** in Appendix B.)

Tuning Cluster Line Communication

/etc/system is a configurable file that allows you to specify certain system parameters without, in most cases, having to remake the CTIX kernel. One set of tuneable parameters specifies the number of RS-422 lines in use and the number of cluster terminals connected to each line.

Step 1 (continued)

The CTIX system default for cluster lines is four cluster lines, each having eight ttys associated with it. You can continue to use this configuration if you have *fewer than 32 cluster terminals*. It is recommended, however, that you tune the system for the exact number of RS-422 terminals and lines you will be using.

The **/etc/system** file consists of a series of sections, each beginning with a section header. The section relating to cluster configuration begins with the keyword section header **!TUNEABLES**. The following two parameters in this section are of interest here:

- The **cl_deflines** parameter sets the number of cluster lines in use.
- The **cl_defdrops** parameter sets the maximum number of cluster terminals that can be clustered on each line.

For example, if you will be using five cluster terminals, two on Channel Ø and three on Channel 1, set **cl_deflines** to **2** and **cl_defdrops** to **3**. When configured this way, Channel Ø has tty256-258 and Channel 1 has tty259-261. (One tty number, tty258, will not be used.)

Step 1 (continued)

To configure **/etc/system** and reinitialize the cluster lines:

1. Become the superuser.

2. Edit **/etc/system** as follows:

   - Uncomment (remove the asterisks from) **cl_deflines** and **cl_defdrops**, the two lines in the file that specify line and drop.

   - Set each of the two parameters; for example,

     ```
     cl_deflines=2
     cl_defdrops=3
     ```

3. Shutdown and reboot the system. (Rebooting initiates the following sequence of events: **mktunedrv**, called from **/etc/drvload**, reads the tuneable parameters information in **/etc/system** and initializes the cluster lines accordingly.)

Step 2

Configuring getty

The CTIX text file **/etc/gettydefs** is used by **getty**, CTIX's terminal initializer, to initialize the line. Each entry specifies a set of communications options and a login message for a particular type of terminal. If the terminal will be active in administrator mode, it requires two entries: one for multiuser mode and one for administrator mode.

/etc/gettydefs is described in detail in Appendix B.

As distributed, **/etc/gettydefs** contains six entries, named by their labels:

- **300**, which defines communications options suitable for an RS-232-C 300-baud terminal and a multiuser mode login message; its next field is **1200**, enabling a user to get the 1200-baud setting by pressing the **Break** key

- **1200**, which defines communications options suitable for an RS-232-C 1200-baud terminal and a multiuser mode login message; its next field is **300**, which enables a user to get the 300-baud setting by pressing the **Break** key

- **9600**, which defines communications options suitable for an RS-232-C 9600-baud terminal and a multiuser mode login message

- **C9600**, which defines communications options suitable for an RS-232-C 9600-baud terminal and an administrator mode login message

- **RS422**, which defines communications options suitable for an RS-422 terminal and a multiuser mode login message

- **CRS422**, which defines communications options suitable for an RS-422 terminal and an administrator mode login message

There should be only one difference between a multiuser mode entry and the corresponding administrator mode entry, aside from the label. The administrator mode entry should have a message (as specified in **gettydefs**) that reminds the user that the system is in administrator mode.

If the terminal is an RS-422 terminal, no changes to **/etc/gettydefs** are required, unless you want to change the login message.

If the terminal is an RS-232-C terminal, follow these steps to configure **gettydefs**:

1. Use **more** or some other CTIX utility to examine the **/etc/gettydefs** file. Using the **termio**(7) pages and Appendix B of this manual as references, decide if there is a suitable entry in the current **/etc/gettydefs** file for your new terminal.

Step 2 (continued)

The baud rate is the most important option: once the new terminal is able to establish communication with the MightyFrame, you can often use "trial and error" to determine if other options need adjustment. (If a user always uses the same terminal, options can be adjusted with an **stty** command in the user's **.profile** file. See Appendix B.)

If your terminal has hardware tabs, you do not need the **tab3** option.

2. If you need to change any initial or final options, use a text editor to copy one of the existing entries and modify it.

   Give the new entry a new label: each label in **/etc/gettydefs** must be unique. (For example, if you copy and modify a 9600-baud entry, you might change the label to 9600B.)

   If you are changing only the login prompt, you do not need to copy an existing entry.

Step 2 (continued)

3. If you want the terminal to be active in administrator mode and you have created or modified a **gettydefs** entry for multiuser mode, duplicate that entry and put a "C" in front of the label for the administrator mode entry.

4. If you have made any alterations to **/etc/gettydefs**, check the correctness of the file. The following command finds errors:

```
/etc/getty -c /etc/gettydefs | more
```

Step 3

Configuring init

The CTIX text file **/etc/inittab** is used by **init**, the CTIX master process spawner. The **/etc/inittab** file, as distributed, is configured to allow only one terminal, ttv000, the terminal connected to Channel 0 on the main CPU board, to log in users. Each new terminal requires that the init table be modified so that the **getty** program monitors the terminal for attempted logins.

To log in to a terminal, there must be a **getty** on the channel. To "put a **getty** on a channel" means to cause the **/etc/getty** program to monitor the channel for attempted logins. To "take the **getty** off the channel" means to cause the channel to be free from interference by **/etc/getty** (for example, when you attach a printer to a channel).

Whether or not a channel has a **getty** on it is determined by (1) the way you have configured **/etc/inittab**, and (2) the current run-level of the system:

- For there to be a **getty** on the channel, the **/etc/inittab** entry for that channel must have the value "respawn" rather than "off" in the third field.

- There is a **getty** on the channel only if the system is currently at a run-level you have specified in the second field of the entry. You can specify multiple run-levels in a single entry.

Step 3 (continued)

See Appendix B for information about the **init** program, the **inittab** file, and how run-levels are defined. See "Configuring Modems" later in this chapter and Appendix B for information about using the **uugetty** program for bidirectional modem connections.

To configure **/etc/inittab** for a new terminal, consult Appendix B and use a text editor to modify the file as follows:

1. Locate the multiuser entry in **/etc/inittab** that applies to the channel to which the terminal is connected. (Refer to Chapter 4, "MightyFrame Physical I/O Channels.")

   If it is an RS-422 terminal, the applicable entry is the first entry for the channel that does not already have a **getty** on it (that is, the first entry that has "off" as the value for the third field).

2. If you have already defined or are now defining more than one multiuser run-level, add the additional run-levels to the second ("rstate") field of the entry. Do not add run-level **6** (administrator mode) to the multiuser entry.

3. Change "off" to "respawn" in the third field of the entry.

4. Make sure the label in the last field is identical to the label for the matching entry in **/etc/gettydefs**. If you created a new entry for this terminal in **/etc/gettydefs**, you now need to edit the last field in **/etc/inittab** so the labels match.

5. If this terminal is to be active in administrator mode, create a new **inittab** entry for this terminal at the end of the **inittab** file. It should be just like the multiuser mode entry except for the following:

   - There should be a "C" in front of the ID in the first field.
   - There should be a "C" in front of the label in the last field.
   - The rstate value (the second field) should be **6**.

The sample below shows two **inittab** entries: one for multiuser mode (run-level **2**), and one for administrator mode. The entries in this sample are appropriate for a 9600-baud terminal that uses the standard 9600-baud entries in **/etc/gettydefs**. The terminal is connected to Channel 0 on the 10-channel expansion board in slot IO1.

```
002:2:respawn:/etc/getty tty002 9600

C002:6:respawn:/etc/getty tty002 C9600
```

Step 4

Configuring Terminal Type

The CTIX terminal type file, **/etc/ttytype**, lists the kind of terminal represented by each terminal number. The CTIX program **tset**, called from **/etc/profile**, uses **/etc/ttytype** to set the TERM environment variable.

The terminal type must be specified as a code in **/etc/termcap**. If the terminal is a PT, the **termcap** code is pt; if the terminal is a GT, the **termcap** code is **gt**.

To configure **/etc/ttytype**:

1. Determine the **termcap** code for the terminal. If you do not already know the code, use a pattern-searching utility such as **grep** to search **/etc/termcap** for the entry that corresponds to your terminal. Entries in **/etc/termcap** consist of fields separated by vertical lines; any field in the correct **termcap** entry is an acceptable code to use in **/etc/ttytype**.

2. Use a text editor to add an entry in **/etc/ttytype** for your terminal.

Example 5-1 includes an illustration of how to search **/etc/termcap** and make an entry in **/etc/ttytype**.

Step 5

Rereading CTIX Terminal Configuration Files

It is *not* necessary to reboot CTIX after you modify **/etc/gettydefs** or **/etc/inittab**. It is only necessary to make **init** reread **/etc/inittab**.

When CTIX is running normally, **init** rereads **/etc/inittab** every time a user logs off and every five minutes. If you do not want to change run-levels, but want **init** to reread **/etc/inittab**, enter the following command:

```
telinit q
```

CAUTION

Use the **telinit** command carefully and precisely. The wrong parameter will stop CTIX suddenly and painfully.

REMOVING TERMINALS

Be absolutely sure that **/etc/inittab** does not refer to any physical devices not in use. It may be difficult for **init** to bring the system to multiuser state if it tries to open nonexistent terminals.

Note that when you physically disconnect an RS-422 terminal, the highest terminal number for that channel is always abandoned. It does not matter which RS-422 terminal you remove. If the removal of an RS-422 terminal is permanent, edit **/etc/system** to reflect the new number of drops and lines.

CONFIGURING MODEMS

Modems require special treatment because they allow two kinds of communication: dial-in connections, and dial-out connections. Dial-in connections are established from outside the system; dial-out connections are established by CTIX. Example 5-2 is an illustration of how to configure **init** for dual use (dial-in/dial-out) connections.

To permit dial-in connections, the line must be monitored by **getty**, just like the directly connected terminals; or by **uugetty**, a special form of the **getty** program. When the modem responds to another modem, **getty** or **uugetty** wakes up and starts the login process.

In a dial-out connection, a user program such as **cu**(1C), **ct**(1), or **uucp**(1C) opens the terminal line and uses the modem to establish a communication link. When the CTIX user program begins, **getty** must not be monitoring the line or it will interfere with the user program. (See Chapter 10, "UUCP," for information on configuring **uucp**.) **uugetty**(1M) is provided for dial-in/dial-out connections.

CONFIGURING getty FOR A DIAL-IN CONNECTION

To configure **getty** or **uugetty** for a dial-in connection, use the procedure described in "Configuring a New Terminal" earlier in this chapter. You will need to create a separate entry in **/etc/gettydefs** for the dial-in modem: copy an existing entry in the file and remove "CLOCAL" from the set of initial options.

It is possible to have terminals with different baud rates using the same modem at different times. If this is the case at your installation, use or modify the linked list of entries in **/etc/gettydefs**. (The distributed version of **/etc/gettydefs** has a linked list for 300 baud and 1200 baud.)

Do not make a dial-in connection eligible for administrator mode, since remote superuser logins are inherently insecure.

CONFIGURING init FOR A DIAL-IN/DIAL-OUT CONNECTION

To establish a dual use for a modem when the dial-out calls are made via **uucp**, do not specify the **/etc/getty** program in the **/etc/inittab** entry for the modem. Instead, specify

/usr/lib/uucp/uugetty

The **uugetty**(1M) program allows dial-in calls, and when the line is free, allows **uucico**, **cu**, and **ct** to use it for dialing out.

If the modem is an intelligent modem, or if there is a **uugetty** at the other end of the line also, you must specify the **-r** argument to **uugetty** in the **inittab** entry: this causes **uugetty** to wait to receive a character from the terminal that will

get the login message, which prevents two **uugettys** from looping. (It requires the terminal user connected to the modem to issue several <carriage return> characters to bring up the login message.)

The **-t** option specifies a timeout and is recommended.

The following **inittab** entry puts **uugetty** on a 1200 baud line used by an intelligent modem and sets a 60-second timeout on the dial-in connection.

```
006:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty006 1200
```

Note that configuring **init** for dial-in/dial-out connections does not complete the dial-out configuration. See Chapter 10, "UUCP," for the procedure to configure a dial-in/dial-out **uucp** modem.

Example 5-1

CONFIGURING TERMINALS
(Page 1 of 7)

In this example, your MightyFrame system has just been installed. You have a Programmable Terminal connected to Channel Ø on the main CPU board, and you have not changed any of the configuration files yet.

You want to add four new terminals: a Freedom 1ØØ connected to Channel Ø on an RS-232-C expansion board in slot IOl, and three new Graphics Terminals. Two of the Graphics Terminals will be connected to Channel Ø on the RS-422 board and the third Graphics Terminal will be connected to Channel 1 on the RS-422 board.

You want the Freedom 1ØØ terminal to be active in administrator mode.

1. First, log in as **root**.

2. Check **/etc/gettydefs** to see if there are terminal definitions suitable for the two types of terminals you are installing.

Example 5-1

CONFIGURING TERMINALS
(Page 2 of 7)

These are the definitions you find in **/etc/gettydefs:**

```
300# B300 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON
OPOST ONLCR B300 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB
3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\nlo
gin: #1200

1200# B1200 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B1200 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\
nlogin: #300

9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\
nlogin: #9600

RS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL I
XON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO ECH
OE #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\nl
ogin: #RS422

C9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IX
ON OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
 TAB3 #MightyFrame CTIX (tm: Convergent Technologies)\n********\n
ADMIN MODE\n**********\nlogin: #C9600

CRS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL
IXON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO EC
HOE #MightyFrame CTIX (tm: Convergent Technologies)\n********\nAD
MIN MODE\n**********\nlogin: #CRS422
```

620-008

① suitable for Freedom 100 terminal and a PT or GT in RS-232-C mode

② suitable for PTs and GTs in multiuser mode

③ suitable for Freedom 100 terminal in administrator mode

There are, in fact, suitable entries, both in multiuser and administrator modes, for the Freedom 100 terminal. The significant options are 9600 baud rate (B9600), no parity (IGNPAR), and 8 bits (CS8).

Example 5-1

CONFIGURING TERMINALS
(Page 3 of 7)

Since there is a definition for the Graphics Terminals in multiuser mode, you do not need to change any of the initial or final options in **/etc/gettydefs**.

You do, however, want to change the login prompt.

Make two copies of **/etc/gettydefs**--one to keep as an oldfile, and one for editing:

```
# cp /etc/gettydefs /etc/ogettydefs
# cp /etc/gettydefs /etc/ngettydefs
```

Now edit **ngettydefs**. When you have finished, the copy of **/etc/gettydefs** will look like this:

```
300# B300 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON
OPOST ONLCR B300 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB
3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #1200

1200# B1200 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B1200 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #300

9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #9600

RS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL I
XON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO ECH
OE #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #RS422

C9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IX
ON OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n********\nADMIN
MODE\n**********\nlogin: #C9600

CRS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL
IXON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO EC
HOE #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n********\nADMIN
MODE\n**********\nlogin: #CRS422
```

620-009

Example 5-1

CONFIGURING TERMINALS
(Page 4 of 7)

Run **/etc/getty -c** on the copy of the **gettydefs** file to check it for errors:

```
# /etc/getty -c /etc/ngettydefs | more
```

Overwrite the current **gettydefs** file with the edited copy:

```
# mv /etc/ngettydefs /etc/gettydefs
```

3. Next, tune the cluster line and drop configuration. Remove the asterisks in front of the lines specifying lines and drops (**cl_deflines** and **cl_defdrops**) and set new values for each of the two parameters. When you are finished editing the file, the affected portion will look like this:

```
cl_deflines=2
cl_defdrops=2
```

This configuration assigns tty256 and tty257 to Channel Ø, and tty258 and tty259 to Channel 1.

Bring the system to single-user mode and reboot. When the system comes up, become superuser again.

4. Make two copies of **/etc/inittab**--one for an oldfile and one for editing. For the Freedom 1ØØ terminal, identify the entry for Channel Ø on the RS-232-C expansion board; it is tty ØØ2. Change "off" to "respawn." Then make a duplicate copy of the entry for administrator mode. To the duplicate entry, add a "C" before the ID (first field) and a "C" before the label in the last field.

Example 5-1

CONFIGURING TERMINALS
(Page 5 of 7)

For the three Graphics Terminals, the appropriate entries are tty256, tty257, and tty258. Change "off" to "respawn" for the three entries. Add a comment to the **inittab** copy to indicate line and drop specifications in **/etc/system**. Then overwrite the current **inittab** file with the edited copy.

The entries in question are shown below:

```
000:12345:respawn:/etc/getty tty000 9600
001:2:off:/etc/getty tty001 9600

: 232 board #1 ports 2 - 11
002:2:respawn:/etc/getty tty002 9600
003:2:off:/etc/getty tty003 9600
004:2:off:/etc/getty tty004 9600
005:2:off:/etc/getty tty005 9600
006:2:off:/etc/getty tty006 9600
007:2:off:/etc/getty tty007 9600
008:2:off:/etc/getty tty008 9600
009:2:off:/etc/getty tty009 9600
010:2:off:/etc/getty tty010 9600
011:2:off:/etc/getty tty011 9600


: RS422 terminals
: /etc/system configured for 2 lines, 2 drops
: tty256-257 on Channel 0
: tty258-259 on Channel 1

256:2:respawn:/etc/getty tty256 RS422
257:2:respawn:/etc/getty tty257 RS422

258:2:respawn:/etc/getty tty258 RS422
259:2:off:/etc/getty tty259 RS422

: 422 terminals below (tty260-287) not
: configured in /etc/system
260:2:off:/etc/getty tty260 RS422
261:2:off:/etc/getty tty261 RS422
262:2:off:/etc/getty tty262 RS422
263:2:off:/etc/getty tty263 RS422



: administrator consoles
C000:6:respawn:/etc/getty tty000 C9600
C002:6:respawn:/etc/getty tty002 C9600
```

620-010

Example 5-1

CONFIGURING TERMINALS
(Page 6 of 7)

5. Now you want to add entries to **/etc/ttytype**, but you do not know the **termcap** code for the Freedom 100 terminal. Use **grep** to search **/etc/termcap**:

```
① # grep freedom /etc/termcap
  v9.3277|freedom.3277|freedom 100.3277:\
② v9|freedom|freedom 100:\
```

① grep searches the terminal data base **/etc/termcap** for entries containing the string "freedom".

② The Freedom 100 terminal is identified in this line. Each of the fields, separated by a vertical line, contains an acceptable code for the **/etc/ttytype** entry.

Edit a copy of **/etc/ttytype** so there is an entry for each terminal you have connected. When you have finished, **/etc/ttytype** will look like this:

```
pt       000
freedom  002
gt       256
gt       257
gt       258
```

Example 5-1

CONFIGURING TERMINALS
(Page 7 of 7)

6. The last step is to instruct **init** to reread **/etc/inittab**. Do this by entering the following command:

   # telinit q

   Power on the new terminals. All terminals now have login messages.

Example 5-2

CONFIGURING A MODEM
(Page 1 of 3)

In this example, you want to put a Bizcomp 1Ø12 intelligent modem on your MightyFrame system. You will use the modem for a UUCP dial-in/dial-out communication link. This example configures **init** and **getty** for such a connection, but does not complete the UUCP configuration. (See Chapter 1Ø, "UUCP," for more information on UUCP configuration.)

This example assumes that you have configured the four new terminals in Example 5-1.

1. Log in as **root**.

2. Look at the 12ØØ-baud entry in **/etc/gettydefs**. The one in your current file is close to what you need, but you want a definition that excludes "CLOCAL" from the list of initial options.

   Make two copies of **/etc/gettydefs** as you did in Example 5-1; edit the copy, **/etc/ngettydefs**.

   Working with **/etc/ngettydefs**, copy the 12ØØ-baud definition, remove CLOCAL from the new entry, and give the entry a new label (M12ØØ). Copy the 3ØØ-baud definition and give it a new label (M3ØØ).

Example 5-2

CONFIGURING A MODEM
(Page 2 of 3)

Your copy of the **/etc/gettydefs** file now looks like this:

```
300# B300 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON
OPOST ONLCR B300 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB
3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #1200

M300# B300 IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON OPOST
ONLCR B300 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB3 #\n\
n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #M1200

M1200# B1200 CIXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON OPO
ST ONLCR B1200 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB3
#\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #M300

1200# B1200 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B1200 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #300

9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #9600

RS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL I
XON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO ECH
OE #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n\nlogin: #RS422

C9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IX
ON OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n********\nADMIN
MODE\n**********\nlogin: #C9600

CRS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL
IXON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO EC
HOE #\n\n\n\n\n\t\t\tPublic Library MightyFrame\n********\nADMIN
MODE\n**********\nlogin: #CRS422
```

620-011

Run **/etc/getty -c** on the **gettydefs** copy to test it for errors:

```
# /etc/getty -c /etc/ngettydefs | more
```

Example 5-2

CONFIGURING A MODEM
(Page 3 of 3)

Overwrite the current **gettydefs** file with the edited copy:

```
# mv /etc/ngettydefs /etc/gettydefs
```

3. Next, edit a copy of **/etc/inittab**. The modem will be connected to Channel 1 on the main CPU board; the entry is **tty001**.

   Change "off" to "respawn"; change the label to "M1200". Change the **/etc/getty** invocation to **/usr/lib/uucp/uugetty** with the **-t** and **-r** arguments.

   The affected part of the **inittab** file looks like this:

```
000:12345:respawn:/etc/getty tty000 9600
001:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty001 M1200
```

620-012

   Overwrite **/etc/inittab** with your copy.

4. You do not need an **/etc/ttytype** entry, since several types of 1200-baud terminals will be dialing into the MightyFrame. The dial-in users can set the TERM variable after they are logged in, or they can set it in their **.profile** files.

5. Last, have **init** read **/etc/inittab** as you did in Example 5-1.

CTIX provides two incompatible print spooling programs: **lp** and **lpr**. **lp** supports multiple printers and has advanced features. **lpr** is simple but adequate for a system with a single printer.

NOTE

Although it is somewhat more difficult to configure, **lp** is the recommended spooling program. This chapter provides a lengthy explanation of how to configure the **lp** spooling system. A complete example is provided at the end of the chapter.

There is an administrative tool for configuring the **lp** spooling system. See the CTIX Administration Tools Manual for a description of the tool.

CONFIGURING THE lpr SPOOLING SYSTEM

lpr can use two kinds of printers:

- Centronics-compatible parallel printer. This is connected to the special parallel printer line.
- Serial printer. This is connected to any RS-232-C terminal line.

The parallel printer line is associated with the special file **/dev/plp**.

An RS-232-C line used by a receive-only printer must not be monitored for logins. Examine **/etc/inittab** and verify the absence of any reference to the line you intend to use. Terminal line usage is discussed in Chapter 5, "Terminals and Modems."

To make **lpr** use the correct line, associate the special file **/dev/lp** with the correct line. If the printer is connected to the parallel printer line, type

```
rm /dev/lp
ln /dev/plp /dev/lp
```

If the printer is connected to an RS-232-C terminal line, type

```
rm /dev/lp
ln /dev/ttyttt /dev/lp
```

where

ttt is the three-digit terminal number.

As shipped, MightyFrame CTIX has **/dev/lp** associated with the parallel printer line.

Use **ls** to verify that a **/dev/lp** file is associated with the correct line. Two special files are associated with the same line if they have they same i-number. (For a discussion of i-numbers, see Appendix A.) To see if **/dev/lp** has the same i-number as **/dev/plp**, type

```
ls -li /dev/plp /dev/lp
```

This produces a listing like this:

```
3438 crwxrwxrwx 2 root root      6, 0 /dev/plp
3438 crwxrwxrwx 2 root root      6, 0 /dev/lp
```

Only the i-number (first item on each line) and the special file name (last item on each line) are of interest here. Verify that the two i-numbers are identical; if they are not, **/dev/lp** is not associated with **/dev/plp**.

To see if **/dev/lp** has the same i-number as an RS-232-C terminal line, type

ls -li /dev/ttyttt **/dev/lp**

where

ttt is the three-digit terminal number.

This produces a listing much like that of the other version of the **ls** command. Again, compare i-numbers: they must be the same or **/dev/lp** is not associated with the indicated line.

If a serial printer is used and the printer's communication requirements do not match CTIX's defaults, you must arrange for CTIX to set and hold terminal options. To do this, add the following two lines at the end of **/etc/rc**:

nohup sleep 2000000 < /dev/lp &
stty options **< /dev/lp**

where options is a list of valid (and required) **stty** options: see **stty**(1).

CONFIGURING THE lp SPOOLING SYSTEM

The **lp** (line printer) spooling system is a group of programs and commands that (1) allow you, as the administrator, to customize the operation and management of printers and (2) permit users to queue and execute print requests. The term line printer applies to all types of hard copy printers: dot-matrix, daisy wheel, laser, login, and so forth.

The following paragraphs describe

- concepts and commands relating to the **lp** spooling system
- how to configure and modify the **lp** spooling system
- how the **lp** and **lpsched** programs work

OVERVIEW OF lp: CONCEPTS AND COMMANDS

lp Concepts

Devices and Printers

lp makes a distinction between devices and printers.

A device is an entry in the **/dev** directory that, except in the case of **/dev/null**, corresponds to a channel on the MightyFrame. A device is always represented by a full CTIX pathname (for example, **/dev/tty002**, **/dev/plp** or **/dev/null**).

A printer is a printer name that you associate with a particular device.

For example, if you reconnect a printer to a different channel, you must change the configuration of the spooling system so that the printer becomes associated with the new device.

Printer Classes

A class is a name given to an ordered list of printers. A class must have at least one printer assigned to it. A printer may be a member of zero or more classes.

Destination

A destination is a printer or a class. One destination may be designated as the system default destination. **lp** directs all output to the default destination unless a user specifies a different destination. If the destination is a class, output directed to the destination is printed by the first available class member. A user can also specify a personal default destination by setting the LPDEST environment variable; a personal default takes precedence over the system default.

Request

A request consists of names of files to be printed and options from the **lp** command line. Each invocation of **lp** creates one request.

Interface Program

An interface program is a shell script or other executable program that is responsible for retrieving files to be printed and printing them on the device associated with the printer. Each printer requires such a program to set options on the device, allow printer-dependent options to be specified on the **lp** command line, format output for printing, and send the formatted output to the device. Several model interface scripts are provided in the software distribution and reside in **/usr/spool/lp/model**.

Login Terminals, Hardwired Printers, PT and GT Local Printers, and Remote Printers

There are essentially four categories of printers in the **lp** spooling system:

- A login terminal is a hard-copy terminal that is sometimes used as a terminal and other times used as a spooled printer. Therefore, this terminal must be configured so that **lp** does not use it as a printer when someone else is using it as a terminal.

- A hardwired printer is not a login terminal; it is directly connected to either of the following:

  - a serial interface (RS-232-C) on the MightyFrame

  - a parallel interface on the MightyFrame

- A PT or GT local printer is a hard-wired printer connected to an RS-232-C channel on a PT or GT. (The PT or GT must be connected to the MightyFrame with an RS-422 cable.)

- A remote printer is accessed via a network, a modem, or both. For the **lp** spooling system to access a printer via a network, the printer must be known both to **lp** on the remote host (where it is local) and to **lp** on your local host.

Commands for General Use. The following commands do not require special permission to execute:

lp initiates printing of named files (or "standard input"). Creates an output request and returns a request ID to the user (see "How the **lp** and **lpsched** Programs Work" below).

cancel cancels output requests created by **lp**. If the argument supplied to **cancel** is a request ID (which can be obtained using **lpstat**), the associated request is canceled, even if it is currently printing. If the argument supplied to **cancel** is a printer name, the currently printing request is canceled and the next queued request is printed.

lpstat reports current status of **lp** spooling system. With no arguments, prints the status of all requests generated by **lp** for the user invoking **lpstat** (status information includes request ID). See **lpstat**(1) for a complete list of options.

disable prevents the scheduler **lpsched** from routing requests to printers.

enable allows **lpsched** to route output requests to printers.

Commands for lp administrators. The commands listed below require special permission to execute: you must have superuser or **lp** privilege. (These commands are described in more detail in "How to Configure and Modify the **lp** Spooling System" below).

lpadmin configures the **lp** spooling system: defines printers, classes, and default destinations; associates each printer with a device and with an interface program. Most of **lpadmin's** command line options cannot be used when **lpsched** is running.

lpsched routes output requests to interface programs that do the printing on devices. Routing involves matching requests to printers (monitoring the status of printers to find the first available printer in a class, and so forth). Invoking the **lpsched** command starts the **lpsched** program.

lpshut stops **lpsched** from running. All printing activity is halted. Previously issued requests remain queued, and **lp** continues to create and queue new requests; any requests that are printing when you invoke **lpshut** are reprinted in their entirety when you restart **lpsched.** You must run **lpshut** before using most features of **lpadmin** and before running **lpmove.**

accept allows **lp** to accept output requests for destinations.

reject prevents **lp** from accepting requests for destinations.

lpmove moves output requests from one destination to another. **lpmove** can move requests for an entire class of printers at one time. This command cannot be used when **lpsched** is running.

HOW TO CONFIGURE AND MODIFY THE lp SPOOLING SYSTEM

The following paragraphs describe how to

- set up a hardwired device as an **lp** printer
- set up a remote device as an **lp** printer
- set up a login device as an **lp** printer
- modify an existing destination

Commands to configure and maintain the **lp** spooling system require that you be the superuser or have **lp** privilege.

NOTE

It is recommended that you use only one spooling system, either **lp** or **lpr**. There are commented-out entries for both **lp** and **lpr** in **/etc/rc**. Edit **/etc/rc** to remove the comment signs in front of the lines relating to **lp** (see Appendix B).

Also edit **/etc/halt** and **/etc/shutdown** to remove the comment signs in front of the lines relating to **lp**.

Setting Up Hardwired Devices as lp Printers

Example 6-1 is an illustration of how to set up several hardwired devices on the **lp** spooling system.

Follow the steps below to set up hardwired devices:

1. Give **lp** exclusive access to the device.

2. If it is a serial device, make sure there is no **getty** on it. (You must have superuser privilege to modify **/etc/inittab**.)

3. Determine if one of the model interface programs in **/usr/spool/lp/model** (or an interface you have already written or modified) is suitable for the printer. If there is no suitable interface, make a copy of an existing interface and modify it. (Normally when there is no suitable interface, it is best to start with the **dumb** model and modify it.) Give **lp** ownership of any new or copied interface programs. Make sure all interface programs you intend to use as models (**-m** option to **lpadmin**) reside in **/usr/spool/lp/model**.

4. If your **lp** spooling configuration will have more than one printer, decide if and how you want to define printer classes and which printer or class will be the default.

5. Run **lpshut** to turn off the scheduler.

6. Run **lpadmin** to associate the new printer with a device, an interface program, and a class, if it will be a class member. If the new printer or its class is the new system default destination, run **lpadmin** to define it as the default destination.

7. Allow **lp** to accept requests for the new destination and enable printing on the new printer.

8. Restart the scheduler.

NOTE

It is strongly recommended that you use the following procedure when you edit an important system file that CTIX may read while you are working on it (for example, **/etc/inittab**):

1. Make two copies of the current version of the file, using **cp**--one of the copies will be the working copy, the other one an "oldfile" for reference.

2. Edit the working copy of the file, not the current version of the file.

3. Replace the current file with the working copy, using **mv**.

The following paragraphs describe the procedural steps in more detail.

Step 1

Giving lp Exclusive Access to the Device

You must give **lp** exclusive access to the device to prevent non-**lp** processes from interfering with **lp's** ability to write to the device. If it is a serial device, enter

```
chown lp /dev/ttyxxx
chmod 600 /dev/ttyxxx
```

where

xxx	is the three-digit device number; for example, **002**.

If it is a parallel device, enter the same commands, with the exception that the device is **/dev/plp** or **/dev/plp1**.

Step 2

Making Sure There Is No getty on the Channel, If It Is a Serial Device

Since this is a hardwired device and not a login terminal, make sure **/etc/getty** is not trying to log users in at this device. Examine **/etc/inittab** and make sure that the value of the third field of the entry for the particular serial line is "off." For example,

```
002:2:off:/etc/getty tty002 9600
```

If you change the entry for this device to remove the **getty**, make sure the **init** program rereads **/etc/inittab**:

```
telinit q
```

It is the responsibility of the interface program to establish the correct **stty** values each time **lpsched** opens the device for writing.

Step 3

The Interface Program

Interface programs can be simple or elaborate, but each program must minimally

- Perform an **stty** on the device, if it is a serial device, to set the baud rate (and other **tty** options) and to merge the standard input with the standard output. (Standard input and standard output must be merged so that **stty** is able to affect the device opened for standard output, rather than standard input, which is the default.)

  stty options **<&1**

- Use a filter (for example, **cat**) to read the files to be printed on the output device; make the number of copies and print the title specified by the user.

- Exit with a code that indicates to **lpsched** whether or not the print job was successful as follows:

  exit Ø

If you use the **dumb** model interface script with a serial printer, you must add the **stty** command as the first executable line. Example 6-1 at the end of this chapter, illustrates how to do this.

Step 3 (continued)

lpsched passes the following positional parameters to the interface:

id user title copies options file ...

where

id is the request ID returned by **lp**.

user is the logname of the user who made the request.

title is the optional title specified by the user (**-t** option to **lp**).

copies is the number of copies requested by the user (**-n** option to **lp**).

options is a list of class-dependent or printer-dependent options separated by blanks and specified by the user (**-o** option to **lp**).

file is the full pathname of a file to be printed. There may be more than one file; each file is a parameter.

Typically, the interface program prints a banner with the user's name, another banner with the user-specified title, the request ID, the date the print job was executed, and the name of the printer (the printer name is passed to the interface as $Ø). The model script **dumb** does each of these.

Step 3 (continued)

Interface programs can provide much more elaborate filters to set margins and pitch, do text wrap-around, format columns, produce troff copy, and so forth.

Interface programs can also provide error-checking: an exit code of Ø means the print job was successful; a status code in the range 1 to 127 is interpreted by **lpsched** to mean that a problem was encountered in printing this particular request (for example, too many nonprintable characters). Note that a nonzero exit code does not disable the printer and does not in any other way affect future requests. **lpsched** notifies users by mail if there was an error in printing the request. (Do not use codes greater than 127; these are reserved for internal use by **lpsched.**)

If an interface uses mechanisms that can fail and this failure can affect future print jobs (for example, a malfunctioning or missing filter), the interface should disable the printer in the event of such an error. This prevents future print requests from getting lost. (When a busy printer is disabled, the interface program is terminated with signal 15.)

When you have finished creating or modifying the interface, make sure **lp** has ownership of it. If you created the interface as **root**, rather than **lp**, type

chown lp interface

where interface is the file name of the interface program.

Step 3 (continued)

You can add any new or modified interface program to **/usr/spool/lp/model**. If the interface resides in this directory, you can specify the interface with the **-m** option when you run **lpadmin**. There are two other ways to specify an interface--as the same interface an existing printer uses (**-e** option), or as a program that does not reside in **/usr/spool/lp/model** (**-i** option). The three options are described in more detail below.

Note that when you run **lpadmin**, it makes a copy of your interface for the **lp** spooling system to use. (This allows you to edit a model while **lp** is active.) Thus, if you make any changes to the interface after you first run **lpadmin**, you must run **lpadmin** again to update **lp**'s copy.

Steps 4, 5, and 6

Configuring the lp Spooling System with lpadmin

To configure the **lp** spooling system with **lpadmin**, first run **lpshut**:

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

Then run **lpadmin** once for each printer you are introducing and once to establish the system default destination.

If the interface program you have selected resides in **/usr/spool/lp/model** and the printer is not a "local printer," the command is

```
/usr/lib/lpadmin -pprinter { -v/dev/plp | -v/dev/plpl | -v/dev/ttyxxx } -mmodel [-cclass] [-h]

/usr/lib/lpadmin -d[dest]
```

where

printer is the arbitrary name you are assigning to this printer. It must be no longer than 14 characters, must consist solely of alphanumeric characters and underscores, and must uniquely identify this destination (that is, it can not be the same as any other printer or any class in the **lp** spooling system).

{**/dev/plp** | **/dev/plpl**} specifies one of the parallel channels. **plp** is the parallel device on the main CPU board, **plpl** is the parallel device on the IOP board or RS-422 expansion board.

Steps 4, 5, and 6 (continued)

xxx — is the three-digit device number you are associating with this printer, if it is a serial printer; for example, **ØØ2**.

model — is the interface program in **/usr/spool/lp/model** that you are associating with this printer.

class — is the class name. A printer can be a member of zero or more classes. You must run **lpadmin** once for each class to which you assign this printer.

-h — is the optional flag specifying that the printer is associated with a hardwired device.

[dest] — is the printer or class that will be the default destination. To configure the **lp** spooling system for no system default, specify **/usr/lib/lpadmin -d**. Note that the **-d** argument is required, whether or not there is a default destination.

Steps 4, 5, and 6 (continued)

For example, to associate a new printer, prl, with an interface whose pathname is **/usr/spool/lp/model/ds** and a device **/dev/tty00**, and to make prl the default destination, the commands are

```
/usr/lib/lpadmin -pprl -v/dev/tty00 -mds

/usr/lib/lpadmin -dprl
```

If the interface program you want to associate with the printer is the same interface an existing printer uses, you can run **lpadmin**, as described earlier, substituting **-m**model with **-e**printer, where printer is the name of the printer already associated with the interface.

If the interface program is not in **/usr/spool/lp/model**, run **lpadmin**, as described above, substituting **-m**model with **-i**interface, where interface is the full pathname of the program you are associating with this printer.

Steps 7 and 8

Bringing the Printer Online and Restarting the Scheduler

To allow **lp** to accept requests for the new destination and enable printing on the new printer, type the following two commands:

/usr/lib/accept dests

enable printer

where

dests — is one or more new printers and/or classes you want **lp** to accept; if you specified a new class with a new printer when you ran **lpadmin**, specify both the printer and class in the **accept** command.

printer — is the name of the printer you specified in the **lpadmin** command.

To restart the scheduler, type

/usr/lib/lpsched

Setting Up Remote Devices as lp Printers

For the **lp** spooling system to access a remote printer, the printer must be configured for **lp** on the remote host (where it is local) and to **lp** on your local host. On the remote host there is an **lp** configuration that specifies an actual device for the printer; for example, **/dev/tty00**. On your host, the printer is associated with **/dev/null**. Your **lp** configuration must also associate the remote printer with an interface program that calls the network. A model script for use with **uucp** is provided in **/usr/spool/lp/model**; this script can also be used with Ethernet.

To make **lp** on your local host equivalent to **lp** on the remote host, add **lp** to the **Permissions** file in **/usr/lib/uucp**.

Follow the steps below to set up remote devices:

1. Determine if one of the model interface programs in **/usr/spool/model** (or an interface you have already written or modified) is suitable for the printer; if there is no suitable interface, make a copy of an existing interface program and modify it. Give **lp** ownership of the new interface program. It is strongly recommended that you use or modify the **uucp** model script. It is more secure to access a remote printer via the UUCP network than to access the printer via the Ethernet network directly.

Refer to the earlier discussion on creating an interface program for a hardwired device. The differences from hardwired devices are the following:

- Since the device is **/dev/null**, there is no **stty** performed on it.
- The interface must forward the files to be printed to the remote printer.

2. Decide if and how you want to define printer classes and which printer or class will be the system default destination.

3. Run **lpshut** to turn off the scheduler.

4. Run **lpadmin**, as described in reference to hardwired devices, with this exception: the device is **/dev/null**.

   For example, to associate a new remote printer, pr2 (not the system default destination), with an interface whose pathname is **/usr/spool/lp/model/enet**, the command is

   /usr/spool/lpadmin -ppr2 -v/dev/null -menet

5. Run **accept**, **enable**, and **lpsched** as described in Steps 7 and 8 for hardwired printers.

Setting Up Login Devices as lp Printers

Configuring the **lp** spooling system for login devices must provide for the possibility that login terminals will be used as terminals as well as spooled printers. It is necessary to disable login printers when they are used as terminals to

prevent print jobs from being queued for these devices. Users should therefore issue the **disable** command when they appropriate such a device for a terminal. Correspondingly, users must enable previously disabled login terminals when they want to use them as spooled printers.

When you run **lpadmin** to introduce a login terminal to the **lp** spooling system, use the **-l** argument; this indicates to **lpsched** that it must automatically disable this printer each time **lpsched** starts running.

If login terminals move from device to device in your installation, you may want to do the following:

- Associate **/dev/null** with login terminals when you introduce them to **lp** (with **lpadmin**).

- Run **lpadmin** again each time a terminal is used as a spooled printer to associate it with a particular device.

The procedure outlined below for configuring and using login terminals as spooled printers assumes that such printers do not travel among devices.

To set up a login terminal as a possible device for spooled printing:

1. Prepare an interface program for each terminal according to the instructions given for hardwired devices in "Setting Up Hardwired Devices as **lp** Printers." Decide if and how you want to define printer classes.

2. Run **lpshut** to turn off the scheduler.

3. Run **lpadmin** as you would for a hardwired device, but use the **-l** argument.

4. Run **lpsched** to restart the scheduler.

To bring a previously introduced login terminal on line as a spooled printer:

1. Issue the **accept** command so the terminal can accept requests.

2. Log in to the terminal. (In this way, you avoid having to modify **/etc/inittab** each time you change a login terminal's use.)

3. Enable the terminal for printing.

4. Disable the terminal from printing when you are finished using it as a spooled printer.

Ordinary users can enable and disable printers. If you want to keep login terminals from being used as spooled printers for a period of time, you may want to ensure this by issuing the **reject** command for each such terminal.

Setting Up PT/GT Local Printers as lp Printers

A PT/GT "local printer" (a printer connected to an RS-422 terminal's RS-232-C channel) presents some complications for the **lp** spooling system. This is because CTIX assigns device numbers for RS-422 terminals "on the fly": a terminal's device number is the lowest available tty number on the particular channel it is connected to when the terminal is powered on.

The fact that a printer's device number is unpredictable necessitates a configuration similar to that of a login terminal. It also requires users to execute the **mktpy** or **mvtpy** commands when they power on their terminals. **mktpy** installs a previously introduced printer on an RS-422 terminal's local channel. **mvtpy** updates a **mktpy** installation by changing the device association when the RS-422 terminal's device number changes; **mvtpy** should be run when the terminal is powered off and back on again.

To introduce a PT/GT local printer to the **lp** spooling system:

1. Prepare an interface program for the printer according to the instructions given for hardwired devices in "Setting Up Hardwired Devices as **lp** Printers" earlier in this chapter.

2. Run **lpshut**.

3. Run **lpadmin** as you would for a hardwired device, with the following exceptions:

   - Associate the printer with **/dev/null**.

   - Use the **-l** option: this disables the printer when the system is rebooted.

4. Issue the **accept** command so the printer can accept requests.

5. Restart the scheduler.

To use the printer, a user executes the **mktpy** command: this enables the printer and changes its device association to the current local device of the user's RS-422 terminal. Each time the MightyFrame is rebooted the local printer becomes disabled; the user must, therefore, invoke **mktpy** after each system reboot.

To install a previously introduced local printer, the user types

/usr/local/bin/mktpy printerA printerB [tty]

where

printerA — is the name of a printer attached to serial Channel A on a PT or GT.

printerB — is the name of a printer attached to serial Channel B on a GT. If there is only one printer and it is attached to Channel B, the user must insert a place marker for printerA in the form of a pair of double quotes (" ").

tty — is the RS-422 tty number. This parameter is required only when the command is issued from a terminal other than the one connected to the printer.

For example, to install printer "diablo" on an RS-422 terminal's serial Channel A, the user types from the same terminal

/usr/local/bin/mktpy diablo

If **mktpy** is invoked without arguments, it prompts for printer names and a tty number. (See your CTIX operating system manual for more information about **mktpy** and **mvtpy**.)

If a user's tty number changes when the terminal is powered off and on, the user must run **mvtpy**. (**mktpy** cannot change the device association for a printer that is already enabled.)

First, the user must determine the new device number by typing the following:

tty

Then, the user executes **mvtpy**.

/usr/local/bin/mvtpy printer tty 1
/usr/local/bin/mvtpy printer tty 2

The first form of the command is for a printer attached to Channel A, the second for a printer attached to Channel B.

Modifying Existing Destinations

These are typical administrative modifications to the **lp** spooling system:

Changing a device association

This can be done with **lpsched** running. Simply run **lpadmin** with two arguments: **-p**printer **-v**device.

Adding a previously introduced printer to a new or existing class

Stop **lpsched** and run **lpadmin** with two arguments: **-p**printer **-c**class.

Removing a printer from an existing class

If the printer is not the last remaining member of this class, stop **lpsched** and run **lpadmin** with two arguments: **-p**printer **-r**class.

Note that removing the last remaining member of a class causes the class to be deleted. No destination can be removed if it has pending requests. Therefore, if there are pending requests, either cancel the requests or use **lpmove** to move them to a different destination before you run **lpadmin**.

To move requests, stop **lpsched** and invoke **/usr/lib/lpmove** in one of two ways

- With the arguments *requests dest*, where *requests* are the request ID numbers returned by **lp** and *dest* is the new destination. (Use **lpstat** to obtain request ID numbers.)

- With the arguments *destold destnew*. This second form moves all requests queued for one destination to another destination.

Removing a printer or class from the **lp** spooling system

Stop **lpsched**, cancel or move all pending requests, and run **lpadmin** with one argument: **-x***dest*, where *dest* is the printer or class you are removing.

If *dest* is the system default destination, run **lpadmin** again to assign a different default destination or to specify that there is no default. Note that the removal of a class never implies the removal of printers.

Changing the system default destination

This can be done with **lpsched** running. Run **lpadmin** with one argument: **-d***dest*, where *dest* is a printer or class.

Changing an interface program

When you (1) make changes to an interface program, or (2) want to associate a printer with an interface program having a *different filename*, stop **lpsched** and run **lpadmin**.

In the first event, the arguments to **lpadmin** are **-p**printer and either **-m**model or **-i**interface.

In the second event, the arguments to **lpadmin** are **-p**printer and either **-m**model, **-i**interface, or **-e**printer.

Make sure **lp** has ownership of all interface programs.

For information on interface programs and related **lpadmin** options, see "Setting Up Hardwired Devices as **lp** Printers" earlier in this chapter.

HOW THE lp AND lpsched PROGRAMS WORK

Printing a Request

The **lp** program initiates printing in the following way:

1. It first checks to see what command line options were specified. In some cases (for example, **-d**dest), **lp** acts immediately to test the validity of the option. In other cases, **lp** merely sets a switch at this stage. **lp** also makes sure it can access the file or files to be printed and that the files are not empty; if either of these conditions is not met, it sends a message to the user's terminal.

2. **lp** checks for a valid destination for the print job. If there is one, **lp** checks the acceptance status of that destination. If no destination is specified on the command line, **lp** checks to see if the user has assigned a value for the environment variable LPDEST; if the user has no personal default, **lp** looks for a system default destination. If **lp** cannot find a valid destination, or if the valid destination is not accepting requests, **lp** sends a message to the user's terminal.

3. If there is not already a file to contain the request, **lp** creates it. (This is the file **lpsched** reads to catch all queued requests and schedule them for printing.) **lp** formats a request, with any options **lpsched** or the interface program will act upon; it assigns the request a sequence number that will queue this request immediately behind the previously queued request. **lp** then opens the file and adds the request to it.

4. If the user specified the **-c** option, **lp** copies the file or files to be printed to the **/usr/spool/lp** directory.

5. **lp** closes the file containing the queued requests.

6. **lp** informs the user of the request ID and destination for the print job.

The **lpsched** program routes the queued request to the first available printer that matches the requested destination. **lpsched** oversees the completion of the print job, as follows:

1. **lpsched** is constantly reading several files to match queued requests to available printers. One of these files contains the requests; other files contain information about the classes to which printers belong, the devices to which printers are associated, and the status of specific printers (enabled or disabled, currently printing, and so forth).

   If **lpsched** can not find an enabled printer to match the destination specified in the request, it immediately sends a message to the user's terminal.

2. When **lpsched** has assigned an available printer to service a particular request, it forks two processes: one to execute the printing and another to wait until the first has died, inform **lpsched** to that effect, and deliver a return code.

   The first forked process takes the command options, request ID, and user name and formats these as positional parameters to pass to the interface. The process checks to see if the interface is executable, and tries to open the device for reading and writing. If it can open the device, it does so with the device opened as standard output and

with standard input redirected to **/dev/null**. (In this way, the interface itself is unaware of the specific device.) If it cannot open the device, it disables the printer associated with the device and sends a message to this effect to the user's terminal.

3. Once the device has been opened, the first forked process forks and execs a shell that executes the interface and directs output to the device. The interface prints the files on the standard output (the device). When the current shell dies, the waiting process informs **lpsched**, and **lpsched** updates the status of the printer.

4. **lpsched** makes an entry in the log file **/usr/spool/lp/log** when printing begins. This entry contains the logname of the user that made the request, the request ID, the name of the printer that the request is being printed on, and the date and time that printing first started. (If this request has been restarted, there will be more than one entry for it in the log file.)

5. If the user has requested a mail message on completion of the print job, **lpsched** sends the mail message.

lpsched's Routine Maintenance Work

In addition to routing queued requests, **lpsched** maintains and updates information resulting from the application of the **lpadmin** command; it receives and processes new device assignments and other modifications to the **lp** spooling system.

lpsched logs routine events, such as when it started and stopped, and any errors reported to it, to **/usr/spool/lp/log**. Each time **lpsched** is started, it moves **/usr/spool/lp/log** to **/usr/spool/lp/oldlog** (replacing the current **oldlog** with a new **oldlog**) and creates a new **/usr/spool/lp/log**.

Each time **lpsched** is started, it also creates a lockfile **/usr/spool/lp/SCHEDLOCK**; this prevents another **lpsched** from being started. **lpshut** kills **lpsched** and causes **SCHEDLOCK** to be removed. The initialization script **/etc/rc** removes **SCHEDLOCK**, if it exists, before starting **lpsched**.

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM

(Page 1 of 7)

In this example, you want to set up two hardwired devices for spooled printing, a DataSouth DS-180 printer operating in parallel mode and a Diablo 630 operating in serial mode. The Diablo will be the system default destination. There will not be any printer classes. You plan to use the DataSouth printer to print 132 column output (for example, program listings); the Diablo for letter quality reports.

You will name the DataSouth printer "ds" and the Diablo "diablo". The DataSouth will be connected to the parallel channel on the main CPU board, the Diablo to tty004.

1. First, become the superuser.

2. As superuser, give **lp** exclusive access to both devices, as follows:

```
# chown lp /dev/plp
# chmod 600 /dev/plp
# chown lp /dev/tty004
# chmod 600 /dev/tty004
```

3. Check **/etc/inittab** to make sure there is no **getty** running on tty004. Since the entry for tty004 is

```
004:2:off:/etc/getty tty004 9600
```

   you do not need to modify it.

4. Check the **/usr/spool/lp/model** directory for suitable interface programs; you decide to modify the file **dumb** twice, once for each printer.

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM

(Page 2 of 7)

This is the distributed **dumb** program:

```
    # lp interface for dumb line printer
    #
    #   SCCS @(#)dumb   1.1
(1) x="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    echo "\014\c"
    echo "$x\n$x\n$x\n$x\n"
(2) banner "$2"
    echo "\n"
    user=`grep "^$2:" /etc/passwd | line | cut -d: -f5`
(3) if [ -n "$user" ]
    then
        echo "User: $user\n"
    else
        echo "\n"
    fi
(4) echo "Request id: $1    Printer: `basename $0`\n"
    date
    echo "\n"
(5) if [ -n "$3" ]
    then
        banner $3
    fi
    copies=$4
    echo "\014\c"
    shift; shift; shift; shift; shift
    files="$*"
(6) i=1
    while [ $i -le $copies ]
    do
        for file in $files
        do
                cat "$file" 2>&1
                echo "\014\c"
        done
        i=`expr $i + 1`
    done
(7) echo "$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
    n\n\n\n\n\n\n"
    echo "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n$
    x\n$x\n$x\n$x"
    echo "$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
    n\n\n\n\n"
    echo "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\
    n"
    echo "$x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n"
(8) exit 0
```

620-013

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM
(Page 3 of 7)

① prints 4 11Ø-column rows of Xs at the top of the banner page

② prints a banner with the user logname

③ looks at **/etc/passwd** for the full user name; prints the full user name, if found

④ prints the request ID, printer name, and date

⑤ prints a banner with the title of the document, if specified on the **lp** command line

⑥ for number of copies, **cats** each file, merging standard error and standard output (prints any errors on hard copy)

⑦ prints 4 borders of rows of Xs at the bottom of the printout

⑧ indicates to **lpsched** that the print job was successful

Make copies of the **dumb** script for modification:

```
# cd /usr/spool/lp/model
# cp dumb ds18Ø
# cp dumb diablo63Ø
```

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM

(Page 4 of 7)

Neither printer requires much modification of the script. These are the changes you make:

- Put a comment at the top of each script to identify the interface. The comment is the only change you make to the ds180 script.

- Since you will be using the Diablo for letter-quality documents, you want to remove the commands in the script that place borders of X's on the banner page and on the last page of the document.

- Add an **stty** command to the beginning of the Diablo script:

```
stty 1200 oddp tab3 cread opost onlcr ixon ff1 cr2 nl0 <&1
```

  This gives you 1200 baud; enable parity (odd) with 7-bit character size; software tabs (necessary if there are no hardware tabs); XON/XOFF protocol (cread is required for software handshaking); and delays (requiring opost for post-process output) for form feeds, carriage return, and line feeds.

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM
(Page 5 of 7)

When you have finished, the Diablo script will look like this:

```
# lp interface for Diablo 630 printer
#
① stty 1200 oddp tab3 cread opost onlcr ixon ff1 cr2 nl0 <&1
echo "\n\n\n\n"
banner "$2"
echo "\n"
user=`grep "^$2:" /etc/passwd | line | cut -d: -f5`
if [ -n "$user" ]
then
    echo "User: $user\n"
else
    echo "\n"
fi
echo "Request id: $1    Printer: `basename $0`\n"
date
echo "\n"
if [ -n "$3" ]
then
    banner $3
fi
copies=$4
echo "\014\c"
shift; shift; shift; shift; shift
files="$*"
i=1
while [ $i -le $copies ]
do
    for file in $files
    do
        cat "$file" 2>&1
        echo "\014\c"
    done
    i=`expr $i + 1`
done
② echo "\014\c"
exit 0
```

620-014

① sets terminal options, merging standard input and standard output

② does a final form feed at the end of the print job

Give lp ownership of both scripts:

```
# chown lp diablo630
# chown lp ds180
```

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM

(Page 6 of 7)

5. Turn off the scheduler and run **lpadmin**:

```
# /usr/lib/lpshut
scheduler stopped
# /usr/lib/lpadmin -pds -v/dev/plp -mds180
# /usr/lib/lpadmin -pdiablo -v/dev/tty004   -mdiablo630
# /usr/lib/lpadmin -ddiablo
```

6. Allow both printers to be accepted by the **lp** spooling system:

```
# /usr/lib/accept ds diablo
destination "ds" now accepting requests
destination "diablo" now accepting
requests
```

7. Make sure the options are set on the printers so they correspond to the interface options, and check that printer cables are plugged into the correct channels. Then turn on the printers, make sure the paper is at top of form, and enable each printer:

```
# enable diablo
printer "diablo" now enabled
# enable ds
printer "ds" now enabled
```

Example 6-1

CONFIGURING THE lp SPOOLING SYSTEM
(Page 7 of 7)

8. Finally, restart the scheduler and check the status of the **lp** system:

```
# /usr/lib/lpsched
# lpstat -t
scheduler is running
system default destination: diablo
device for diablo: /dev/tty004
device for ds: /dev/plp
diablo accepting requests since Sep 26 17:54
ds accepting requests since Sep 26 17:54
printer diablo is idle.  enabled since Sep 26 17:56
printer ds is idle.  enabled since Sep 26 17:57
```

Whenever a new device requiring a driver is added to a MightyFrame, CTIX must be configured to load the device's driver. Drivers fall into two basic categories:

- Loadable drivers. These are drivers that can be loaded dynamically with each system boot according to an entry in the initialization script **/etc/drvload**. All supported CTIX drivers, except those for disk controllers, are loadable.

  In most cases, configuring loadable drivers is a simple matter of adding or uncommenting a line in **/etc/drvload**. Any loadable driver, except for the Ethernet driver, can be "hard-coded" into a CTIX kernel configuration, rather than being loaded dynamically; "Hard-coding a Loadable Driver into the Kernel," at the end of this chapter, documents how to do this.

- Nonloadable drivers. These are drivers that cannot be loaded dynamically and must be configured into the CTIX kernel. The only supported drivers in this category are disk controller drivers. SMD disks cannot be used on a Mighty-C computer system.

NOTE

The information in this chapter is subject to change. Consult all relevant Release Notices before you install a new driver.

To install a device driver you have written, refer to the Writing MightyFrame Device Drivers Manual.

LOADING LOADABLE CTIX DEVICE DRIVERS

The CTIX configuration file **/etc/drvload** defines the set of loadable drivers to be dynamically loaded when the system boots.

As distributed, this file contains entries for the following ten loadable drivers:

- Tuneable Variables driver
- Kernel Profiler driver
- Kernel Debugger driver
- Shell Layers (SXT) driver
- Parallel Printer driver
- RS-422 (cluster) driver
- Terminal Accelerator (IOP) driver
- Ethernet (ENP) driver
- Pseudo-Terminal (PTS) driver
- 1/2" Tape (XM) driver (not available for Mighty-C)

LOADING A DRIVER THAT ALREADY HAS AN ENTRY IN /etc/drvload

You do not need to change the entries in **/etc/drvload** to add or remove the RS-422 and IOP drivers, since CTIX determines if the hardware is attached, and loads the drivers when appropriate.

For the other eight drivers, use a text editor to add or remove a comment sign in front of each driver entry. Note that the Tuneable Variables driver must be in the first position in the file. Note also that if you are using Ethernet, you must uncomment the PTS driver entry. (See also Appendix B.)

ADDITIONAL STEPS FOR ETHERNET

These additional steps are required to configure CTIX for Ethernet:

1. Modify **/etc/rc** to set the UUCP node name and to uncomment the line containing the **enpstart** command.

> **NOTE**
>
> Steps 2, 3, and 4 do not apply to a Mighty-C computer system, since the Mighty-C does not have a VMEbus.

2. Modify **/etc/system**(4) to specify slot number(s) of the CMC Ethernet board(s), board type, starting address, and address length.

   - Ethernet board type is **1**.

   - The first Ethernet board should have a starting address of **C0DE0000**. Address length is specified as **131072** (this is 20000 hex); thus, the second Ethernet board starts at address **C0E00000**, the third board at **C0E20000**, and so forth.

   For example, to configure **/etc/system** for two Ethernet boards, one in slot 0 and the other in slot 1, add the following lines to the !VMESLOTS section of the file:

```
0   1   C0DE0000      131072
1   1   C0E00000      131072
```

   (See the **/etc/system** description in Appendix B.)

   Note that regardless of actual slot number or address, the first Ethernet board in the **/etc/system** file (that is, the Ethernet board with the lowest slot number relative to other Ethernet board slot numbers) corresponds to **en0** (Ethernet interface unit 0), the second to **en1**, and so forth: Ethernet interface unit numbers are position dependent in this file.

3. Run **ldeeprom** so that the **enp** driver knows about the boards and their addresses and can correctly initialize these addresses at system start-up.

(**ldeeprom** reads **/etc/system** and outputs information about boards, slot numbers, and addresses to the EEPROM. The drivers are then able to obtain this information from the EEPROM when they are loaded.) The command is

/etc/lddrv/ldeeprom

4. If you are installing more than one Ethernet board, change the jumpers on boards that are additional to the first board so they set to the correct addresses. (All CMC Ethernet boards have their jumpers set at the default address CØDEØØØØ.) Refer to the MightyFrame VME Ethernet Card Manual.

ADDING LOADABLE DRIVER ENTRIES TO /etc/drvload

The following drivers can also be loaded dynamically:

- sna
- bsc

Note that all necessary entries for these drivers are in **/etc/master**. The procedure that installs **sna** or **bsc** software places the relevant **.o** files in the directory **/etc/lddrv**. (Refer to the appropriate Release Notice.) To load the driver dynamically at system boot, add the following line to **/etc/drvload**:

./lddrv -av handlername

where

handlername is **sna** or **bsc**.

NOTE

Since a Mighty-C computer system does not use SMDs, the information in the following paragraphs does not pertain to the Mighty-C configuration.

The following procedure describes how to add an SMD or ST506 controller that does not control the boot drive. To make an SMD a boot device, use the "raw install" tape and follow the instructions provided in the Release Notice.

To add an SMD disk controller, perform the following steps:

1. Modify the **/etc/system** file to specify the slot number(s) of the Interphase SMD controller board(s), board type, starting address, and address length. (If you have used the "raw install" tape to configure an SMD as a boot device, the controller for that device also has the initialization routine name **initVs32** specified in the last field.)

   - Interphase SMD controller board type is **2**.

   - Available addresses are in the range C1000000 to C100FFFF. Choose an address (say, C1000200) for the first board and a non-overlapping address for each successive board.

Address length is specified as 512 bytes (this is 200 hex). Thus, if one disk controller board is at C1000200, another board can be at C1000400, a third board at C1000600, and so forth.

Entries in **/etc/system** are in slot order, and slot order corresponds to disk controller order (for example, c1, c2, c3). Addresses, however, are not position dependent in the file (that is, c1 can have a higher address than c2).

For example, if c0 is the controller for the onboard ST506 disks, and c1 and c2 are SMD controllers in VME slots 2 and 3 respectively, the following entries in **/etc/system** are valid for the SMD controllers:

```
2    2    C1000400      512
3    2    C1000200      512
```

If there are no ST506 disks, and c0 and c1 are SMD controllers in VME slots 2 and 3 respectively, the following entries in **/etc/system** are valid for the SMD controllers:

```
2    2    C1000400    512    initVs32
3    2    C1000200    512
```

Note that only c0 can control a boot device, since the system expects the loader and **/unix** to be on the first drive controlled by c0.

2. Run the program **ldeeprom** so that the driver knows about the boards and their addresses and can correctly initialize these addresses at system start-up. (You need to run **ldeeprom** each time you change VMESLOTS information in **/etc/system**; if you add Ethernet entries at the same time you add SMD entries, you need to run **ldeeprom** only once.) The command is

   /etc/lddrv/ldeeprom

3. To reconfigure the CTIX kernel to add a controller for a nonbootable drive, follow this procedure below (refer to Chapter 8 for a discussion of disk naming conventions and **config** (1M) for a description of **dfile**):

   - **cd** to **/usr/sys/cf**. If you are currently booting from an ST506, make a copy of **dfile** (call it, say, **dfileaddsmd**) and modify the **dfile** copy to add entries for the SMD controllers. If you are currently booting from an SMD, and want to add an ST506, make a copy of **dfileVME** and modify the **dfileVME** copy to add an entry for the ST506.

   - The first part of **dfile** contains entries for disk and other devices. The disk devices are specified in controller order. An entry for an SMD controller is a line of the form

     Vsmd3200

The first entry in the file as distributed is **diskonbd**. Since controller entries are positional in the file (cØ is the first entry, cl the second entry, and so forth), this entry specifies that cØ is the on-board ST5Ø6 controller. To add entries for SMD controllers, place the entry or entries above or below the **diskonbd** entry, depending on which device will have what controller number.

For example, the following entries specify that the ST5Ø6 controller is cØ, and cl and c2 are SMD controllers:

```
diskonbd
Vsmd32ØØ
Vsmd32ØØ
```

Since, in this case, you are not changing the specification for **root** and **swap**, you do not need to alter any other parts of the file.

4. Rebuild the CTIX kernel. Enter

make VER=yourversion **DFILE=**dfilename

where

yourversion — is the CTIX version name you are creating; for example, **5.1Øsmd**.

dfilename — is the filename of the **dfile** copy you have modified.

The command

make VER=5.1Øsmd DFILE=dfileaddsmd

creates a file called **CTIX5.1Øsmd** in the directory **/usr/sys**. Always make a different version number from the standard CTIX release.

5. Bring the system to single-user mode and remount **/usr** (see Chapter 8).

6. Move **/unix** to **/OLDunix** so you have a backup of the current kernel:

   mv /unix /OLDunix

7. Make the new CTIX kernel the boot default by entering

   cp /usr/sys/CTIXyourversion **/**

   ln /CTIXyourversion **/unix**

   For example, if yourversion is **5.1Øsmd**, the command is

   cp /usr/sys/CTIX5.1Øsmd /

   ln /CTIX5.1Øsmd /unix

8. Sync and reboot the system. (Before you type the **reboot** command, always wait for the disks to become quiescent.) The command is

   sync
   sync
   sync
   reboot

9. If the system does not reboot properly, or if any other catastrophic event (like a system crash) occurred after you moved **/unix**, boot the system with the maintenance tape, run **fsck**, move **/OLDunix** to **/unix**, figure out what went wrong if it is not obvious, and start over.

Chapter 13, "System Maintenance and Troubleshooting," describes the procedure for restoring an SMD-only system if the EEPROM data has become corrupt.

HARD-CODING A LOADABLE DRIVER INTO THE KERNEL

Any loadable driver, except the Ethernet driver, can be hard-coded into the CTIX kernel, rather than being loaded dynamically. There is one circumstance in which this is absolutely necessary: when you want RS-422 terminals to be active in administrator mode.

If CTIX encounters serious file system problems during a system boot, it goes to **init** state 6 and waits for a root login to bring it to single-user mode. If the RS-422 cluster driver is configured to load dynamically and is not hard-coded, it will not be loaded when the system enters state 6. Therefore, it will be impossible to use an RS-422 terminal to bring the system to single-user mode.

A special **dfile** with cluster driver entries, **dfilecl**, is provided to help you hard-code the cluster driver in a relatively painless fashion. **dfilecl** has four entries, right after the disk controller entries, that must be included in a **dfile** that hard-codes the cluster driver. These entries are

```
cluster
tsp
tsy
tst
```

If you have not reconfigured the CTIX kernel since you installed the operating system, use **dfilecl** to remake the kernel.

If you are not sure if the kernel has been reconfigured, check the **/usr/sys** directory for clues! If, for example, you find a program file named **CTIX5.1Øsmd**, look for a **dfile** that may have generated the new **unix**.

If you are using SMDs, locate the **dfile** that generated the current **unix** and add the four entries below the controller entries.

To hard-code a cluster driver, or any other loadable driver except the Ethernet driver,

1. **cd** to **/usr/sys/cf**. If you are hard-coding the cluster driver, either use **dfilecl** or add the four cluster entries to the **dfile** that generated the current **unix**.

If it is a different driver from the cluster driver, check **/etc/master** for the driver name to use in the **dfile** entry. (This is the same driver name that you would find in or add to **/etc/drvload**.) Copy the **dfile** that generated the current **unix** and add the appropriate entry or entries.

2. Rebuild the CTIX kernel as described earlier in this chapter.

NOTE

Once you have hard-coded a loadable driver, it is not necessary to comment out the driver entry in **/etc/drvload**. When the system boots, you will get a warning message, which you can ignore.

8 DISKS

This chapter describes the following:

- use of MightyFrame disks
- how disks are organized
- how you, as system administrator, make the disk's storage capacity available to users

MIGHTYFRAME DISKS: INTRODUCTION

There are two basic kinds of disks for a Mighty-Frame computer: 5 1/4-inch "on-board" disks (ST5Ø6), and SMD disks. SMD disks are 8-inch "rack mount" (X-Box), 19-inch "rack mount," and very large standalone "wash tub" disks.

NOTE

A Mighty-C computer system uses ST5Ø6 disks only, and not SMD disks. Therefore, all the information in this chapter that relates to SMDs only is irrelevant to Mighty-C configuration.

On-board disks and SMD disks are organized similarly; except where noted, the information in this chapter applies to all kinds of MightyFrame disks.

A MightyFrame computer (that is not a Mighty-C) can have on-board disks only, SMD disks only, or a combination of on-board disks and SMD disks. SMD drives can be standalone, or they can be contained in MightyFrame enclosures. The three types of MightyFrame enclosures and the disks they hold are

- A-Box (base enclosure). Holds on-board disks only, with a maximum of three drives.

- V-Box (VME expansion enclosure). Holds VME cards and up to two SMD drives.

- X-Box (SMD expansion enclosure). Holds up to four SMD drives.

The maximum number of on-board disks on a Mighty-Frame system is three; the maximum number of SMDs is eight. There is one disk controller for the three on-board disk drives and one disk controller for each pair of SMD drives.

Possible hardware configurations are listed in Table 8-1.

Table 8-1

MIGHTYFRAME (NOT MIGHTY-C) ENCLOSURES AND TOTAL DISK CAPACITY*

Enclosures	On-board	Total SMDs
A-Box only	1-3	Ø
A-Box and V box	Ø-3	Ø-2
A-Box, V-Box, and X-Box	Ø-3	1-6
A-Box, V-Box, and 2 X-Boxes	Ø-3	2-8
A-Box and 2 B-boxes	Ø-3	2-8

*SMDs can also be standalone. There is a maximum total of 8 SMDS, enclosed and standalone.

The procedure that installs CTIX initializes and installs the following in the first ST5Ø6 or SMD disk:

- the reserved area (slice Ø)
- the root file system, excluding **/usr** (slice 1)
- a swap slice (slice 2)
- **/usr** file system (slice 3)
- one or more additional slices (slices 4, 5, 6, and so forth) if the disk is at least 85 Mbytes.

NOTE

During the installation of operating system software, if the system detects the presence of both an ST506 controller and an SMD controller, it prompts you to specify which type of disk you want to boot from. It then initializes the first disk of the type you specified and installs the root and additional partitions in that disk.

All MightyFrame disks are initialized during the manufacturing process. If a disk is not the boot device, it is divided into two slices:

- the reserved area (slice 0)
- a second slice (slice 1)

You should never need to completely reinitialize a MightyFrame disk, except in one unusual circumstance. (See "Initializing and Configuring Disks" later in this chapter.) You will, however, probably want to repartition (adjust slice boundaries on) one or more disks, using the **iv** command.

Note that the software installation procedure creates a kernel configuration such that you can use three on-board disks. To allow CTIX to use additional drives and controllers, you must

- modify **/etc/system** and run **ldeeprom**
- add controllers to the kernel configuration file and remake the kernel

Adding drives and controllers is described in Chapter 7, "CTIX Device Drivers."

This chapter discusses use of the CTIX **iv**(1), **swap**(1M), **mkfs**(1M), **labelit**(1M), and **fsck**(1M) commands. For detailed information on these commands, see your CTIX operating system manual.

NOTE

There are two administrative tools for performing disk administration: one tool allows you to examine the current configuration of a disk; the other allows you to make a file system in a previously-configured disk. See the CTIX Administration Tools Manual for a description of the tools.

DISK ORGANIZATION

The paragraphs below explain how disks are organized. A few concepts are introduced here that describe how the disk is divided into manageable units and what purposes the units serve.

A disk has at least 1, and at most, 16 slices. A slice is simply a section of the disk that is used as a unit. Slices are numbered in hexadecimal Ø to F. Slice Ø, also called the Reserved Area, is preassigned to hold the data required to manage the disk; there is normally at least one additional slice.

More than one slice can occupy a particular area of a disk. Overlapping slices make convenient units for certain backup procedures.

If a disk can boot CTIX, slice 1 contains the root file system, as described below. You must also provide a swap slice. CTIX uses slice 2 on the bootable disk as swap space unless you configure the kernel to look elsewhere. The **swap** command allows you to specify other slices or parts of slices as swap space in addition to the main swap device (see "Adding Swap Space with the **swap** Command" below).

NAMING CONVENTIONS

CTIX slices are accessed as either block devices or character devices. (See Appendix A, "File System Concepts" for a discussion of special files and block and character devices.) The name of a disk's special file that is a CTIX slice takes one of two forms:

/dev/dsk/c#**d**#**s**#

or

/dev/rdsk/c#**d**#**s**#

where

dsk specifies that you are accessing the slice as a block special file.

rdsk specifies that you are accessing the slice as a character special file ("raw" device).

c# is the disk controller: Ø for the first controller, 1 for the second controller, and so forth. The first controller corresponds to the first entry in the kernel configuration file, the second to the second entry, and so on. (See Chapter 7, "CTIX Device Drivers.")

d# is the disk: Ø for the disk in the first disk drive position, 1 for the disk in the second disk drive position, and so forth.

s# is the slice: Ø for the first (reserved) slice, 1 for the second slice, and so forth. Numbering is hexadecimal (A through F for slices 1Ø through 15).

For example, the name of the block special file for the second slice on the first disk of controller Ø is **/dev/dsk/cØdØs1**.

By convention and by default, the slice named **/dev/dsk/cØdØs1** or **/dev/rdsk/cØdØs1** contains the root file system. The assignment of **cØdØ** to the disk that contains the root file system is arbitrary and is a function of the kernel configuration that is installed when you load the operating system. You can change the kernel configuration, for example, to make CTIX boot from a drive that is not **cØdØ**. (See Chapter 7 for information on changing the configuration of the CTIX kernel.)

NOTE

This chapter assumes the default configuration in which the disk that boots CTIX is disk Ø of controller Ø. If you change the kernel configuration so that **cØdØs1** does correspond to the root file system, make the device name changes in this chapter.

Since there are two SMD drives per controller, if the first SMD is **c1dØ**, the third SMD will probably be **c2dØ**, and the fourth **c2d1**.

FILE SYSTEMS

An important use of a slice is to contain a file system. A file system consists of CTIX files plus the data structures the CTIX kernel requires to support file uses. Normally, all the slices on a disk, except for Ø (the Reserved Area) and any swap areas, contain file systems.

NOTE

In the case of overlapping slices, only one of the slices occupying an area of the disk can contain a file system.

To use a file system, it must be mounted to give it a place in the CTIX directory hierarchy. The root file system in slice 1 is permanently mounted; other file systems are mounted by the **mount** command or by an automatic procedure (using **mount**) that is executed when the system is booted.

Users use the mounted file systems as a single hierarchy of directories and files. For the most part, the division of CTIX's file hierarchy into slices is not apparent to users and disk organization is not an ordinary user's problem.

CAUTION

Be careful not to destroy the root file system. Doing so renders the system unusable.

SWAP SLICES

The total size of all swap areas, together with the size of physical memory, determines the limit on virtual program size.

By default, slice 2 on the disk from which CTIX was booted is the initial swap slice: there must be one such device available to CTIX.

CTIX also uses add-on swap slices to increase the amount of available swap space. The **swap** command allows you to add swap area without repartitioning a disk. Putting one or more invocations of **swap** in the initialization script **/etc/drvload** is the normal way to make add-on swap space available to CTIX.

NOTE

While it is possible to repartition the disk containing the initial swap slice so that slice 2 is smaller than it was when you installed CTIX, it is not recommended that you do so. Use the **swap** command to increase the total swap area over the initial slice size, rather than to replace any part of it.

Experience will tell you how much swap space you need, but here are some general guidelines:

- Total swap space should amount to the larger of the following two measurements: (1) between 1 1/2 and 2 times physical memory, or (2) 4 times the virtual size of the largest process.
- 4 Mbytes of total swap space is the minimum under any circumstances.

A sign that swap space is too small is the failure of a large number of commands with messages like "not enough space." If there are indications that swap space is inadequate, examine the console file **/etc/log/confile** for the message "running low on swap." (See "Adding Swap Space with the **swap** Command," later in this chapter.)

Do not write a file system or any other data into either the initial swap slice or add-on swap slices.

OTHER SLICES

A disk can see uses other than holding file systems or swap slices. Some data base systems manage slices directly, without using a file system.

INITIALIZING AND CONFIGURING DISKS

The paragraphs below explain how to use the **iv** command to perform the following:

- Completely initialize (reformat) a disk and divide it into slices. Normally you would use this procedure in one circumstance only--to change the error correction mode from ECC to CRC or vice versa.
- Add a download area, loader, or dump area.
- Adjust the slice sizes on a disk already in use.

If you use any of these procedures on a disk already in use (for example, to adjust the slice sizes), assume that all data on the disk will be lost. Do a complete backup before adjusting slice sizes or otherwise changing the disk configuration. (See Chapter 12, "Backups and Restores.")

Perform the steps below to initialize and configure a disk:

1. Examine a disk description file for your disk. Use it to determine the dimensions of the disk.

2. Plan the disk's division into slices and determine how the slices are to be used.

3. Modify the disk description file to describe the configuration you want to obtain.

4. Test the new file for errors.

5. Run **iv**, the CTIX disk initialization utility.

Use this procedure both to initialize a disk and to specify new slice boundaries on a disk already in use.

CAUTION

Before applying this procedure to any disk containing data you do not want to lose, copy or back up the disk. It is possible to rearrange slice boundaries without damaging all existing slices, but do not count on this. Reformatting (completely reinitializing) a disk destroys all data on the disk.

If you adjust the size of the initial swap slice, reboot the MightyFrame system as soon as you finish reconfiguring the disk.

Example 8-1, at the end of this chapter, illustrates how to configure three MightyFrame on-board disks.

EXAMINING THE DISK DESCRIPTION FILE

The prototype disk description files in **/usr/lib/iv** describe the various disks commonly used with MightyFrame. If the disk is not presently configured in any usable way (for example, a disk whose configuration was accidentally undone), use **cat** or some other text file utility to examine the file for your disk.

cat /usr/lib/iv/_file_

where

file is the prototype description file name.

If the disk is already configured and you want to adjust the slice boundaries, create a disk description file that describes the _current_ configuration of the disk and use **cat** or some other text file utility to examine the file.

While it is not absolutely necessary that the disk description files you create be kept in a directory separate from the prototype files, you will find it useful to do so. To create and examine the description file, type:

/etc/iv -d /dev/rdsk/c_c_**#d**_d_**#sØ > **_file_
cat _file_

where

file is the name of the disk description file you create. Example: You have created a directory named **/usr/lib/iv.work** for your description files and the present file describes the first disk; you name your file **desc.winch1**. Then *file* is **/usr/lib/iv.work/desc.winch1**.

The disk description file is a text file that gives the physical characteristics of the disk. If it is a prototype file, it gives a prototype division of the disk into slices. If it is a file created by **iv**, it gives a description of the current state of the disk, including a list of bad blocks and the actual division of the disk. It is a good idea to make and keep a printed copy of the description file for each disk so that you have a record of bad block information.

Determining Disk Dimensions

Of interest at the beginning of the disk description file is information concerning *heads*, *cylinders*, and *sectors*. *Sectors* is an odd number, since the disk has a sector on each track reserved as a bad block alternate; subtract one from *sectors* before using it. Calculate the total number of 1024-byte logical blocks the disk can contain, as follows:

heads **x** *cylinders* **x** (*sectors* / 2) = disk size in logical blocks

Also calculate the number of logical blocks per track (call this value *bpt*):

sectors / 2 = *bpt*

Some commands described below require values expressed in numbers of logical blocks. Others, however, require values expressed in physical sectors. One logical block equals two physical sectors. Since a physical block is equivalent to a physical sector, one logical block also equals two physical blocks.

Notice how the disk is divided in the description file. Below is an example of a disk description file created by **iv** for an Hitachi disk.

```
type            HD
name            52HIT
cylinders       645
heads           7
sectors         17
steprate        14
hitech
precomp         64
ecc
$
badblocktable   1
loader          /usr/lib/iv/loader
$
1048
2441
2855
5064
8975
15119
15677
15678
18204
20925
23533
23534
42091
43918
52366
60466
60467
62419
70500
73439
74153
$
0
32
12032
18032 $
$
$
```

Slice information in the disk description file. The beginning of each slice is expressed as a logical block number; a slice boundary must also be a track boundary. The disk described by this file is a system disk that has four slices: the reserved area begins at block 0 (as it always does), the root file system begins at block 32, the swap area begins at block 12032, and the fourth slice (containing the **/usr** file system) begins at block 18032.

The total number of logical blocks on this disk is 7 x 645 x ((17-1) / 2) = 36120. The number of blocks per track is ((17-1) / 2) = 8, which means that all slice boundaries (after 0) must be divisible by 8. The disk has an initial swap size of 18032 - 12032 = 6000 logical blocks, which is approximately 6 Mbytes. The disk's fourth slice is 36120 - 18032 = 18088 logical blocks, which is approximately 18 Mbytes.

There are no overlapping slices described in this file. (For information on how to specify overlapping slice, see "Changing Slice Boundaries" later in this chapter.)

Understanding the Rest of the Description File

iv(1) is a complete reference to the contents of a description file. A description file consists of five parts, separated by a line containing a "$" only. The five parts are

- general volume description
- reserved area files description
- bad blocks description
- partition table description
- down load area description

Note the following with respect to the parts of the description file:

- The "name" field in the general volume description is a good place to enter the serial number of the disk for your future reference.

- The value for "steprate" (the speed at which the disk controller pulses to the disk drive) is **14** for all on-board (ST506) disks. (Do not specify a step-rate value for SMDs.)

- **ecc** describes the mode of error correction for this disk. If **ecc** is not specified, the mode of error correction is **crc**. To change the mode, delete the **ecc** line from the file and completely reformat (**iv -i**) the disk; all data on the disk is lost in this case. SMD drives are always run in **ecc** mode.

- In the case of SMDs, you must specify a value for **gap1** and **gap2**. Gap size values can be obtained from the disk manufacturer's documentation. See also the disk description for your SMD drive in **/usr/lib/iv**. Correct values for gap sizes are given in these prototype files.

  Note that the **formatextra** flag is required for some SMDs. (See the **iv**(1) manual page.)

- Part 2, the Reserved Area description, of the file shown above describes a disk that boots CTIX, and is very different from a disk that does not boot CTIX. If a disk boots CTIX, it must have a line specifying a value for the loader; if

there are download images, they must be specified in the bootable disk's Reserved Area description also. For both non-bootable and bootable disks, the "badblocktable" entry is mandatory.

If a disk does not boot CTIX, its reserved area is normally only 1 track (8 logical blocks if it is an on-board disk), and "badblocktable" is normally its only entry.

- The value for "badblocktable" is the number of logical blocks to allocate for bad block information: **1** for disks under 130-Mbytes, **2** for disks between 130- and 260-Mbytes, **3** for 260-Mbyte disks, **5** for 635-Mbyte disks or greater.

- You can specify a "dump" area (expressed as the size in logical blocks) to contain a dump after a system crash. The description file given above does not specify a dump area. A dump area is not required, and if you have a dump area, it need not be on the bootable disk. If you do specify a dump area, it must be large enough to hold all crash information. A rule of thumb is to make the dump area the size of physical memory. Do not specify more than one dump area for a MightyFrame system.

- The loader requires 20 logical blocks. The value for "loader" is the full pathname of the file in **a.out** format to put in the loader. A second, optional parameter is the size of the loader area. Do not specify more than one loader.

- The value for "downloadarea" is the size, in logical blocks, for the area to contain system images for downloading (for example, PT and GT RS-422 download images). A download area is only required when you are using terminals in RS-422 mode. (Download area is not specified in the disk description file given above.) The value for "downloadarea" is normally **300**. Note that the system image files themselves are specified in the last part of the description file. Do not specify more than one download area for a MightyFrame system.

- Slice 0 must be large enough to contain all of the reserved area components: add them up so you know where to begin slice 1. For example, if you have an on-board disk (8 blocks per track) and you are allocating 1 block for badblock information, 5 x 1024 = 5120 blocks for dump area, and 300 blocks for download area, calculate the size of slice 0 as follows: 1 (VHB) + 1 (badblock) + 20 (loader) + 300 (download) + 5120 (dump) = 5442 logical blocks + 6 blocks so the partition ends on a track boundary (number of logical blocks is divisible by 8) = 5448. Slice 1, therefore, should begin at block 5448.

  In the case of an on-board disk that is not a bootable disk and does not contain a dump area, slice 1 should begin at block 8.

- Bad blocks are listed in the third part of the disk description and are expressed as physical block numbers. The disk described in this file has 21 bad blocks.

Planning the Disk

In planning the size of the disk's slices, consider the applications or users who are likely to use the disk. Determine how many 1024-byte logical blocks each application or user is likely to require. If you have a lot of users who will create a lot of small CTIX files, they can share a file system, provided none of them is careless about using up extra space. Estimate their total requirements.

Try to put user files in a file system separate from the root file system and other file systems that hold system utilities.

Slices always contain a whole number of tracks. Round off your estimates to the nearest multiple of bpt.

The first and most important slice is the reserved area. A reserved area that does not support a system boot needs one track; a reserved area that does support a system boot needs at least 32 blocks to contain the Volume Home Block, bad block table, and loader.

CREATING A DISK DESCRIPTION FILE TO USE WITH iv

If you plan to configure the disk with different slice boundaries or other parameters from those in the description file, you must create a disk description file that contains the proper slice information.

If you are working with a prototype description file, create a new disk description file in the following way:

1. Copy the appropriate prototype description file from **/usr/lib/iv**.

2. Use a text editor to change the disk name, slice information, and any other information in the copy.

It is not absolutely necessary that the disk description file have any particular name or be in any particular directory. However, you will find it useful to keep all the disk description files you have created in a special directory, separate from **/usr/lib/iv**, and to name them in a way that indicates their purposes.

If you are working with a description file created by **iv**, use a text editor to change the disk name, slice information, and any other information in the actual description file.

Changing the Disk Name

The disk name goes on an existing line. The line is of the form

name ▶ diskname

where

name identifies the name line.

▶ is a tab character, generated by pressing the **Tab** key or **Control-I**.

diskname is the disk name (the serial number of the disk is recommended for the disk name). Only the first six characters are used. If the specified name is less than six characters long, the actual name is padded out to six with blanks.

Changing Slice Boundaries

When you have determined your slice requirements, use a text editor to modify the slice information in the description file. Make sure to begin new slices at a track boundary, and specify the correct number of blocks for the reserved area.

Slices must be in ascending order, unless they overlap.

NOTE

Overlapping slices are useful for doing backups with the **dd** program. (See Chapter 12, "Backups and Restores.") Overlapping slices are, however, prone to mishap, and must be very carefully specified in the disk description file and elsewhere.

Below are the rules for specifying overlapping slices in the description file:

- All slices on a disk extend either to the starting block of another slice, or to the end of the disk.

- If a slice extends to the end of the disk, a dollar sign ($) must follow the starting block on the same line that the starting block is specified. (The last slice in the partition table is assumed to extend to the end of the disk; in this case, the "$" is optional.)

- Therefore, if the starting block for a particular slice (say, slice 3) is less than or equal to the starting block of the previous slice (slice 2), slice 2 extends to the end of the disk and has a "$" following the block number.

The following partition table description is appropriate for a nonbootable 40-Mbyte disk. It specifies a reserved area, three slices (say, for file systems), and one slice that overlaps the three smaller slices:

```
0
8 $
8
18000
29064 $
```

TESTING A NEW DESCRIPTION FILE

If you have modified a disk description file, check it for errors before you use it to reformat or repartition a disk, as follows:

/etc/iv -o outputfile descfile

where

outputfile — is the file to output a report on descfile, normally the current description of the disk. When you are simply checking for errors, outputfile can be **/dev/null**.

descfile — is the name of the disk description file you are checking.

RUNNING iv

The **iv** program formats a disk and divides it into slices, or simply rearranges slice boundaries. This program is used to completely reinitialize disks (this is an exceptional case) or to rearrange the slice boundaries on disks.

CAUTION

Always bring the system to single-user mode before you run **iv**.

The **iv** command takes the one of the following forms:

/etc/iv -u /dev/rdsk/cc#dd#sØ file

/etc/iv -i /dev/rdsk/cc#dd#sØ file

where

file — is the name of a disk description file.

The first form suppresses formatting, so that slices whose boundaries do not change should not be damaged. Use this form of the **iv** command in almost all cases. However, it is not safe to assume that any data will survive application of **iv** to a disk. Always copy the entire contents of a disk before using **iv** on it.

CAUTION

The **iv -i** option reinitializes the disk and destroys all existing data on that disk. Use it only when you want to change the error-correction mode of a disk or when you want to completely overwrite everything on a disk. Use the **-u** option in almost all circumstances and especially to reconfigure disks in use.

If you adjust the size of the initial swap slice, reboot the system after you run **iv**.

ADDING SWAP SPACE WITH THE swap COMMAND

To add swap space, make an entry for each add-on slice in **/etc/drvload**. Each entry takes the following form:

swap -a /dev/dsk/cc#dd#ss# [swaplow] [swaplen]

where

/dev/dsk/cc#dd#ss# is the add-on slice.

[swaplow] is where the swap area begins in the slice. Value is the offset in logical blocks from the beginning of the slice. If omitted, the swap area begins at the first block in the slice.

[swaplen] is the length of the swap area in logical blocks. If omitted, the swap area extends to the end of the slice.

For example, to specify all of a 4-Mbyte slice **/dev/dsk/cØd1s3** as add-on swap space, the command is

swap -a /dev/dsk/cØd1s3

The **swap** command also deletes swap space. To delete the 4-Mbyte slice you added in the previous example, the command is

swap -d /dev/dsk/cØd1s3

To obtain information on the status of all swap areas, type

swap -l

MAKING AND USING A FILE SYSTEM

Each file system requires that you perform the following steps:

1. Create the file system with **mkfs**.

2. Mount the file system and arrange for its automatic mounting by adding it to the file **/etc/mountable**.

3. Create a **lost+found** directory for the file system.

4. Add the file system to the CTIX file **/etc/checklist**.

CAUTION

In the case of overlapping slices, exercise extreme caution when you create a file system. If you mistakenly attempt to create a file system in a slice that overlaps a slice with an existing file system, you will destroy data in the existing file system.

CREATING THE FILE SYSTEM

mkfs creates a file system by writing file system structures into a slice. **labelit** puts a label on the slice. To create a file system and label it, type

/etc/mkfs /dev/rdsk/cc#dd#ss#
/etc/labelit /dev/rdsk/cc#dd#ss# fsname vname

where

fsname — is the local name of the directory on which the file system is normally mounted. Examples: A file system normally mounted on **/a** is **a**; a file system normally mounted on **/usr/src** is **src**.

vname — is your name for the disk that holds the file system. Examples: **winch0**, for the first on-board disk; **SMD0**, for the first SMD disk, and so forth.

To find out the current fsname and vname of a previously labeled file system, type

/etc/labelit /dev/rdsk/cc#dd#ss#

Do not try to label a file system that is mounted.

MOUNTING A FILE SYSTEM

Every file system must be mounted to be used; this gives the file system a place in the CTIX file hierarchy. To mount a file system, use the **mount** command as follows:

/etc/mount /dev/dsk/cc#dd#ss# dir [-r]

where

dir — is the name of an empty directory. Subsequent references to dir will actually be to the root directory of the newly mounted file system; this gives users normal access to any files on that file system.

This directory can be on the root file system or it can be on another mounted file system. But the directory's file system must be already mounted.

[-r] — is the optional parameter that, when specified, controls access to the file system. If the -r flag is specified in the command line, the file system is mounted read-only. Files in the file system can be read, subject to normal permission rules, but cannot be modified, created, or deleted by any user. If r is missing, the file system is mounted read/write. All files in the file system can be read, modified, created, or deleted, subject to normal permission rules.

Without any argument,

/etc/mount

displays a list of currently mounted file systems.

A **umount** undoes a **mount**:

/etc/umount /dev/dsk/cc#dd#ss#

Note that **umount** will fail with a "busy" message if a file on the file system is open or is a working directory. The **fuser** command identifies processes using files in a particular file system and is useful in tracing open files that cause **umount** to fail.

The root file system (the file system in slice 1 of the disk from which CTIX was booted) is always accessible without a mount. Do not apply **umount** or **mount** to this file system.

Mounting and unmounting is automatic in the following circumstances:

- CTIX normally mounts file systems automatically when the operating system goes to multiuser mode. To accomplish this, you must add commands to the **/etc/mountable** file (see "Arranging for Automatic Mounting" immediately below).

- The **shutdown** program, which takes the system to single-user mode, unmounts all file systems except the root file system. See Chapter 2, "CTIX Modes and Superuser Status."

Arranging for Automatic Mounting

All file systems can and should be mounted automatically when the system is booted. To insure this, put the correct **mount** commands in **/etc/mountable**. For example,

```
/etc/mount /dev/dsk/c0d0s4 /usr
/etc/mount /dev/dsk/c0d1s1 /u
/etc/mount /dev/dsk/c0d1s2 /u/src
```

Note the ordering in the above example: Do not try to mount a file system on a file system that is not yet mounted.

CREATING THE lost+found DIRECTORY

Each file system must have a special directory for use by **fsck**, the file system maintenance program.

To create this directory, perform the following steps:

1. Make sure that the file system is mounted.

2. Make the file system's root directory your working directory.

3. Run the program

   /etc/mklost+found

EDITING THE CHECKLIST

The integrity of each file system should be checked whenever the MightyFrame computer is booted. To do this, use a text editor to modify the file **/etc/checklist**. Each line specifies a file system's character special file. The root file system's character special file is specified as the first entry, with additional entries (one for each file system) below it:

/dev/rdsk/c0d0s1

/dev/rdsk/cc#dd#ss#

Make sure that there are no duplicate entries. It is also important that the root file system be listed first.

CHECKING FILE SYSTEM INTEGRITY

If CTIX stops running for any reason, it is time to check the integrity of whatever file systems were mounted when the operating system went down. This is especially true after unplanned down times (such as operating system crash, power outage, or accidental reset of the system), but is also mandatory for planned down times. The **fsck** program examines the file system and repairs damage to file system structures.

> **NOTE**
>
> Make sure the file system you are checking, unless it is the root file system, is unmounted before you run **fsck**. Do this by bringing the system to single-user mode.

ROUTINE CHECKS OF FILE SYSTEMS

CTIX automatically checks file systems whenever it is booted. **fsck** is run with an option that performs most repairs automatically. If file system problems are too difficult for **fsck** to handle automatically, CTIX goes to administrator mode. If this happens, log in as root on the most convenient terminal and run **fsck** manually, as described below.

RUNNING fsck MANUALLY

To manually check a file system:

1. Bring the system to single-user mode and do not remount the file system you want to check.

2. Run **fsck** without the **-p** option. If you are checking the root file system, run **fsck** with the **-b** option.

Do not try to unmount a root file system.

The following command is required to check the root file system:

/etc/fsck -b /dev/rdsk/cØdØs1

To check a file system other than the root file system, type

/etc/fsck /dev/rdsk/cc#dd#ss#

This form of the **fsck** command requires your permission before each action.

Repair the normal root file system first. This simplifies work on the other file systems.

If you cannot understand **fsck**'s actions, do the following:

1. Familiarize yourself with the following **fsck** description and with the information in Appendix A.

2. Continue running **fsck**, but assume you will have to run it again. Grant permission only for minor repairs. Assess the condition of the file systems. (Examples of minor repairs: removing a small or unimportant file; linking a file to **lost+found**.)

3. For each damaged file system, consider how much work would be lost if the file system were thrown away and restored from back up. If this loss is not significant, abandon further work on the file system and restore from back up.

fsck DESCRIPTION

The messages and suggested actions **fsck** displays are presented below. Run with the **-p** option, **fsck** takes most of its own suggestions, stopping only when action would mean loss of data.

For a discussion of the terminology employed in this description, see Appendix A, "File System Concepts."

The messages below each report a disk problem and suggest a standard fix.

(CONTINUE) Results may be invalid. Continue anyway?

(CLEAR) Clear this i-node?

(REMOVE) Remove this directory entry?

(FIX) This value is wrong. Replace it with the right value?

(RECONNECT) We have a good i-node but no directory entry for it. Make a directory entry for it in **lost+found**?

CAN NOT SEEK: BLK B **(CONTINUE)**
CAN NOT READ: BLK B **(CONTINUE)**
CAN NOT WRITE: BLK B **(CONTINUE)**

I/O failed on the file system. If you decide to continue, do a second run to confirm the results of the first. Make sure the disk is not write-protected.

PHASE 1: CHECK BLOCKS AND SIZES

UNKNOWN FILE TYPE I=I **(CLEAR)**

I-node I has an invalid type. If you decide to clear the i-node, its directory entries will be **UNALLOCATED** in Phase 2.

LINK COUNT TABLE OVERFLOW (CONTINUE)

An **fsck** internal table is full. A second **fsck** run will be necessary to confirm the results of the first. This message will repeat each time **fsck** encounters an allocated i-node whose link count is Ø.

B **BAD I=**I

Block B on i-node I is bad. You will get a **BAD/DUP** message in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=I **(CONTINUE)**

I-node I has a large number of bad blocks. If you choose to continue, **fsck** will skip to the next i-node. Run **fsck** again to verify your results.

B **DUP I=**I

I-node I claims block B, but this block is already on **fsck**'s list of allocated blocks. This will cause a **BAD/DUP** message in Phase 2 and Phase 4. This error invokes Phase 1b. Be careful if you see this error. A good procedure is to note the i-numbers with this error and finish running **fsck** without changing the file system. Before running **fsck** again, run **ncheck** to discover the names of the affected files as follows:

/etc/ncheck -i numbers
/dev/rdsk/name

where

numbers — is a list of i-numbers. The numbers are separated from each other by spaces.

name — is the same special file name used with **fsck**.

EXCESSIVE DUP BLKS I=I **(CONTINUE)**

I-node I has a large number of duplicate blocks. If you choose to continue, **fsck** will skip to the next i-node. Run **fsck** again to verify your results.

DUP TABLE OVERFLOW (CONTINUE)

An **fsck** internal table is full. A second **fsck** run will be necessary to confirm the results of the first. This message will repeat each time **fsck** encounters a duplicate block.

POSSIBLE FILE SIZE ERROR I=I

The indicated file has the wrong number of blocks for a file its size. A file-size error does not necessarily indicate a real error; it can indicate a file that was written to nonsequentially.

DIRECTORY MISALIGNED I=I

Size of the indicated directory is not a multiple of 16.

PHASE 1B: RESCAN FOR MORE DUPS

B DUP I=I

I-node I claims block B, but this block is already on **fsck**'s list of allocated blocks.

PHASE 2: CHECK PATHNAMES

ROOT I-NODE UNALLOCATED. TERMINATING

The root directory of a file system is always linked to i-node 2. If i-node 2 is not allocated, the file system is damaged beyond repair.

ROOT I-NODE NOT DIRECTORY (FIX)

I-node 2 must be a directory.

DUPS/BAD IN ROOT I-NODE (CONTINUE)

Some of the duplicate blocks belong to i-node 2. This will make it very difficult to repair the file system.

I OUT OF RANGE I=I NAME=F (REMOVE)

F, a directory entry, refers to an i-node that does not exist.

UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE)

The specified directory entry is a link to an unallocated i-node.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE)

The specified directory entry is a link to an i-node with bad blocks or blocks duplicated by another file.

BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T

You specified the -q option and **fsck** spotted inconsistent data in the specified directory.

PHASE 3: CHECK CONNECTIVITY

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

The indicated directory is nonempty and uncorrupted but lacks a directory entry. (Its former parent has no link to it.)

SORRY, NO lost+found DIRECTORY

No files can be reconnected until you replace the missing **lost+found** directory. If you really need to reconnect your unreferenced i-nodes, do the following: Finish this **fsck** run (being careful not to clear any i-nodes!). Then recreate the missing **lost+found** directory (see "Creating the **lost+found** Directory," above). Finally, run **fsck** on the file system again.

SORRY, NO SPACE IN lost+found DIRECTORY

No more files can be reconnected until you expand the **lost+found** directory. If you really need to reconnect your unreferenced i-nodes, do the following: Finish this **fsck** run (being careful not to clear any i-nodes!). Then expand the **lost+found** directory (see "Creating the **lost+found** Directory," above). Finally, run **fsck** on the file system again.

DIR I=I1 **CONNECTED. PARENT WAS I=**I2.

The directory whose i-number is I1 now has a link in **lost+found**. **Fsck** has made the directory's .. entry refer to **lost+found**; formerly .. referred to i-node I2.

PHASE 4: CHECK REFERENCE COUNTS

UNREF FILE I=I **OWNER=**O **MODE=**M **SIZE=**S **MTIME=**T **(RECONNECT)**

The indicated ordinary file is nonempty and uncorrupted but lacks a directory entry (the directories that had links to it lost them).

SORRY, NO lost+found DIRECTORY
SORRY, NO SPACE IN lost+found DIRECTORY

See the **lost+found** messages in Phase 3.

(CLEAR)

A chance to abandon the last **UNREF** file without rerunning **fsck**. Be absolutely sure you want to clear this i-node.

LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)

The link count for the specified ordinary file is X, but Y files actually have links to it.

LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)

The link count for the directory is X, but Y files actually have links to it.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Ordinarily, unreferenced and empty files and directories silently disappear. If the -n option is specified, this prompt appears for empty files and directories.

BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)
BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

The specified directory or file is unreferenced and has bad or duplicate blocks.

FREE I-NODE COUNT WRONG IN SUPERBLK (FIX)

fsck's count of free i-nodes does not match the count in the superblock.

PHASE 5: CHECK FREE LIST/BITMAP

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)
EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)

This is your last chance to avoid reconstructing the free list.

BAD FREEBLK COUNT
X BAD BLKS IN FREE LIST/BITMAP
X DUP BLKS IN FREE LIST/BITMAP
X BLKS MISSING

Final notes on the dire state of the free list. Any of these will invoke the **BAD FREE LIST** message.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)

Fsck's count of free blocks does not match the value in the superblock.

BAD FREE LIST/BITMAP (SALVAGE)

If the file system is otherwise all right, it is always a good idea to salvage the free list.

FS STATE BAD

File system is marked dirty. Do you want to mark it clean? Respond yes unless there is still damage.

Example 8-1

Configuring MightyFrame Disks
(Page 1 of 6)

In this example, you have three 40-Mbyte on-board disks you want to configure for a MightyFrame system with 5 Mbytes of physical memory. Your requirements are as follows:

- There should be a dump area on one of the disks.
- There will not be a download area, since you are using only RS-232-C terminals.
- There will be two fairly equal-sized file systems in addition to the **root** and **/usr** file systems installed when you loaded CTIX.
- There will be one very large slice without a file system to be used for a data base.
- There should be 8 1/2 Mbytes of swap space, including add-on swap space.

To configure the disks,

1. First create a directory to hold the work files. Then examine the description files that describe the current state of each of the disks:

```
# cd /usr/lib
# mkdir iv.work
# cd iv.work
# /etc/iv -d /dev/rdsk/c0d0s0 > desc.winch0
# /etc/iv -d /dev/rdsk/c0d1s0 > desc.winch1
# /etc/iv -d /dev/rdsk/c0d2s0 > desc.winch2
```

Example 8-1

Configuring MightyFrame Disks
(Page 2 of 6)

2. When you **cat desc.winch0**, you get the description file presented earlier in this chapter. You determine that the disk has no dump area and no download area. There are four slices altogether:

   - Slice 0 is a 32-block reserved area.
   - Slice 1 is a 12,000-block slice containing the root file system.
   - Slice 2 is a 6000-block swap slice.
   - Slice 3 is an 18088-block slice. (Refer to the calculations on slice size in the subsection, "Determining Disk Dimensions.")

3. Check **/etc/mountable** to see if there is a file system in slice 3:

   ```
   # cat /etc/mountable
   /etc/mount /dev/dsk/c0d0s3 /usr
   ```

   This tells you slice 3 has a file system mounted on the **/usr** directory.

4. **cat desc.winch1** and **desc.winch2**. These files are identical, except for the bad blocks listed. There are two slices on each disk: slice 0 is 8 blocks and slice 1 is 36112 blocks.

Example 8-1

Configuring MightyFrame Disks
(Page 3 of 6)

5. You decide to adjust slice boundaries as follows:

   - dØ needs no rearranging of slice boundaries, since there is no room to add additional slices.

   - dl will have three slices altogether: an 8-block reserved area, and two slices for file systems. The total number of logical blocks on this disk is 7 (heads) x 645 (cylinders) x ((17-1) / 2) = 36120. The number of logical blocks available for file systems is 36120 - 8 = 36112. If the first file system is 15992 blocks, the second file system is 36112 - 15992 = 20120 blocks.

     The partition table description for this disk will look like this:

     ```
     $
     Ø
     8
     16000 $
     $
     ```

   - d2 will have three slices altogether: a reserved area large enough to contain a dump, a 2 1/2-Mbyte add-on swap space, and one large slice for the data base.

Example 8-1

Configuring MightyFrame Disks
(Page 4 of 6)

The Reserved Area needs to be the size of physical memory (5 Mbytes) plus one track. The Reserved Area description will look like this:

```
$
badblocktable   1
dump            5120
$
```

For d2, the partition table description will look like this:

```
$
0
5128
7628 $
$
```

6. Change each of the disk description files in **/usr/lib/iv.work** using a text editor.

   - **desc.winch0** needs only one modification: Change the name field from 52HD to the serial number of the disk. (The serial number is written on the sheet of paper containing bad block information for the disk.)

   - Change the name field and the partition table description in **desc.winch1**.

   - Change the name field, Reserved Area description, and partition table description in **desc.winch2**.

Example 8-1

Configuring MightyFrame Disks
(Page 5 of 6)

7. Bring the system to single-user mode.

8. Test each of the disk description files:

```
# /etc/iv -o /dev/null desc.winchØ
# /etc/iv -o /dev/null desc.winchl
# /etc/iv -o /dev/null desc.winch2
```

9. Reconfigure each of the disks using the **iv** command with the **-u** option:

```
# /etc/iv -u /dev/rdsk/cØdØsØ desc.winchØ
# /etc/iv -u /dev/rdsk/cØdlsØ desc.winchl
# /etc/iv -u /dev/rdsk/cØd2sØ desc.winch2
```

10. Make file systems in **dlsl** and **dls2**; then label them and mount them. To mount slice 1 on **/u** and slice 2 on **/u/src**:

```
# /etc/mkfs /dev/rdsk/cØdlsl
# /etc/mkfs /dev/rdsk/cØdls2
# /etc/labelit /dev/rdsk/cØdlsl u winchl
Current fsname: u, Current volname...
# /etc/labelit /dev/rdsk/cØdls2 src winchl
Current fsname: src, Current volname...
# mkdir /u
# /etc/mount /dev/dsk/cØdlsl /u
# mkdir /u/src
# /etc/mount /dev/dsk/cØdls2 /u/src
```

Example 8-1

Configuring MightyFrame Disks
(Page 6 of 6)

11. Now run the **mklost+found** program for each file system.

```
# cd /u
# /etc/mklost+found
creating 256 slots...
removing dummy files...
done
drwxrwxrwx  2 bin   bin   4192 Oct 9 15:57 /u/lost+found
# cd src
# /etc/mklost+found
drwxrwxrwx  2 bin   bin   4192 Oct 9 15:58 /u/src/lost+found
```

12. Add the two entries to **/etc/checklist**. When you have finished editing the file, it will look like this:

```
/dev/rdsk/c0d0s1
/dev/rdsk/c0d0s3
/dev/rdsk/c0d1s1
/dev/rdsk/c0d1s2
```

13. Add the two entries to **/etc/mountable**. When you have finished editing the file, it will look like this:

```
/etc/mount /dev/dsk/c0d0s3 /usr
/etc/mount /dev/dsk/c0d1s1 /u
/etc/mount /dev/dsk/c0d1s2 /u/src
```

14. Finally, add the **swap** command to **/etc/drvload** so that the add-on swap slice on d2 becomes part of the total available swap space. When you have finished, the end of the **/etc/drvload** script will look like this:

```
swap -a /dev/dsk/c0d2s1
echo added swap c0d2s1
```

9 ADDING AND SUPPORTING USERS

This chapter describes what is required to support MightyFrame users. It explains how to

- add users to the MightyFrame system
- provide users an appropriate CTIX environment
- allow users to run two special CTIX commands: **at** and **crontab**
- move user files when file systems become crowded
- make CTIX as secure as necessary for users
- use CTIX programs to communicate with users
- deny users access to the system

ADDING USERS

To give a new user basic access to the system, follow the steps below:

1. Assign the user a unique login name.
2. Choose a file system and home directory to hold the user's file.
3. Create an entry for the user in the password file.
4. If the user is a member of one or more groups, edit **/etc/group**.
5. Create the user's home directory.

Example 9-1, at the end of this chapter, shows how to add users.

NOTE

There is an administrative tool to add a new user. See the CTIX Administration Tools Manual for a description of the tool.

ASSIGNING A LOGIN NAME

The login name uniquely identifies the user. The login name must be one-to-eight characters long and consist of letters or digits. If the name has letters, at least one of them must be lowercase. Two users must not have the same login name.

CHOOSING A FILE SYSTEM AND HOME DIRECTORY

Decide which file system will hold the user's permanent files. If possible, reserve one or more file systems for normal (nonsystem) users. Divide the file systems between users based on your estimate of the users' storage needs.

Name user home directories in a way that is convenient for you. One good system follows these two conventions:

- The user's home directory is in the root directory of the user's file system.
- The simple name of the home directory is the same as the user's login name.

Decide on the name of the user's home directory, but do not create the actual directory yet. This will be easier to do once you have created the user's password file entry.

EDITING /etc/passwd AND RUNNING passwd

Each user requires an entry in the CTIX password file, **/etc/passwd**. This file is read by **login** every time someone tries to log in. (See "**login**" in Appendix B.) To add a new user to the password file:

1. Log in as **root**.

2. Use a text editor to add the user's entry to the file.

3. Use the **passwd** program to assign a password to the new user.

NOTE

It is strongly recommended that you use the following procedure when you edit **/etc/passwd** or any other important system file that CTIX may read while you are working on it:

1. Make two copies of the current version of the file, using **cp**: one of the copies will be the working copy, the other an "oldfile" for reference.

2. Edit the working copy of the file, not the current version of the file.

3. Test the working copy of the file. **pwck**(1) tests the **passwd** file for errors.

4. Replace the current file with the working copy, using **mv**.

Appendix B describes the format of **/etc/passwd**. To add a new user to this file, either modify one of the template lines, or add a new line, as follows:

1. Enter the user's login name in the first field.

2. Leave the second field empty. The **passwd** program enters the encrypted password in this field. (After the encrypted password has been entered by **passwd**, you can edit this field to cause the password to expire on a certain date. (See **passwd**(1).)

3. Enter a unique user ID (UID) number in the third field. The UID must be a decimal number greater than or equal to 100. **The numbers 0-99 are reserved: Do not use them.**

4. If the user is associated with a group, enter the group ID (GID) number in the fourth field; if the user is not in a group, set this field to 100. The numbers 0-99 are reserved: Do not use them. (See "Adding a User to a Group" below.)

5. Enter the user's full name or some information useful for accounting purposes in the fifth field. (The fifth field has no standard usage. Several CTIX programs, however, expect to find information in this field that identifies the user in some way other than the login name.)

6. Enter the full pathname of the user's shell in the sixth field. (If this field is empty, the Bourne shell is assumed.)

To specify a password for the new user, run **passwd**. The form of the command is

passwd name

where name is the user's login name. **passwd** prompts for a password, and the user enters his or her password. The password must be at least six characters, must have at least one numeric character and at least two alphabetic characters, and must differ from the user's login name. (See **passwd**(1) for other restrictions on the password.)

The password does not appear on the screen. To prevent error, **passwd** makes you type the password twice.

If you do not run the **passwd** program (leaving the second field of the password file entry blank), the user requires no password to log in. This creates an insecure environment and should be avoided. See "Making CTIX Secure" below.

The "sync" user requires no password and no home directory of its own. Having a **sync** entry in **/etc/passwd** allows nonprivileged users to run the **sync** command.

The CTIX program **pwck**(1) scans **/etc/passwd** for inconsistencies; run **pwck** to check for errors.

ADDING A USER TO A GROUP

A user can be a member of one or more groups. Only one group ID (GID) per user can be specified in **/etc/passwd**. You can give a user membership in other groups by specifying the user's name in the appropriate entries in **/etc/group**.

To add a user to an existing group, add the user's login name to the last field of the entry for that group in **/etc/group**. Login names of group members are separated by commas.

To establish a new group, create an entry for the group in **/etc/group**. (See **group**(4).)

NOTE

You can run the **passwd** program to give the group a password-protected login; that way, a user is prompted for a password when he or she enters the **chgrp** command.

However, the practice of establishing groups and group logins is a system security risk and is discouraged.

CREATING THE USER'S HOME DIRECTORY

To set up the new user's home directory, follow these steps:

1. Create the directory.
2. Give the user ownership of the directory.
3. Give the user's group group ownership of the directory.
4. Set the protection mode of the directory.

Use **mkdir** to create the home directory. The command has the form

mkdir dir

where dir is the home directory name. Actually, any number of directory names are permitted.

Use **chown** to give the user ownership of the home directory. The command has the form

chown name dir

where

name is the user's login name.

dir is the user's home directory.

Use **chgrp** to give the user's group group ownership of the home directory. The command has the form

chgrp group dir

where

group is the name of the group or the numerical group ID.

dir is the user's home directory.

Use **chmod** to set the protection mode of the user's home directory. The command has the form

chmod mode dir

where

mode is a protection mode. **755** gives the owner all access to the directory, and gives other users read and search (execute) access. **700** gives the owner all access to the directory and gives other users no access at all. For other modes, see **chmod**(1).

dir is the user's home directory.

Each of these commands can take as input multiple files or directories.

SETTING UP A USER'S CTIX ENVIRONMENT

PROFILE FILES

The **/etc/profile** file establishes the default environment for all Bourne shell users. A Bourne shell user's individual environment is established in his or her own **$HOME/.profile** file. **/etc/profile** is executed before **.profile**; therefore, any environment variables that you set in **/etc/profile** can be reset by a user in **.profile**. Users can establish their own environments by creating **.profile** files, or you can create **.profile** files for them.

It is strongly recommended that you set the **umask** (CTIX file creation mode mask) in **/etc/profile**. With no mask, files and directories are created with a mode of **666** (rw-rw-rw-). Adding the following line to **/etc/profile** sets the **umask** to **22**:

```
umask 22
```

When the **umask** is set to **22**, all files and directories are created with a mode of **644** (rw--r--r--). (See **umask**(1) and Appendix B.) It is a good idea to inform users about the **umask** and to discourage users from having files on the system with an excessive degree of privilege for other users.

If **/etc/profile** invokes the **tset** command to set the TERM environment variable, or if all users have the same terminal type, users are not absolutely required to have a **.profile** file.

It is particularly important to set the search path variable, $PATH, in an intelligent way. The goal should be both to minimize the time the system spends searching for commands and to make the search path convenient for the user. Since $PATH is read left to right, it is most efficient to have the most likely place to find the command listed first (**/bin**); however, users may prefer to have the current directory searched first. Here are some other guidelines:

- Make absolutely sure a directory is not searched more than once.
- Fewer entries make for a speedier search.
- Searches of large directories should be avoided if possible. Put any large directory paths at the end of $PATH.

A typical search path that begins at the current directory (current directory is implied by the initial ":") is shown below.

```
PATH=:/bin:/usr/bin:/usr/local/bin
```

Users may want to search their own binary files before they search **/bin**; for example,

```
PATH=:$HOME/bin:/bin:/usr/bin:/usr/local/bin
```

As system administrator, you will probably want to add **/etc** to the search path in your **.profile**.

Appendix B describes the distributed version of **/etc/profile** and gives guidelines for creating **.profile** files. Appendix B also describes the login sequence for C shell users.

SETTING THE TZ (TIME ZONE) VARIABLE

As distributed, **/etc/TZ**, the time zone file, is specified as PST (Pacific Standard Time). Edit this file to change the time zone.

ALLOWING USERS TO RUN crontab AND at

crontab(1) and **at**(1) are special CTIX programs that users can run only when you have permitted them to do so. **crontab** allows users to use **cron** to execute commands at specified times according to instructions in users' **crontab** files. **at** allows users to execute commands at a time specified on the command line. Note that after a user has set up a **crontab** file, he or she must run the **crontab** command to copy the file into the directory containing all users' **crontab** files.

The **at.allow**, **at.deny**, **cron.allow**, and **cron.deny** files are either zero-length, if they exist, or specify names of users who are allowed or denied the privilege to run **at** and **crontab**.

To permit all users to run **crontab**, make sure there is a zero-length **cron.deny** file in **/usr/lib/cron** or that all user names are listed in **cron.allow**. To permit all users to run **at**, make sure there is a zero-length **at.deny** file in **/usr/lib/cron** or that all user names are listed in **at.allow**.

To deny all users permission to run **crontab**, make sure there is a zero-length **cron.allow** file in **/usr/lib/cron** or that all user names are listed in **cron.deny**. To deny all users permission to run **at**, make sure there is a zero-length **at.allow** file in **/usr/lib/cron** or that all user names are listed in **at.deny**.

To allow users permission to run **crontab**, add user names to both **cron.allow** and **cron.deny**. To allow only certain users permission to run **at**, add user names to both **at.allow** and **at.deny**.

MOVING USERS

If space runs short on a file system, you may need to move a user to a new file system. This requires the following steps.

1. Inform the user of his or her new home directory name.

2. Copy the user's files to their new location.

3. Update the user's password file entry.

4. Delete the user's old files.

Use the same procedure for assigning moved home directory names that you use for assigning new home directory names.

Example 9-2 illustrates how to move a user to a different file system.

The following command will create the file copies in the new file system. The copies will have the same modification dates as the originals; this is desirable if the user uses **make**.

```
( cd old ; find . -depth -print |
cpio -pdmuv new )
```

where

old is the full name of the old directory.

new is the full name of the new directory.

It is a good idea to perform a system backup before you remove a large number of files, both to have an off-line archive of a user's files and to safeguard against inadvertently removing the wrong files.

To remove the old user files, use **rm**:

```
rm -rf old
```

where

old is the full name of the old directory.

MAKING CTIX SECURE

The information below provides some guidelines and procedures to (1) prevent accidental loss of data, (2) help guard against violations of system security, and (3) alert you to possible system security breaches.

It is beyond the scope of this manual to provide information about every risky practice and every penetrable spot in the CTIX system. In fact, it is impossible to make CTIX entirely secure from tampering.

The intent here is to identify some risky and unnecessary practices and suggest some obvious (but often neglected) precautions you ought to take.

PREVENTING ACCIDENTAL LOSS OF DATA

CTIX allows you to remove files and directories very easily and without confirmation. Even the most experienced CTIX users and administrators occasionally remove files they never intended to remove. Here is how to minimize the nuisance of restoring lost data:

- NEVER be logged in as superuser when you do not absolutely require that privilege. Normal users can inadvertently destroy their own files. The **bin**, **sys**, or **lp** user can inadvertently destroy some system files and not others. The superuser can inadvertently destroy any and all system files and files belonging to other users.

- Make sure users who should not have system privilege are not actually exercising privilege. (See "Avoiding Violations of System Security" below.)

- Back up all system and user files in a way that makes recovery of lost data easy. Test your backup strategy to make sure it works. It is possible to back up diligently every day only to find out that recovering an individual lost file from backup is a four-hour procedure! (See Chapter 12, "Backups and Restores.")

AVOIDING VIOLATIONS OF SYSTEM SECURITY

The most useful advice is also the most obvious advice:

- Make sure all users have passwords; make sure users change their passwords frequently.
- Encourage users to invent passwords that are not easy to guess (such as names of spouses and friends).
- Strongly encourage users not to divulge their passwords.
- Do not divulge the superuser password; change the superuser password frequently.

Here are some other ways to help prevent abuse:

- Avoid changing privilege of CTIX programs to set UID.
- Do not use group passwords. Keep or put "NONE" in the group password field.
- To make sure users change their passwords, cause user passwords to expire. **passwd**(1) contains information on how to do this.
- Make sure the **umask** is set in **/etc/profile**.
- Discourage users from leaving their logged-in terminals unattended for long periods of time.

- Always specify HUPCL (hang up on close) in the **gettydefs** options for a channel used by a modem. Encourage users to always log out before they end a dial-up session. If a dial-up connection is terminated before logging out, users should dial in and log out before terminating the session again.

  In the case of modems controlling multiple channels, it is especially important for dial-up users to check the process table when they log in after an unwanted termination. If their process from the previous session is running on a channel different from their current channel, they need to kill the process.

- When a user leaves, invalidate the user's password entry. Either remove the entry entirely, or add a character to the password field as described below.

- Police the system regularly for possible security violations, as described below.

CHECKING FOR POSSIBLE SECURITY BREACHES

Take these steps to find out if anyone has exercised unauthorized privilege:

- Check the **/usr/adm/sulog** file regularly. This file is a list of all privileged user logins.

- Look for suspicious program privileges and for any changes to **/bin** and **/usr/bin**.

The following shell script detects changes in CTIX program permissions and writes the results to a file that is readable only by **root**. This example script can be called from **root's crontab** file:

```
find /bin /usr/bin -mtime +2 -print > /usr/adm/findlog
find / -perm -04000 -print >> /usr/adm/findlog
chmod 400 /usr/adm/findlog
```

COMMUNICATING WITH USERS

The following CTIX programs and files are specifically intended to help you communicate with users about system-related events:

/etc/motd This is an ASCII file, "message of the day," that is called from **/etc/profile** and printed on the user's screen when the user logs in. Edit the file to inform users about scheduled down times, for example.

news(1) This program is called from **/etc/profile** and prints files in **/usr/news**. You can use **news** for any communication where you want to update information without having to delete dated information.

wall(1) This program sends an immediate message to all currently logged-in users. It is typically invoked to inform users that the system will be shut down.

BARRING AND DELETING USERS

The following paragraphs describe procedures for denying users access to the system. "Barring a User" describes how to deny access when the user may not be removed permanently. "Permanently Removing a User" describes how to completely undo the steps that gave the user access and how to remove the user's files.

Example 9-3 illustrates how to permanently bar a user from the system.

BARRING A USER

To bar a user without permanently removing him or her, invalidate the user's password file entry. One way to do this is to insert a **%** at the beginning of the entry's encrypted password; use a text editor to do this.

When a user's password file entry is invalid, any attempt by that user to log in is rejected as "incorrect."

To restore the user, remove the **%**.

NOTE

There is an administrative tool that allows you to bar a user. The tool does not remove the user's files. See the CTIX Administration Tools Manual for a description of the tool.

PERMANENTLY REMOVING A USER

It is a good idea to postpone permanent removal of a user until after regular file backups. If this is inconvenient, consider using the temporary procedure, above, until the next regular backup.

To permanently remove a user from the system:

1. Invalidate the user's password file entry.

2. Use the **find** command to search for all files the user may have created, and then decide which of these files should be removed from the system.

3. Remove the user's files.

4. Remove the user's password file entry.

Use a text editor to invalidate the user's password field as described above.

The former user may leave files in places other than his or her home directory. The following command does a comprehensive search for the former user's files, but takes a lot of time to execute:

find / -user *x* **-print**

where

x is the user's login name or numeric user ID. If the user's password file entry is gone, the login name will not work but the numeric user ID will.

rm will remove the user's home directory and all the files it contains. The command takes the form

rm -fr *dir*

where

dir is the user's home directory.

CAUTION

The above form of the **rm** command can remove a large number of directories quickly. Note that the **rm** command does not announce the files it is removing; use this command carefully and precisely.

If a user's file outlasts the user's password file entry, run **ls -l** on the file to find the former user's numeric user ID.

Use a text editor to remove the password file entry.

Example 9-1

Adding Users
(Page 1 of 2)

You have four new users who choose login names "john," "dick," "gwen," and "jack." You decide that there is room for them on the file system whose special file name is **/dev/dsk/cØd1s1**. This file system is normally mounted on **/a**. The new users' home directories will be **/a/john**, **/a/dick**, **/a/gwen**, and **/a/jack**.

1. Create password file entries for "john," "dick," "gwen," and "jack." Each of these users uses the Bourne shell. (You have previously given root and yourself, "susan," a password; you have deleted the "demo" entry for security reasons.)

   To make changes to the password file, make two copies of the file, one as a working copy, the other an "oldfile." Edit the working copy.

   When you have finished editing the working copy of **/etc/passwd**, it will look like this:

```
root:ltswVzihmJ/4Y:0:0:Root:/:
daemon:NONE:1:1:Admin:/:
bin:NONE:2:2:Admin:/bin:
sys:NONE:3:3:Admin:/usr/src:
adm:NONE:4:4:Admin:/usr/adm:
uucp:NONE:5:1:uucp:/usr/lib/uucp:
nuucp:NONE:6:1:uucp:/usr/spool/uucppublic:/usr/lib/uucp/uucico
sync::20:1:MightyFrame sync command:/:/bin/sync
lp:NONE:71:2:lp Administrator:/bin:
isam:NONE:1000:3:ISAM Administrator:/user/isam:
sccs:NONE:90:7:SCCS Pools:/source:
susan:dh71yq4YwG9GA:101:100:Susan Jones:/a/susan:
john::110:100:John Smith:/a/john:
dick::111:100:Dick Spencer:/a/dick:
gwen::112:100:Gwen Ross:/a/gwen:
jack::113:100:Jack Richards:/a/jack:
tdl::103:100:Pt/Gt 232 auto-download:/:/usr/local/bin/tdl
```

620-015

Example 9-1

Adding Users
(Page 2 of 2)

2. Next, run the **passwd** program for each of the new users. (The users provide their own passwords.)

```
# passwd john
new password: ######
retype new password: ######
# passwd dick
new password: ######
retype new password: ######
# passwd gwen
new password: ######
retype new password: ######
# passwd jack
new password: ######
retype new password: ######
#
```

3. Now provide home directories for "john," "dick," "gwen," and "jack." Make each directory with all access for the owner and read-only access for other users.

```
# mkdir /a/john /a/dick /a/gwen /a/jack
# chown john /a/john
# chown dick /a/dick
# chown gwen /a/gwen
# chown jack /a/jack
# chgrp 100 /a/john /a/dick /a/gwen /a/jack
# chmod 755 /a/john /a/dick /a/gwen /a/jack
#
```

Example 9-2

Moving a User's Files to a Different File System

You want to move "gwen" from the file system mounted on /a to a file system mounted on **/b**.

1. Change the home directory for "gwen" in **/etc/passwd**. The new entry will look like this:

   gwen:UC)W7.pjZUBcw:112:100:Gwen Ross:/b/gwen:

2. Copy gwen's files to **/b/gwen** as follows:

```
# (cd /a/gwen; find . -depth -print |
cpio -pdumv /b/gwen)
# rm -rf /a/gwen
#
```

Example 9-3

Barring a User

In this example, you permanently bar "jack."

1. Invalidate jack's password entry by inserting a **%** in front of his password.

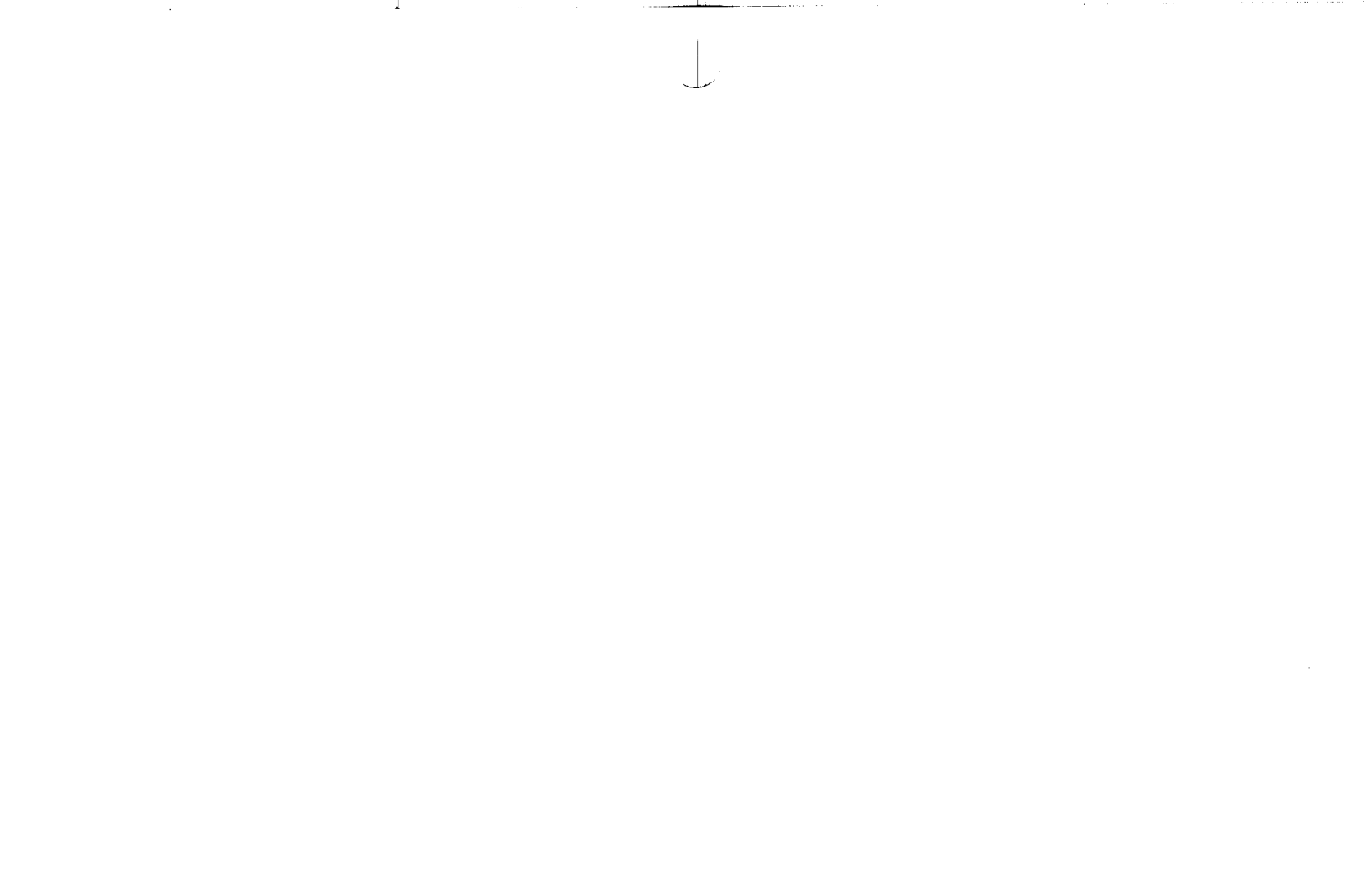
2. Do a comprehensive search of the system for files belonging to jack:

   find / -user jack -print

3. Remove jack's files. (All of jack's files are in **/a/jack.**)

   # rm -rf /a/jack

4. Remove jack's entry from **/etc/passwd.**



This chapter explains how to configure individual UUCP (UNIX-to-UNIX Copy Program) communication links and how to configure a MightyFrame system that runs UUCP. The discussion is of three parts:

- basic UUCP concepts
- configuring a UUCP communication link
- maintaining the system that runs UUCP

NOTE

This chapter describes UUCP configuration files and demons. However, it is not a complete reference to UUCP user and administrative commands. For this purpose, refer to the following entries in your CTIX operating system manual:

cu(1C)	**uusched**(1M)
ct(1C)	**uustat**(1C)
uucheck(1M)	**uusub**(1C)
uucico(1M)	**uuto**(1C)
uucleanup(1M)	**Uutry**(1M)
uucp(1C)	**uux**(1C)
uugetty(1M)	**uuxqt**(1M)
uulog(1C)	

This chapter documents the version of UUCP released with 5.1Ø MightyFrame CTIX software. Versions of UUCP released prior to 5.1Ø have a substantially different administrative interface. However, the current version of UUCP is completely communications-compatible with the previous version.

BASIC UUCP CONCEPTS

UUCP employs the same communication methods used by users on interactive terminals. A communication link is established between two computer systems when one computer system calls another. The call is required even when the two systems are directly connected.

CTIX UUCP supports three kinds of communication links:

Direct link

> The two systems are directly connected, normally with two RS-232-C ports connected by a null modem cable.

Telephone link

> Each system has a modem. The system that exercises the link must have a smart modem (a modem that can dial up another modem). Note that the CTIX kernel currently supports smart modems but not 801 automatic call units.

Network

> Each system is using an Ethernet or X.25 network interface. Network interfaces are sometimes called "built-in" functions.

NOTE

This chapter does not give complete information about configuring UUCP to be used with TCP, UCBTCP, or INET network connections. For information about these protocols and configuring UUCP files for such connections, see the CTIX Internetworking Manual.

If a system uses multiple telephone links, it does not need a modem for each link, although multiple modems increase the number of links that can be active at one time.

The examples below configure the network shown in Figure 10-1. System alpha and system beta are at the same location; system homebase is at a different location in the same city; system gamma is in a different part of the country. There is one direct link: alpha to beta. There are three telephone links: alpha to gamma; beta to homebase (bidirectional); and homebase to gamma. The direct link runs at 9600 baud, the telephone link runs at 1200 baud. Note that gamma controls no links; all UUCP jobs that originate at gamma must wait for a call from alpha or homebase.

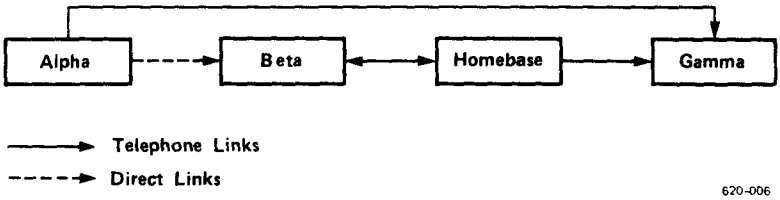


Figure 10-1. UUCP Configuration Example

The actual sending and receiving of data is done by the UUCP copy-in/copy-out demon, **uucico**. (A demon is a program that normally runs in the background.) **uucico** runs with its effective user ID set to **uucp**, implementing the UUCP feature that all UUCP commands are executed as that user. Each communication link requires a **uucico** running on each system, in dialog with the **uucico** on the other system. The **uucico** on the system that initiates communication is the master; the other **uucico** is the slave.

uucico on the master system is normally invoked in one of two ways:

- Directly, by **uucp** or **uux**. This occurs when a user invokes **uucp** or **uux** or when another program (such as **mail**) calls **uucp** or **uux**.

- Indirectly, by the UUCP scheduling subsystem via an entry in the **uucp crontab** file. **uudemon.hour** (called from **crontab**) periodically calls **uusched**, which calls **uucico** to perform the data transfer.

(The other, less common, way to invoke **uucico** is directly with debugging options or via **Uutry**. Debugging is described later in this chapter.)

C., D., and X. FILES

Every time **uucp** or **uux** is invoked, it performs the following functions:

- Creates a subdirectory under **/usr/spool/uucp** (if the subdirectory does not already exist) with the name of the remote system to be called.

- Creates a **C.** file in the subdirectory. The **C.** file is a request file containing information about a **uucp** or **uux** job to be performed (for example, options specified to **uucp**, pathnames of files to copy, and so forth). If the **-C** option was specified to **uucp**, files to be copied to the remote system are copied to **D.** files in the spool subdirectory.

  If **uux** was invoked, it creates a **D.** file with commands to be executed and other **D.** files to hold data, as required.

- Unless the user specified the **-r** option (**-r** tells **uucp** or **uux** to create the request, thereby queuing the job, but not to call **uucico**), **uucp** calls **uucico** with the **-s** option, which specifies the remote system to be called.

If **uucico** successfully performs the data transfer, it completes any necessary chores (like making logfile entries, and informing the user that the transfer is complete if the user requested notification) and then deletes the **C.** and **D.** files. How the master **uucico** establishes the connection, communicates with the slave **uucico**, and, in cooperation with the slave **uucico**, performs the transfer is described in more detail below.

In the case of a **uux** request, the **D.** file containing commands to be executed will have been transferred to an **X.** file. **uucico** calls **uuxqt** to process the **X.** files after the exchange is complete.

If the data transfer is not performed, the **C.** file (and **D.** files, if they were created) remain in the spool subdirectory until one of the following events occurs:

- The scheduling subsystem, performing a routine check for unfilled requests, calls **uucico**, and **uucico** successfully transfers the data.

- **uucico** is called on behalf of a different request (when **uucico** is called, it exercises the link on behalf of all queued requests for a particular remote system), and **uucico** successfully transfers the data.

- **uucico** on the remote system exercises the link and, in cooperation with the local **uucico**, transfers the data.

- The request becomes sufficiently dated for **uucleanup** to remove it and send mail to that effect to its originator.

- A user or administrator deletes the **C.** file.

The Scheduling Subsystem

uudemon.hour is called once an hour (or at any other interval you specify) by **cron**; upon execution, it calls **uusched**, the scheduling demon. **uusched** scans the **uucp** spooling subdirectories under **/usr/spool/uucp** for unfilled requests. When **uusched** encounters a **C.** file, it calls **uucico** with the **-s** option, and **uucico** attempts to exercise the link with the remote system. If a connection is established and a successful data transfer occurs, **uucico** removes the **C.** file and any associated **D.** files.

uudemon.poll provides a mechanism to ensure that an attempt to exercise the link is made at regular intervals, whether or not there is an actual request waiting in the local queue. This is particularly useful when the remote system is passive. For example, in a case where the remote system does not have dial-out capability, or is not authorized to dial-out to the phone number of the local system. The local system is incapable of knowing whether there is a request waiting in the remote queue unless it forces the link to be exercised.

uudemon.poll is called from

/usr/spool/cron/crontabs/uucp

and is normally started a few minutes before **uudemon.hour**. Each time **uudemon.poll** is called, it performs the following functions:

- Checks the Poll file (**/usr/lib/uucp/Poll**) for information about the systems to be polled this hour.

- Creates a spool subdirectory for each system polled, if a does not already exist.

- Creates a zero-length **C.** file, if there are no **C.** files currently in the sub-directory.

The existence of a "dummy" **C.** file causes **uusched** to call **uucico** and attempt to exercise the link. If **uucico** fails to establish a connection with the remote system the first time it tries, it attempts to establish a connection the next hour when called by **uusched**, and so forth. (This is the same procedure **uusched** and **uucico** would follow if the **C.** file were an actual request.)

File Transfer with uucico

uucico is the program that performs the actual data transfer. Several **uucico** processes can be running at once. **uucp** and **uux** spawn **uucico**, to make sure a new job is transmitted right away, if possible; thus, the more users use the UUCP system, the more **uucicos** are run. Lock files prevent two **uucicos** from communicating with the same remote system at the same time and from trying to use the same modem or line. The UUCP system file prevents **uucicos** from calling a system at a time forbidden by the administrator.

Each time a master **uucico** calls a system, master and slave follow the procedure described below. To simplify things, SysM designates the master's system, SysS the slave's.

1. The master checks the SysM UUCP system file to see when calls to SysS are permitted and to find out how to call SysS. If the time is wrong or if SysS must establish all communication with SysM, the master gives up on calling SysS (end of procedure).

2. The master checks the lock files in the SysM UUCP spooling directory. If all possible communication lines are locked (in use by other programs, including other **uucicos**) or if SysS is locked (another SysM **uucico** already talking to SysS), the master gives up on calling SysS (end of procedure).

3. The master creates lock files to claim the communication line and to prevent other SysM **uucicos** from talking to SysS.

4. If the line requires a telephone connection, the master opens the line and has the smart modem or automatic call unit dial up SysS. If the line is direct or a network link, the master just opens it.

5. Emulating a user on a terminal, the master logs in to SysS, using a special UUCP user name (for example, "nuucp"). The special user's shell is **uucico**; since the SysS **uucico** is run without the option making it a master, it runs as a slave.

6. The master tells the slave that SysM is calling.

7. The slave checks to see if communication with SysM requires call-back. If call-back is not required, go to the next step. If call-back is required, the slave refuses to talk to the master, executes a master **uucico** of its own, and terminates. The slave's termination logs the master off of SysS. The SysM master removes its lock files and gives up on calling SysS (end of procedure).

8. The slave creates a lock file for SysM in the SysS UUCP spooling directory.

9. Master and slave execute SysS's UUCP jobs queued on SysM.

10. The master asks the slave if it wants to terminate the conversation. If SysS does not have jobs queued for SysM, the slave responds "Yes"; go to step 13. If SysS has jobs queued for SysM, the slave says "No"; go to the next step.

11. Master and slave execute SysM's UUCP jobs queued on SysS.

12. The slave asks the master if it wants to terminate the conversation. If SysM does not have jobs for SysS, the master says "Yes"; go to the next step. If SysM has jobs for SysS (queued while SysS's jobs were being executed), go to step 9.

13. The slave removes SysM's lock file and terminates; the slave's termination logs the master off SysS. The master closes the line and removes its lock files. The master is done with SysS (end of procedure).

CONFIGURING COMMUNICATION LINKS

Each communication link requires the following configuration steps.

1. Install the necessary communication hardware.

2. Assign a node name to each system.

3. If the system that makes the calls uses a smart modem, makes sure the modem name is specified in the dialer's file.

4. Make sure that **getty** does not monitor the terminal interfaces that will be used for outgoing calls. **uugetty** is permissible.

5. If desired, create an appropriate special file for the caller's lines.

6. Configure the caller's terminal interfaces for use by UUCP.

7. Make sure that **getty** or **uugetty** monitors the terminal interfaces that will be used for incoming calls.

8. Provide a user name on the called system for use by the caller.

9. Change the UUCP system file on the caller to make calls to the other system. Change the UUCP system file on the called system to accept calls from the other system.

10. Configure each system to place necessary security restrictions on the other system.

Some of the following steps need not be repeated for each new communication link:

- A node name is usually assigned only once. Only conflicts (such as when two networks are merged) require a new node name.

- A single terminal interface and smart modem can call more than one system.

- A single terminal interface and modem can receive calls from more than one system.

- A single user name can handle logins from more than one system. However, for security reasons, it is recommended that each system have a different user name and password.

- If the system file allows calls to be made to another system, it also allows calls from that system unless restrictions are imposed on that system in the **Permissions** file entry.

Thus, at some steps in the procedure above it is only necessary to ensure that the existing configuration supports the new link.

HARDWARE INSTALLATION

RS-232-C channels on MightyFrames can handle communication up to 38,415 baud, although 9600 baud is the recommended limit.

A direct link requires a null modem (cross connected) cable. (This is the standard Convergent Technologies RS-232-C terminal cable.) Limited distance modems or line drivers may be necessary for cables longer than fifty feet.

A telephone link requires a smart modem on the calling system and a compatible autoanswer modem on the other system. See the installation instruction for your brand of modem.

To find out the CTIX terminal line numbers for the RS-232-C channels you have chosen, see Chapter 4, "MightyFrame Physical I/O Channels."

In the examples below, assume the following hardware configuration. B & F is an imaginary manufacturer of autoanswer, autodial modems that run at 1200 baud.

alpha	cable to beta on tty001 B & F modem on tty002
beta	cable from alpha on tty001 B & F modem on tty002
homebase	B & F modem on tty001
gamma	B & F modem on tty001 (Since gamma is a passive system, it could use an answer-only modem.)

ASSIGNING A NODE NAME

The node name is the system name that appears in all UUCP system designations. The **setuname** command sets a system's node name as follows:

setuname -n name

where name is the node name. Note that names are truncated to eight characters.

setuname must be executed each time that CTIX is rebooted. To make this happen automatically, modify the command in the start-up script, **/etc/rc**.

Either of the following commands verifies that the node name has been correctly set:

```
uname -n
uuname -l
```

For example, to set the node name to homebase, enter the following command:

```
setuname -n homebase
```

To provide for future automatic setting of the node name, change the **setuname** command in **/etc/rc** to

```
setuname -n homebase
```

See the discussion of **/etc/rc** in Appendix B.

SMART MODEM NAMES

The smart modems that UUCP can use are described in the text file **/usr/lib/uucp/Dialers**. This file is documented under **Dialers**(5) in your CTIX operating system manual. New modems are defined simply by editing the file.

In following examples, the name for the imaginary B & F smart modem is assumed to be **bf**.

getty OR uugetty ON THE CALLING SYSTEM

uugetty is a special form of the **getty** program used for dial-in/dial-out connections. Unlike **/etc/getty**, **/usr/lib/uucp/uugetty** does not interfere with dial-out connections, because it relinquishes control of the line when a dial-out connection is made by **uucico**, **cu**, or **ct**. **/etc/getty**, on the other hand, must not monitor lines used for calling other systems. If a line

is used for outgoing connections only, it should not have either **getty** or **uugetty** monitoring it. (See Chapter 5, "Terminals and Modems," and Appendix B for more information about **getty** and **uugetty**.)

The **who** command tells you the lines that **getty** and **uugetty** monitor:

```
who -l
```

or

```
who -l | grep ttyxxx
```

where xxx is the three-digit terminal number. The second form of the command restricts output to the line you are interested in. If **getty** or **uugetty** monitors the line, **who** prints a status for the line. You are interested in the status's first field (who or what is using the line), the second field (the terminal number), and the fifth field (the numeric ID of the process controlling the line).

To remove **getty** from the terminal line, or to change the monitor to **uugetty**, perform the following steps:

1. If no status is shown, neither **getty** nor **uugetty** is monitoring the line. Skip the remaining steps. If the first field of the user status is "LOGIN," no one is using the line. Go to the next step. If the first field in the **who** status is a user name, that user is using the line. Have the user log off and get another **who** status.

2. Edit **/etc/inittab** according to the procedure outlined in Chapter 5, "Terminals and Modems." If **uugetty** will monitor the channel, make sure there is a suitable **gettydefs** entry for incoming calls to the modem. See Appendix B for the format of the **/etc/inittab** and **/etc/gettydefs** files.

3. Have **init** reread **/etc/inittab**:

   telinit q

CAUTION

Use the **telinit** command carefully and precisely. The wrong parameter will stop CTIX suddenly and painfully.

4. Repeat the **who** command to verify that you did the last three steps correctly.

In the examples below, the systems from the previous examples use the following lines for calling other systems:

alpha	tty00l (direct line to beta) tty002 (smart modem)
beta	tty00l (direct line from alpha) tty002 (smart modem)
homebase	tty00l (smart modem)
gamma	tty00l (smart modem)

In this example, homebase's administrator removes the **getty** from tty001 and replaces it with **uugetty**. First he or she checks the status of the channel as follows:

```
# who -l | grep tty001
LOGIN    tty001      Apr 12 11:20    old     5309
```

Next, the administrator changes the two entries for this channel in **/etc/inittab** so they look like this:

```
001:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty001 M1200

C001:6:off:/etc/getty tty001 C9600
```

The **-r** option to **uugetty** is required, since this is a smart modem. A timeout of 60 seconds is also specified. The label field for the first entry in this example specifies a 1200-baud entry in **/etc/gettydefs** that the administrator has modified. (See Example 5-2 in Chapter 5.) The administrator does not want **uugetty** to monitor the channel when the MightyFrame is in state **6** (administrator mode). Therefore, he simply changes "respawn" to "off" in the C001 entry.

Finally, the administrator makes **init** reread **/etc/inittab** to verify that the process number of the program monitoring the channel has changed, as follows:

```
# telinit q
# who -l | grep tty001
LOGIN    tty001      Apr 12 11:25    old     5325
```

CREATING SPECIAL FILES FOR CALLING

New special files for calling lines are not strictly necessary, but do help to avoid mistakes when using the lines. Accessing the wrong modem or direct line is apt to be less drastic a mistake than accessing a terminal line not intended for use by **uucp**.

Use **ln** to create a special file to call over a direct line:

ln /dev/tty_xxx_ **/dev/d.**_name_

where

xxx is the terminal interface's three-digit line number.

name is the name of the system that the direct line calls.

Also use **ln** to create a special file to call over telephone lines:

ln /dev/tty_xxx_ **/dev/cua**_y_

where

xxx is the terminal interface's line number.

_y_ is a sequence number that identifies the particular outgoing phone line.

The last part of the special file name (**d.**_name_ or **cua**_y_) is the _device name_. The device name is used in UUCP configuration files and by interactive communications programs such as **cu**.

The following command verifies that a terminal line special file is equivalent to a special file:

ls -li /dev/ttyxxx **/dev/**devname

where

xxx — is the terminal interface's line number.

devname — is the corresponding device name.

The **ls** command prints out two file status lines, each of which begins with an i-number. The two i-numbers should be the same.

In the following examples, the administrators on alpha, beta, and homebase create calling special files. The first example is for alpha.

```
# ln /dev/tty001 /dev/d.beta
# ln /dev/tty002 /dev/cua0
#
```

The next example is for homebase.

```
# ln /dev/tty001 /dev/cua0
```

Note that this configuration is not necessary on the system being called because the lines used by UUCP are just login terminals to the called system.

CONFIGURING THE CALLER'S TERMINAL INTERFACE

The UUCP lines file, **/usr/lib/uucp/Devices**, configures lines used to call out. This text file has two kinds of lines: comments, which are ignored, and line configurations. A comment begins with a pound sign (#). Prototype examples are given as comments at the beginning of the file. Use these examples as a reference when configuring your devices.

Figure 10-2 shows the relationship between the **Devices** file and the **Systems** and **Dialers** files.

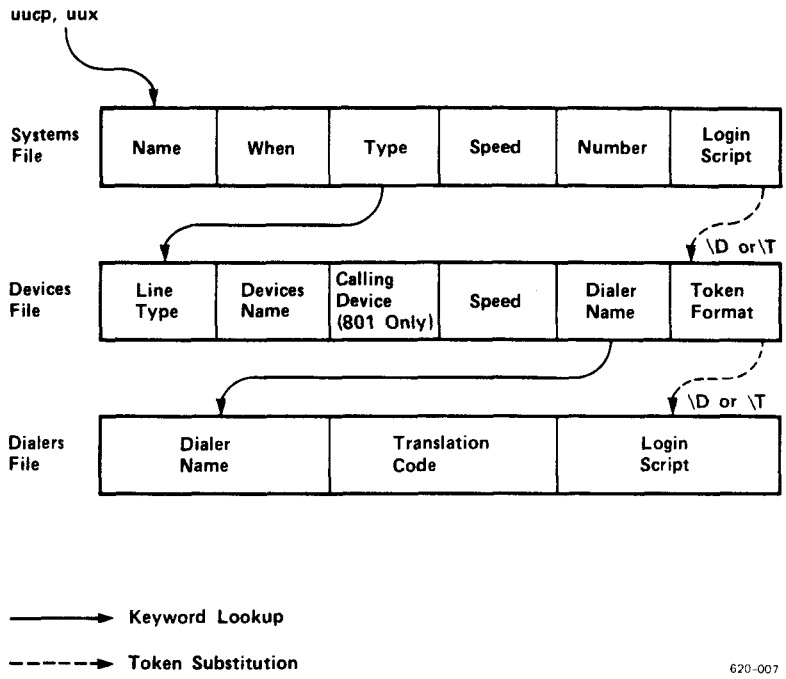


Figure 10-2. Systems, Devices, and Dialers Files

A line configuration describes how to connect your system to the remote system using a particular device type and is a text line in the following form:

```
type name cd speed dialer tokenformat [...] [dialer tokenformat]
```

where

type	is the line type. systemname (as specified in the third field of the **Systems** file) for direct lines; **ACU** for phone lines (both smart modem and automatic call unit); **TCP**, **INET**, or **BSDTCP** for one of the Ethernet protocols (see the CTIX Internetworking Manual); and other types as defined in the prototype examples at the beginning of the **Devices** file.
name	is the device name (**d.**name or **cua**y). For TCP, the value is **unused**.
cd	is the calling device (if it is an 801 dialer) and is specified as **801**. For any other device, including TCP, the value is **unused**.
speed	is the normal baud rate of the line: the value **Any** or a specific baud rate. For TCP, the value for this field is **Any**.
dialer	is the name of the caller as specified in the first field of the **Dialers** file or the name of a built-in function (for example, TCP). Use **direct** for a direct line; **TCP** for a TCP Ethernet connection; **ventel** if this is the name you specified in the **Dialers** file for your ventel modem, and so forth.

tokenformat is the token passed to the **Dialers** file or to the built-in. It can include a **\D** (meaning "do not translate the phone number token in the **Dialcodes** file") or **\T** (meaning "translate the phone number token in the **Dialcodes** file"). **\T** is used for built-in functions (like TCP) that require expansion. **\D** and **\T** are filled in by the number token passed from the number field in the **Systems** file.

For a TCP entry, this field specifies the TCP logical port to connect to on the remote system.

If more than one dialer is involved, more dialer-tokenformat pairs are specified. In this case, each dialing routine is applied in succession. This is useful for accessing smart modems attached to data switches. If no tokenformat is specified, the default **\D** is applied to callers in the **Dialers** file and the default **\T** is applied to built-ins. If more than one pair is specified, **\T** or **\D** must be used as a placeholder.

The following examples configure the calling lines on alpha, beta, and homebase, using "bf" modems. The alpha administrator appends the following lines to **/usr/lib/uucp/Devices:**

```
beta d.beta unused 9600 direct
ACU cua0 unused 1200 bf \D
```

The beta administrator appends the following line to **/usr/lib/uucp/Devices:**

```
ACU cua0 unused 1200 bf \D
```

(Note that with respect to both ACU entries above, **\D** is the default and does not need to be specified.)

Homebase is the same as beta.

getty OR uugetty ON THE CALLED SYSTEM

Lines for receiving UUCP calls can be configured precisely like lines for users calling in. In fact, a telephone dial-up line can serve both UUCP and ordinary user logins. (Direct lines work both ways too, but the **cu** program is required to get access to them.) To configure login lines, see Chapter 5, "Terminals and Modems." Note that **uugetty** should be used if the called line is bidirectional or has a smart modem attached; otherwise, **getty** can be used.

UUCP does not require the called system to answer the call with a particular terminal line.

UUCP USER NAMES

A system that wants to accept calls from other systems must provide a user whose home directory is **/usr/spool/uucppublic** and whose shell is **/usr/lib/uucp/uucico** (the copy in/copy out demon). The called system must also mention the user name in the UUCP **Permissions** file **LOGNAME** entry.

CTIX is distributed with such a user, named **nuucp**. Use the **passwd** command to set **nuucp's** password (see **passwd**(1) in your CTIX operating system manual and Chapter 9 and Appendix B of this manual). Disseminate **nuucp's** password carefully, as it gives access to the network.

The system will work with **nuucp** as the only special user, but additional UUCP user names provide the following security features:

- You can change the password for one UUCP user, locking out one group of systems without affecting communications with other (presumably more trustworthy) systems.

- UUCP allows you to impose file copying restrictions in addition to those imposed by CTIX file permissions. These restrictions can divide neighboring systems into classes according to the name they use to log into the local system.

See Chapter 9, "Adding and Supporting Users," to create new user names.

If the **nuucp** user is not already in the **Permissions** file, **/usr/lib/uucp/Permissions**, and if it is the only UUCP login allowed, the LOGNAME option must be specified as follows:

LOGNAME=nuucp

The above entry in the **Permissions** file allows any remote system using the **nuucp** login name and password to log in with default permissions or with permissions as specified in the LOGNAME entry. See "Security Measures: the **Permissions** File," below.

In the network example in Figure 10-1, the four systems modify their password and **Permissions** files as follows:

alpha: This system does not want any remote logins. Thus alpha's administrator decides to edit nuucp's entry in **/etc/passwd:**

```
nuucp:NONE::6:1:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

The invalid second field renders logins by other systems impossible. Since alpha does not permit other systems to log in, no LOGNAME entry is necessary in

/usr/lib/uucp/Permissions

(A MACHINE entry is, however, necessary. See the subsection below on the **Permissions** file.)

beta: Beta's administrator decides that other systems at the same site will login as **nuucp**, but offsite (and potentially untrustworthy) systems will login as ouucp. Therefore, he edits the following lines into beta's **/etc/passwd:**

```
nuucp:::6:1:/usr/spool/uucppublic:/usr/lib/uucp/uucico
ouucp:::6:1:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

He then uses **passwd** to assign the password "ourgang" to nuucp and the password "xyzzy" to ouucp. He tells alpha's administrator the nuucp password and homebase's administrator the ouucp passwd.

Next, The administrator edits **/usr/lib/uucp/Permissions** to contain the following LOGNAME options:

LOGNAME=nuucp
LOGNAME=ouucp

homebase: This system currently wants to accept remote system logins by beta. Alpha or gamma may want access later, but if they do they will have the same status as beta. Homebase already has the user nuucp; the administrator gives nuucp the password "greekaccess".

gamma: Gamma treats all the other systems the same. The administrator gives nuucp the password "compo" and adds the standard LOGNAME option to **/usr/lib/uucp/Permissions:**

LOGNAME=nuucp

THE UUCP SYSTEM FILE

The UUCP system file, **/usr/lib/uucp/Systems**, tells those systems that neighbor the local system. If the local system can call the other system, the system file provides one or more procedures for making the call.

Each line in the system file is either a comment or a system line. A comment begins with a pound sign (#) and is ignored. Each neighboring system requires one or more system lines. The system line has one of two forms. The first form permits the other system to call the local system:

name

where

name is the node name of the other system.

The second form allows the local system and the other system to call each other. Do not provide both forms for one system. If there is more than one procedure to call the remote system (as with a system that has more than one phone number), provide one system line for each such procedure; the different procedures will be tried in the order listed. The second form is of the type

name when type speed number login

where

name is the node name of the other system.

when specifies permitted calling times and minimum retry period. The simplest value is **Any**, which places no restrictions. Multiple time fields, each separated by a comma, may be specified. When takes the following form (time and retry are optional; omit the semicolon if you omit retry):

dayshours[,dayshours...,dayshours];[retry]

where

days is a concatenation of abbreviations indicating the days on which the procedure line can be used. Abbreviations are **Su**, **Mo**, **Tu**, **We**, **Th**, **Fr**, and **Sa** for specific days; **Wk** for any weekday; and **Any**, for any day of the week.

hours specifies the time of day when the procedure can be used. If hours is omitted, the procedure can be used any time of day. The period is specified by two 24-hour clock times separated by a dash (for example: **0900-1700** for 9 A.M. to 5 P.M.). The period can extend through midnight.

retry specifies the minimum waiting period, in minutes, after the first unsuccessful call. An exponential backoff algorithm is used to calculate additional retries. If retry is omitted, the minimum waiting period is five minutes. The retry period applies if the last call to name was unsuccessful. The system will not be called again until the minimum waiting period has expired. This feature prevents UUCP from continuously trying to access a system that may be unavailable.

Here are some examples for when: **Any0800-0700** means any time except 7 A.M. to 8 A.M.; **SaSu** means any time from 1201 A.M. Saturday to midnight Sunday; **Wk1800-0600,Sa,Su** means any time during weekdays except 6 A.M. to 6 P.M. and any time Saturday and Sunday; **Any;10** means any time but do not permit the first retry if the initial attempt was unsuccessful and less than 10 minutes ago.

type is the line type as specified in the first field of the **Devices** file. Use system name for direct links; **ACU** for telephone links; **TCP**, **INET**, or **BSDTCP** for one of the Ethernet protocols, and other types as defined in the prototype examples at the beginning of the **Devices** file.

speed is the speed used. This must match the speed in the **Devices** file. The value is **unused** for TCP. **Any** will match any speed.

number is the phone number. The characters have the following meaning:

0-9	dial 0-9
*	dial *
#	dial #
-	delay four seconds
=	wait for next dial tone

On direct lines, this field is unused.

For autodialers, you can specify an alphabetic abbreviation (dialing prefix) before the numeric part of the phone number (for example, **boston555-1212**). The dialing prefix is defined in the **Dialcodes** file.

For data switches such as Develcon and Micom, the number field is the token the switch needs to get to the particular system. It is passed to the token-format string (as a substitution for **\D** or **\T**) in the **Devices** file.

For direct connections, this field is the device name for the channel being called (for example, **d.beta**).

For TCP, the value is **unused**.

login defines the login procedure. Login consists of a list of expect/send sequences:

expect send expect send ...

The last item can be either an expect or send. The local system waits for the remote system to print a line containing the first expect. When it arrives, the local system sends a line consisting of the next send and waits for the next expect, and so on. When the local system sends or reads the last item, the remote system should have let it log in and be running a slave **uucico**.

An expect can include a subprocedure used if the remote system does not print the desired string before a certain period of time. In this case the expect consists of subexpect/subsend sequences:

subexpect-subsend-subexpect-subsend ...

If the remote system prints a line containing the first subexpect, the rest of the expect is skipped. If subexpect does not appear before a certain period of time, the local system sends the next subsend, waits for the next subexpect, and so on.

Below is an example of a login script:

ogin--ogin--ogin nuucp assword xyzzy

This login logs into a remote system as nuucp, passwd xyzzy. If the master **uucico** cannot get a login prompt at first, it tries twice to provoke one by sending an empty line.

The following escape sequences can be included in the login procedure:

\N Send a null character.

\b Send a backspace character.

\c If at the end of a string, suppress the new-line that is normally sent. Otherwise, ignore the sequence.

\d Delay one second before sending or reading more characters.

\n Send a new-line character.

\r Send a carriage-return.

\s Send a space character.

\t Send a tab character.

**** Send a \ character.

EOT Send EOT character (EOT new-line is actually sent twice).

BREAK Send a **Break**.

\ddd Send the character represented in octal.

Examples of system files are shown below. The first example is alpha's system file, which allows it to call (and be called by) gamma and beta:

```
gamma Any ACU 1200 408-5558328 ogin--ogin--ogin nuucp assword compo
beta Any beta 9600 unused ogin--ogin--ogin nuucp assword ourgang
```

The next example is beta's system file, which allows it to call homebase and accept calls from alpha and homebase:

```
alpha
homebase Any ACU 1200 5550101 ogin--ogin--ogin nuucp assword ourgang
```

The following example is homebase's system file. It provides procedures for homebase to call beta and gamma. Calls to gamma are restricted to night hours and weekends to save long-distance costs.

```
beta Any ACU 1200 9w5559393 ogin--ogin--ogin ouucp assword xyzzy
gamma Wk2200 ACU 1200 9w408-5558328 ogin--ogin--ogin nuucp assword compo
gamma SaSu ACU 1200 9w408-5558328 ogin--ogin--ogin nuucp assword compo
```

Finally, gamma's system file permits calls from alpha and homebase:

```
alpha
homebase
```

TESTING THE LINK

You can test a link using UUCP or the **cu** program.

Testing with UUCP

The following steps use UUCP to test a link:

1. Execute a UUCP transfer between the two systems, using the **-r** option to suppress the **uucico** demon. For example:

   uucp -r shortfile name**!~**

   where name is the name of the system to be called.

2. Become the superuser.

3. Run the **Uutry**(1M) program:

   /usr/lib/uucp/Uutry name

   where name is the name of the system to be called.

The demon's debug output contains certain messages that are meant to debug the demon itself and are not documented. But messages relating to your configuration are self-explanatory.

Note that this procedure should transfer the file to the **uucp** login's home directory on the remote system, usually **/usr/spool/uucppublic**.

To find out how normal demons have done, check the log files: **/usr/spool/uucp/.Log/*** (today's activity, organized into files under a subdirectory for each UUCP program); and **/usr/spool/uucp/.Old/Log-Week-1**, and so forth (each day's worth of activity for the last few weeks; this can be adjusted by editing the **uudemon.clean** shell script).

Testing with cu

You can use **cu** in various ways to test different aspects of the configuration and to produce debugging output, if you need it. Ultimately, you should (1) establish a connection to the remote system using the **cu** program, and (2) log into and out of the remote system. (Enter a line beginning with tilde-period (~.) when you are done.)

The following strategy is recommended when you attempt to establish the connection:

1. Run **cu** with the name of remote system as an argument:

   cu systemname

   For example,

   cu beta

   If there are problems (for example, with the **Systems** or **Devices** file) the connection will not be established. Wait for about 60 seconds to receive the "connected" system message. If you do not receive it, interrupt **cu** and proceed to the next step.

2. Run **cu** with the **-d** option (for debugging output) and the name of remote system as arguments:

   cu -d systemname

   Use the debugging information to try to correct the problem, and then retest the connection. If the debugging information is not sufficient to correct the problem, proceed to the next step.

3. Use **cu** to test the line. The procedures to test a direct line and to test a telephone link are different. In either case, first make sure the device is owned by **uucp**; then follow the steps below.

   To test a direct link with **cu**, there must be a separate line in the **Devices** file for **cu** to use. This line has the following format:

   Direct *device* **unused** *speed* **direct**

   where

   device is the device name for the link.

   speed is the baud rate for the line.

   To run the **cu** command on the calling system, enter:

   cu -d -l *device*

   where

   device is the device name for the link, as specified in the **Devices** file.

   To use **cu** on a telephone link, there must be an entry of the following form for the device in the **Devices** file:

   Direct *device* **unused** *speed* *name*

where

device	is the device name for the link.
speed	is the baud rate for the line.
name	is the modem name; for example, **bf**.

The following command tests a telephone link:

cu -d -l device number

where

device	is the device name for the link, as specified in the **Devices** file.
number	is the telephone number of the remote system.

If this version of **cu** does not work, try it without the phone number; you will have to enter modem commands directly. (This may be an indication of a bad **Dialers** entry.)

MAINTENANCE

There are three types of UUCP maintenance:

- automatic maintenance
- security configuration
- emergencies

AUTOMATIC MAINTENANCE

Automatic maintenance is performed by UUCP background programs (demons). Certain chores are standard and are performed by standard UUCP demons, which may be configured.

Each system running UUCP should regularly execute the following standard demons: **uudemon.hour**, at four minutes before each hour; **uudemon.clean**, at 4 A.M. each day; and **uudemon.admin** several times a day, depending on how often you want to examine status information (**uudemon.admin** is optional for systems that send and receive only a small number of UUCP jobs).

If a system must wait for all calls, it may be desirable to have other systems poll it. Use **uudemon.poll** to do this. (See "The Scheduling Subsystem" earlier in this chapter.)

To arrange for **cron** to start the UUCP demons, make sure the following lines are uncommented in **/usr/spool/cron/crontabs/uucp:**

```
56 * * * * /usr/lib/uucp/uudemon.hour > /dev/null 2>&1
0 4 * * * /usr/lib/uucp/uudemon.clean > /dev/null 2>&1
48 8,12,16 * * * /usr/lib/uucp/uudemon.admin > /dev/null 2>&1
54 * * * * /usr/lib/uucp/uudemon.poll > /dev/null 2>&1
```

The demons perform the following functions:

uudemon.hour

Runs **uusched** and **uuxqt**. **uusched** scans the UUCP spooling subdirectories for waiting jobs (unfilled requests, each of which has a **C.** file); for each subdirectory, if **uusched** finds a waiting job, it invokes a master **uucico** with the **-s** option to attempt completion of local UUCP jobs for

a particular remote system. **uuxqt** scans the UUCP subdirectories for **X.** files (command files from remote systems), and, when it finds such a file, executes the remote job on the local system.

uudemon.clean

Uses **uucleanup** to kill all UUCP jobs waiting on the local system that are at least seven days old. Uses **uucleanup** to kill all **uux** commands trying to execute on the local system that are two days old. Tries to send mail regarding any deleted requests to the originators of the requests and sends warning messages for jobs that have been queued for one day. Moves the contents of today's log to **/usr/spool/uucp/.Old** and removes old saved logs. Removes all spool subdirectories with no files. Uses **find** to remove all ordinary files from the public directory, **/usr/spool/uucppublic**, that are more than 30 days old and to remove other old files and directories that are part of the UUCP system. Uses **uustat** to gather status information (UUCP jobs currently in the queue) and mails a daily status summary to **uucp**. Removes old mail from **uucp's** mail directory.

uudemon.admin

Mails UUCP status information (actually, a subset of the status information gathered by **uudemon.clean**) to the UUCP administrator. Your system may not need to run this demon if your volume of UUCP traffic is very small.

uudemon.poll

Checks the Poll file **/usr/lib/uucp/Poll** for information about the systems to be polled this hour (see below). For each system to be polled, checks for any waiting jobs; if there are none, this demon creates a dummy **C.** file (and a subdirectory for the remote system, if no subdirectory exists). This causes **uusched** (when invoked by **uudemon.hour** several minutes later) to call **uucico** and attempt to exercise the link.

Since the demons are shell scripts, you can edit them to add, change, or delete features. Systems that cannot afford the amount of space used by **/usr/spool/uucp** and **/usr/spool/uucppublic** usually shorten the time period parameters for the **uucleanup** and **find** commands in **uudemon.clean**. You may want to change the value of the MAILTO variable in **uudemon.admin** and **uudemon.clean**; note, however, that the UUCP mailbox receives mail from many sources and should be checked on a regular basis.

To set up the polling mechanism, you must edit the **Poll** file, **/usr/lib/uucp/Poll**. Create a one line entry for each system to be polled.

The format of the Poll file is as follows:

systemname ▶ hour hour ... hour

where

systemname is the name of the system to be polled.

▶ is the tab character.

hour — is an hour in 24-hour format to poll the system. Hours are separated by spaces.

The following example shows an entry in homebase's **Poll** file for system **gamma**:

```
gamma      Ø8 11 16 2Ø
```

SECURITY MEASURES: THE Permissions FILE

The **Permissions** file, **/usr/lib/uucp/Permissions**, is a text file that controls the way users and systems use UUCP to access the local system. The Permissions file consists of the following five types of controls that apply to general classes of files and to specific users and systems:

- files local users can use UUCP to copy to remote systems
- files a remote system can access and what their read and write permissions are for the remote system
- login name each remote system must use to talk to the local system
- commands that can be executed for a particular remote system
- if a remote system must be called back to confirm its identity

Format of the Permissions File

Each entry is a logical line; physical lines are terminated with a \ (backslash) to indicate continuation. Entries are made up of space-delimited options. Each option specified in the file must have a value assigned to it; an option and its value are separated by an = (equal) sign. Note that spaces may not appear within an option. Some values can be a list of items, such as commands that are permissible to execute. Items within a list are separated by a : (colon).

Comment lines begin with # (pound sign); they mark the entire line up to the newline character. Blank lines are ignored.

Types of Entries

There are two types of entries:

- LOGNAME entries specify permissions for remote sites when they log in to the local machine.

- MACHINE entries specify permissions for sites when the local machine makes the calls.

If a local system has a bidirectional connection to a remote system (it can call or be called), it must have a LOGNAME entry and a MACHINE entry unless it will use the default options of the MACHINE entry. (See below.)

LOGNAME and MACHINE entries can be combined into a single entry, if the same permissions are to be specified for systems that log in and systems that are called. In this case, the LOGNAME option and the MACHINE option are both included in the entry.

Keep in mind the following basic rules about LOGNAME and MACHINE entries:

- Any login name used by any remote system to login to the local system must appear in one and only one LOGNAME entry.

- Any remote system whose name does not appear in a MACHINE entry has the following default permissions and restrictions:

  - Locally generated send and receive requests will be satisfied.

  - The remote system can send files to the local system's UUCP public directory.

  - The local system will execute a default set of commands for the remote system: the defaults are **rmail**, **rnews**, and **lp**.

The following is a list of options with an explanation of their uses.

Options	Description
MACHINE	A MACHINE entry specifies the permissions in effect when the local system calls a remote system. Multiple remote systems can have the same set of permissions, which can be specified with one MACHINE entry. A list of remote systems contains a : separator between systems. Remote systems are specified by their system names. For example, a MACHINE entry in alpha's **Permissions** file for gamma and beta, which will have the same set of permissions, begins with the MACHINE option specified as MACHINE=gamma:beta A MACHINE entry can specify OTHER as a value for the MACHINE option. This allows systems not mentioned in specific machine entries to have options that are *different from* default options (for example, permitted commands).
LOGNAME	A LOGNAME entry specifies the set of permissions in effect when the local system is called by a remote system. Multiple remote systems can have the same set of permissions on the local system, which can be specified with one LOGNAME entry. Remote systems are specified by their login names. A list of user names contains a : separator between names.

Options	Description
	For example, a LOGNAME entry in gamma's **Permissions** file for alpha and homebase, who will have the same set of permissions on gamma and whose user names are nuucp and xuucp, begins with the LOGNAME option specified as LOGNAME=nuucp:xuucp Note that there must be an entry in **/etc/passwd** for each login. You must also run the **passwd** program to assign a password for each user name. See "UUCP User Names," earlier in this chapter. Note also that user names that appear in one LOGNAME entry can not appear in any other LOGNAME entry on the local system.
REQUEST	A REQUEST option can appear in either a LOGNAME entry or a MACHINE entry and specifies if the remote system can make requests to receive local files. REQUEST=yes specifies that the remote system can request files. REQUEST=no specifies that the remote system can not request files. The latter value (no file access) is the default.

Options	Description
SENDFILES	SENDFILES specifies whether the local system, when called by the remote system, will satisfy file access requests made by the remote system without the local system returning the call. SENDFILES=yes specifies that the local system will satisfy file access requests generated by the remote system without the local system being the calling system. SENDFILES=call specifies that the local system will not satisfy file access requests generated by the remote system unless the local system is the calling system. (The remote system, in this case, should arrange to be polled.) The default is SENDFILES=call, meaning that file access requests will be satisfied only when the local system calls the remote system. This option is meaningful in LOGNAME entries only, and is ignored in MACHINE entries.
READ and WRITE	The READ and WRITE options specify permission to access files in particular directories. The value for each of these options is a colon separated list of directories, specified by full pathnames.

Options	Description

	The READ option specifies permission to copy a file to the remote system; the WRITE option specifies permission to deposit a file on the local system.
	
	Since **uucico** runs with an effective user ID of **uucp**, only file access operations granted to that user will be allowed by normal file protection mechanisms.
	
	Thus,
	
	`READ=/ WRITE=/`
	
	specifies permission to read and write any file on the system accessible by **uucp** (normally by setting the "other" bit in the file protection code).
	
	`READ=/usr/spool/uucppublic \`
	`WRITE=/usr/spool/uucppublic`
	
	specifies read and write permission for the UUCP public directory only. This is the default.

Options	Description
	Note that if the READ or WRITE option is specified (that is, if you want to extend READ and WRITE access to directories other than the public directory), you must specify the full pathnames of all directories, including the default UUCP public directory. For example, WRITE=/usr/spool/uucppublic:/usr/news
NOREAD and NOWRITE	These options allow you to exclude specific directories (and their subdirectories) from the general access permissions granted by READ and WRITE. For example, READ=/ NOREAD=/etc \ WRITE=/usr/spool/uucppublic:/usr/news allows the remote system to read any file on the system with "other" user privilege, except for files in **/etc** and subdirectories of **/etc**; and allows the remote system to write to the public directory and **/usr/news** only.

Options	Description
CALLBACK	The CALLBACK option is used with LOGNAME entries and specifies that no data transfer will take place, but the calling system, as specified during handshake, will be called back. CALLBACK=yes specifies the action. Note that if two sites have this option set for each other, a call will never result in any data transfer. CALLBACK=no is the default and specifies that the local system does not agree to callback (unless it is polling). It will proceed with the transfer unless SENDFILES=call (the default) is in effect.
COMMANDS and VALIDATE	The COMMANDS option specifies which commands the local system will execute for a remote site. This option is meaningful in MACHINE entries only. The VALIDATE option provides some security by requiring specified systems to log in with a particular user name. This option is meaningful in LOGNAME entries only.

Options	Description

CAUTION

The COMMANDS option is inherently insecure. Commands like **cat** and **uucp**, and any other command that reads or writes files, are potentially very dangerous. Using the VALIDATE option adds a measure of security, but does not offer any protection when a remote system gains access to a user name and password it was not supposed to have.

To specify commands to be executed by a remote system, give the command name if the program is in **/bin**, **/usr/bin**, or **/usr/local/bin**. Give the full pathname of the program if it is not in one of those directories.

The default value for the COMMANDS option is

```
COMMANDS=rmail:rnews:lp
```

NOTE

The default value for the COMMANDS option is subject to change. Consult the current Release Notice for possible changes to this value.

Options	Description

To specify that any command may be executed by the remote system, give the value "ALL" to the COMMANDS option.

```
MACHINE=gamma:beta REQUEST=yes \
COMMANDS=ALL \
READ=/ WRITE=/

LOGNAME=nuucp VALIDATE=beta \
REQUEST=yes SENDFILES=yes \
READ=/ WRITE=/
```

in homebase's **Permissions** file gives unlimited file and execution access to gamma and beta (subject to normal CTIX file protection mechanisms).

EMERGENCIES

UUCP creates most of its temporary files in **/usr/spool/uucp**. UUCP users are prone to create many files in **/usr/spool/uucppublic**. Therefore, keep a close eye on the free space of the file system that holds **/usr/spool**--running out will paralyze UUCP and possibly other parts of your system as well. If there is a lot of UUCP activity on your system, consider making a separate partition and file system for **/usr/spool** to protect the rest of the **/usr** file system.

Note that if you restart UUCP on your system after a hiatus, jobs queued on neighboring systems to execute on or pass through your system will arrive all at once, possibly creating a new logjam.

UUCP and some other communications programs create lock files called **/usr/spool/locks/LCK..**name, where name is a device or remote system name. If the system or UUCP crashes while a communication program was working, these files may remain, preventing your restarting communication. Remove the lock files to restart communication, but be sure they don't belong to an active **uucico**, **cu**, or other such program. The safest time to remove a UUCP lock file is when CTIX is in single-user mode. You may find it useful to have the **rm** command in the CTIX start-up script, **/etc/rc**.

ASSERT ERROR MESSAGES

/usr/spool/uucp/.Admin/errors

The following is a list of Assert error messages that can appear in **/usr/spool/uucp/.Admin/errors**. Note that these errors are fatal. The program aborts, and the file name, SCCS ID, and line number appear in the error message along with the text given below. In most cases, the errors result from file system problems; use the error number (errno), when provided, to investigate the problem.

CAN'T OPEN

An open() or fopen() failed; errno appears in ().

CAN'T WRITE

A write(), fwrite(), fprintf(), or other write operation failed; errno appears in ().

CAN'T READ

A read(), fgets(), or other read operation failed; errno appears in ().

CAN'T CREATE

A creat() call failed; errno appears in ().

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; errno appears in ().

CAN'T CHMOD

A chmod() call failed; errno appears in ().

CAN'T LINK

A link() call failed; errno appears in ().

CAN'T CHDIR

A chdir() called failed; errno appears in ().

CAN'T UNLINK

A unlink() call failed; errno appears in ().

WRONG ROLE

This is an internal logic problem.

CAN'T MOVE TO CORRUPT DIR

An attempt to move some bad **C.** or **X.** files to **/usr/spool/uucp/.Corrupt** failed. The directory is missing or has wrong modes or owner.

CAN'T CLOSE

A close() or fclose() call failed; errno appears in ().

FILE EXISTS

The creation of a **C.** or **D.** file is attempted, but the file exists. This occurs when there is a problem with sequence file access and is usually a software error.

No uucp server

A tcp/ip call is attempted, but there is no server for uucp.

BAD UID

The UID can not be found in **/etc/passwd.** The file system is corrupt, or the **/etc/passwd** file is inconsistent.

BAD LOGIN_UID

Same as above.

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 large or unusual request was attempted; send it with MR to the appropriate place.

TOO MANY SAVED C FILES

Same as above.

RETURN FROM fixline ioctl

An ioctl, which should never fail, failed. There is a system or driver problem.

BAD SPEED

An incorrect line speed appears in the **Devices** and/or **Systems** files.

PERMISSIONS file: BAD OPTION

There is a bad line or option in the **Permissions** file. You need to fix it.

PKCGET READ

The other side probably hung up; don't worry about it.

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 some internal problem. Send in an MR.

CAN NOT ALLOCATE FOR

There is some kernel problem. A call to calloc() failed.

XMV ERROR

There is a problem with some file or directory. It is probably the spool directory, since the modes of the destinations should have been checked before the program attempts this.

CAN'T FORK

An attempt to fork and exec failed. The current job should not be lost, but will be attempted later by **uuxqt**. No action need be taken.

/usr/spool/uucp/.Status/sysname

The following messages can appear in the system status file **/usr/spool/uucp/.Status/sysname**:

OK

Things are OK.

NO DEVICES AVAILABLE

There is currently no device available for the call. Check to see that there is a valid entry in the **Devices** file for this system.

TALKING

Self-explanatory.

SOME FAILURE

To be determined.

BAD SEQUENCE CHECK

If sequence checking is used between systems, the sequence numbers do not agree. (This is almost never used.)

LOGIN FAILED

The login for the given machine failed. It could be a wrong login or password, wrong number, a very slow machine, or failure to get through the login script.

CONVERSATION FAILED

The conversation failed after a successful startup. This usually means that one side went down, the program aborted, or the line just hung up.

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 login/machine name that does not agree with our Permissions file. They could be trying to masquerade.

DEVICE LOCKED

The calling device is currently locked and in use by some process.

ASSERT ERROR

An ASSERT error occurred. See **/usr/spool/uucp/.Admin/errors**.

WRONG MACHINE NAME

The called machine is reporting a different name in the Shere= message.

CALLBACK REQUIRED

The called machine requires that it call us.

REMOTE HAS A LCK FILE FOR ME

The remote system has a LCK file for this system. They could be trying to call us. If they have a version of UUCP prior to 5.1Ø, the process that was talking to us may have failed, leaving the LCK file. If they have the current version of UUCP and they are not trying us, then the process that was talking to us is hung.

REMOTE DOES NOT KNOW ME

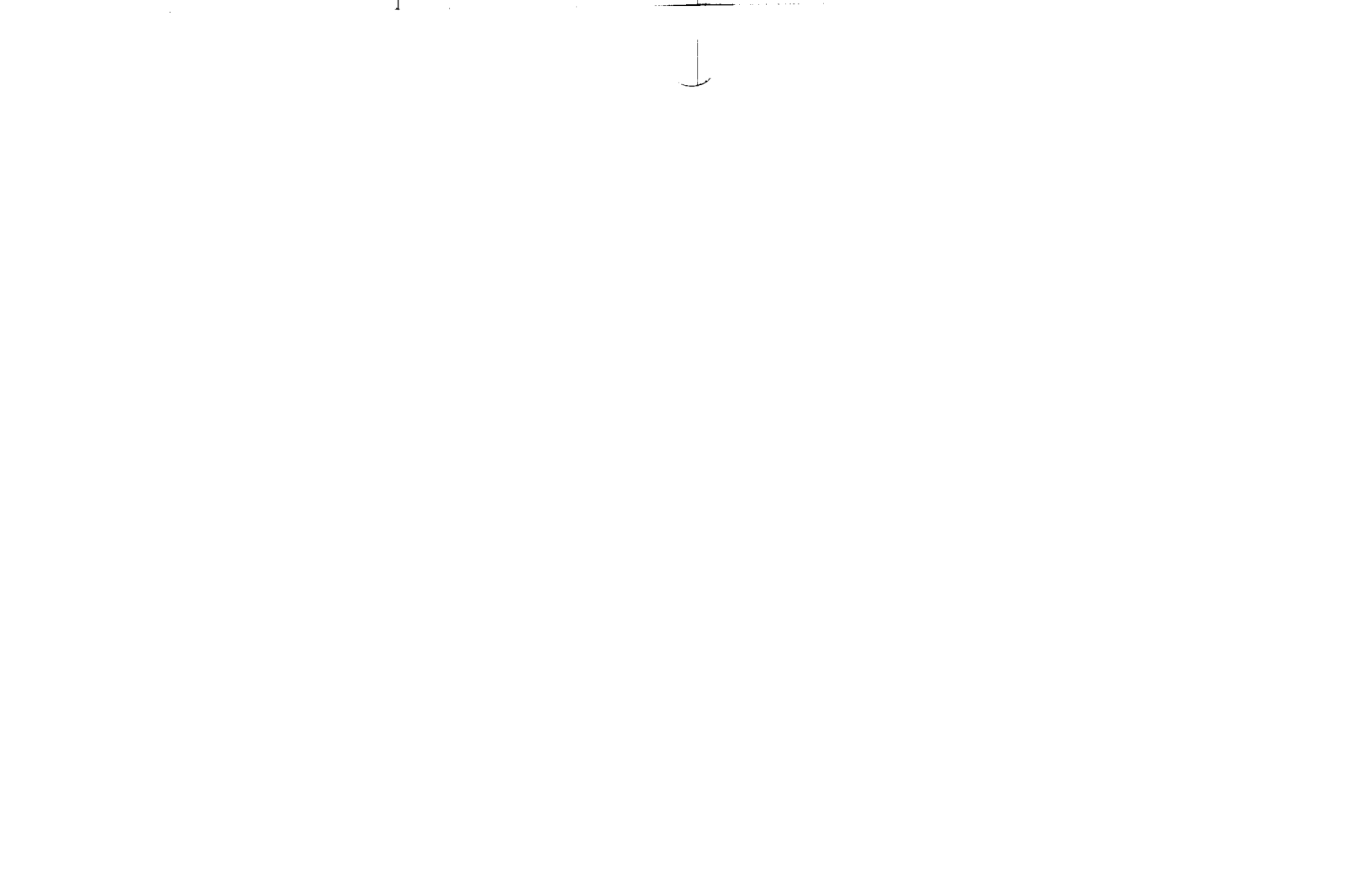
The remote system does not have our name in their **Systems** file.

REMOTE REJECT AFTER LOGIN

The login we used does not correspond to what the remote system is expecting.

REMOTE REJECT, UNKNOWN MESSAGE

The remote system rejected us for an unknown reason; they are probably running a nonstandard version of UUCP.



11 CTIX SYSTEM ACCOUNTING

CTIX system accounting provides methods to collect per-process resource utilization data, record connect sessions, monitor disk utilization, and charge fees to specific logins. A set of C language programs and shell procedures is provided to reduce this accounting data into summary files and reports. This chapter describes the structure, implementation, and management of this accounting system; it also describes the reports generated and the meaning of the columnar data. Tables 11-1 through 11-4, at the end of this chapter, describe system accounting files.

OVERVIEW

The following list is a synopsis of the accounting system:

- At process termination, the CTIX system kernel writes one record per process in **/usr/adm/pacct** in the form described in **acct.h**.

- The **login** and **init** programs record connect sessions by writing records into **/etc/wtmp**. Date changes, reboots, and shutdowns (via **acctwtmp**) are also recorded in this file.

- The disk utilization programs **acctusg** and **diskusg** break down disk usage by login.

- Fees for file restores, and so forth, can be charged to specific logins with the **chargefee** shell procedure.

- Each day **cron** executes the **runacct** shell procedure to reduce accounting data and produce summary files and reports.

- The **monacct** procedure can be executed on a monthly or fiscal period basis. It saves and restarts summary files, generates a report, and cleans up the **sum** directory. These saved summary files could be used to charge users for CTIX system usage.

FILES AND DIRECTORIES

The **/usr/lib/acct** directory contains all of the C language programs and shell procedures necessary to run the accounting system. The **adm** login (currently user ID of 4) is used by the accounting system and has the login directory structure shown in Figure 11-1.

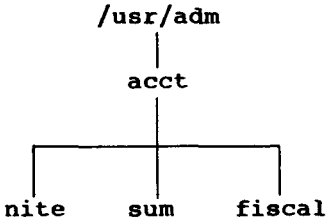


Figure 11-1. Directory Structure of the "adm" Login

The **/usr/adm** directory contains the active data collection files. (For a complete explanation of the files used by the accounting system, see Tables 11-1 through 11-4 at the end of this chapter.) The **nite** directory contains files that are reused daily by the **runacct** procedure. The **sum** directory contains the cumulative summary files updated by **runacct**. The **fiscal** directory contains periodic summary files created by **monacct**.

DAILY OPERATION

When the CTIX system goes to multiuser mode, **/usr/lib/acct/startup** is executed, which does the following:

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 via **turnacct**. **turnacct on** executes the **accton** program with the argument **/usr/adm/pacct**.

3. The **remove** shell procedure is executed to clean up the saved **paact** and **wtmp** files left in the **sum** directory by **runacct**.

cron runs the **ckpacct** procedure 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 **cron** tries to restart **runacct** after a failure processing these records.

The **chargefee** program can be used to bill users for file restores, and so forth. It adds records to **/usr/adm/pacct** which are picked up and processed by the next execution of **runacct** and merged into the total accounting records.

cron executes **runacct** 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 you shut down the system using **shutdown** or **halt**, the **shutacct** procedure is executed. It writes a shutdown reason record into **/etc/wtmp** and turns the process accounting off.

You should execute **/usr/lib/acct/prdaily** every day to print the previous day's accounting report.

SETTING UP THE ACCOUNTING SYSTEM

To automate the operation of this accounting system, perform the following steps:

1. Uncomment the line in **/etc/rc** that calls the **/usr/lib/acct/startup** program.

2. Uncomment the line in **/etc/halt** and **/etc/shutdown** that calls the **/usr/lib/acct/shutacct** program.

3. Uncomment the line in **/usr/spool/cron/crontabs/root** that runs daily accounting.

4. Uncomment the three lines in **/usr/spool/cron/crontabs/adm** that process the daily and monthly accounting data.

NOTE

There is an administrative tool to start the accounting subsystem and to turn it off. See the CTIX Administration Tools Manual for a description of the tool.

runacct

runacct is the main daily accounting shell procedure. It is initiated by **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**, this files 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 log-offs to logins exceeds three to one, 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 This file, produced by the **acctcms** program, contains the daily command summary. The ASCII version of this file is **nite/daycms**.

sum/cms This file contains the accumulation of each day's command summaries. It is restarted by the execution of **monacct**. The ASCII version is **nite/cms**.

sum/loginlog This file, produced by the **lastlogin** shell procedure, 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 is designed to prevent damage to files in the event of errors. A series of protective mechanisms 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 **fd2log**. **runacct** sends a message if the files **lock** and **lock1** 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 **/etc/log/confile**, mail is sent to **root** and **adm**, locks are removed, diagnostic files are saved, and execution is terminated.

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, which shows all 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 filing accounts,

/usr/adm/Spacc?.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/tacct**MMDD, so that **sum/tacct** can be recreated in the event it becomes corrupt 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/rprt**MMDD, remove the locks, then exit.

RECOVERING FROM FAILURE

The **runacct** procedure can fail for a variety of reasons--a system crash, **/usr** running out of space, or a corrupted **wtmp** file. If the **active**MMDD 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 are produced by **runacct**. The recommended recovery actions are included after each error message.

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 for 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 **acctconl** 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 this is the first invocation of the day. The argument MMDD is necessary if **runacct** is being restarted and specifies the month 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/fd1log&
```

- 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&
```

FIXING CORRUPTED FILES

Unfortunately, the accounting system is not completely foolproof. Occasionally, a file will become corrupted or lost. Some of the files can simply be ignored or restored from the backup. However, certain files must be fixed in order to maintain the integrity of the accounting system.

FIXING wtmp FILES

The **wtmp** files seem to cause the most problems in day-to-day operation of the accounting system. When the date is changed and the CTIX 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 **acctonl** to fail. The following steps illustrate how to fix a **wtmp** file.

cd /usr/adm/acct/nite

fwtmp < wtmp.MMDD **> xwtmp**

ed xwtmp

delete all 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 taact ERRORS

If you are using the accounting 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 alright, the latest **sum/tacct.**MMDD should be repaired, then **sum/tacct** recreated. A simple repair procedure is shown below.

cd /usr/adm/acct/sum

acctmerg -v <tacct.MMDD **> xtacct**

ed xtacct

remove all 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.

UPDATING HOLIDAYS

The **/usr/lib/acct/holidays** file contains the prime/nonprime table for the accounting system. The table should be edited to reflect your holiday schedule for the year. The format is composed of the following three types of entries:

Comment Lines:

Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk.

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
```

The time field has a special condition so that 2400 is automatically converted to 0000.

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 366, which indicates 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.

DAILY REPORTS

runacct generates five basic reports upon each invocation. These reports cover the following areas:

- connect accounting
- usage by person on a daily basis
- command usage reported by daily and monthly totals
- report of the last time users were logged in

The following paragraphs describe the reports and the meaning of their tabulated data.

DAILY REPORTS (CONNECT ACCOUNTING)

In the first part of the report, the **from/to** banner should alert you to the report period. The times are the time the last accounting report was generated until the time the current accounting report was generated. This entry is followed by a log of system reboots, shutdowns, power failure recoveries, and any other record dumped into **/etc/wtmp** by the **acctwtmp** program. (See **acct**(1).)

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 terminal lines). The columns are as follows:

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 channel was accessed for a **login**(1) session.

ON — This column does not have much meaning anymore. It used to give the number of times that the channel 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 channel when the **getty**(1M) is first invoked when the system is brought to multiuser state. Where this column comes 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 it grows rapidly, execute **acctconl** to see which tty line is the noisiest. If the interrupting is occurring at a very rapid rate, general system performance will be affected.

DAILY USAGE REPORT

This report gives a breakdown by user of system resource use. Its data consists of the following:

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 (CPU). 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 distributed, prime time is defined as 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. This column really identifies 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 referred to as a "line hog." That is, this person logs in first thing in the morning and hardly uses the terminal the rest of the day. Watch for these kinds of users. This column 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. The disk accounting is accomplished by the program **acctdusg**.

OF PROCS This column reflects the number of processes invoked by a user. Large numbers in this column indicate that a user has a shell procedure that is creating too many processes.

OF SESS This indicates the number of times a user logged onto the system.

DISK SAMPLES This indicates the number of times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.

FEE — This 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, and so forth.

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 reports 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 you an idea about the most used commands. Based on these commands' characteristics of system resource utilization, you will know best how to fine tune the system.

These reports are sorted by TOTAL KCOREMIN, which is an arbitrary measure, but often a good one, for calculating "drain" on a system.

COMMAND NAME — This is the name of the command. All shell procedures are combined together under the name **sh**, since only object modules are reported by the process accounting system. You should monitor the frequency of programs called **a.out** or **core** or any other suspicious name. Use **acctcom** to determine if superuser privileges were used.

NUMBER CMDS — This is the total number of invocations of this particular command.

TOTAL CPU-MIN — This is the total accumulated processing time of this program.

TOTAL REAL-MIN — The total 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)

which gives a relative measure of the total available CPU time used 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 performs.

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 logins and login directories.

SUMMARY

CTIX system accounting was designed to ensure that the system runs smoothly and without error. To make optimal use of the accounting system, 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 provides valuable information and provides accurate breakdowns of the usage of system resources.

Table 11-1

FILES IN THE /usr/adm DIRECTORY

File Name	Function
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**

Table 11-2

FILES IN THE /usr/adm/acct/nite DIRECTORY

(Page 1 of 2)

File Name	Function
active	Used by **runacct** to record processes and print warning and error messages. **active**MMDD is the same as **active** after **runacct** detects an error.
cms	ASCII total command summary used by **prdaily**
ctacct.MMDD	connect accounting records in **tacct.h** format
ctmp	output of **acctconl** program, connect sessions 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.**

Table 11-2

FILES IN THE /usr/adm/acct/nite DIRECTORY

(Page 2 of 2)

File Name	Function
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

Table 11-3

FILES IN THE /usr/adm/acct/sum DIRECTORY

File Name	Function
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**
pacct.MMDD	concatenated version of all pacct files for MMDD, removed after reboot by **remove** procedure.
rprtMMDD	saved output of **prdaily** program
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

Table 11-4

FILES IN THE /usr/adm/acct/fiscal DIRECTORY

File Name	Function
cms?	total command summary file for fiscal ? in internal summary format
fiscrpt?	report similar to **prdaily** for fiscal ?
tacct?	total accounting file for fiscal ?

Offline backup protects CTIX fixed-disk files from the unexpected. Backup provides copies of files and file systems against accident, carelessness, and technical mishap.

CTIX provides five programs to create archive copies of files and file systems on disks and tapes. This chapter introduces the available programs and uses one of these, **cpio**, to demonstrate a general backup and restore strategy. This chapter assumes that you will be using quarter-inch tape, rather than hard disk, as the backup medium.

Backups require four kinds of tasks:

- choose a program to do the copying
- schedule backups
- perform backups
- restore files and file systems from backups

NOTE

There is an administrative tool to perform backups and restores. See the CTIX Administration Tools Manual for a description of the tool.

CHOOSING A COPY PROGRAM

Table 12-1 summarizes the various copy programs CTIX provides for backup and other archive functions. (See your CTIX operating system manual for complete information about each program.) Each program has advantages and disadvantages; your selection should take the following into account:

- How much data you are backing up. If you have 1200 Mbytes to copy, you will probably want to use a very fast program like **dd** or **volcopy**. In general, the advantages gained by selecting one copy program over another increase with the amount of data you are backing up.

- How you partition your disks. Using a program like **dd**, **finc**, or **volcopy** can be tedious when you back up a large number of individual partitions. If you overlap some of them for backup purposes, **dd** gains advantage over the other programs.

- Whether the volumes are mostly full or are relatively empty of data. **dd** and **volcopy** are advantageous when volumes are full.

You may want to experiment a few times with the various programs. When you do select a program for a regular backup routine,

- Test it (that is, try restoring files from your backup tapes).

- If it works, use it consistently for a particular kind of backup.

Table 12-1

SUMMARY OF CTIX COPY PROGRAMS

(Page 1 of 3)

Program	Advantages for Backup	Disadvantages for Backup
cpio	used with **find** to descend directory hierarchy, a good all-purpose program for total and incremental backups, and for restores; prompts at end of tape	you must unmount file systems you do not want to copy if they are lower in the tree than file-systems you are copying; in general, not as easy to copy single file systems
dd	fast; good for single file systems; overlapping disk partitions make convenient units of files for backup; arbitrary block sizes available; the only program that can be used to backup a non-UNIX file system (data base)	difficult to recover individual files (entire archive must be copied back to disk); only one file (partition) per archive; not useful for incremental backup; partitions must be smaller than 50 Mbytes, since **dd** does not prompt at end of tape

Table 12-1

SUMMARY OF CTIX COPY PROGRAMS

(Page 2 of 3)

Program	Advantages for Backup	Disadvantages for Backup
finc	good for single file systems; can be used for incremental backups; **frec** can restore individual files; provides label checking; prompts at end of tape	only on file system per archive; labeling all tapes can be inconvenient
tar	used with **find**, same advantages as **cpio**, recursively descends directory tree without find; can move directory hierarchies; the most portable copy program among UNIX system	same disadvantages as **cpio**, cannot be used for multitape backups

Table 12-1

SUMMARY OF CTIX COPY PROGRAMS

(Page 3 of 3)

Program	Advantages for Backup	Disadvantages for Backup
volcopy	good for single file systems; **frec** can restore individual files; provides label checking; prompts at end of tape; fast, can almost stream tape	not useful for incremental backup; only one file system per archive; copies volume structures along with files (if corrupt, can be a problem); wasteful of tape space if volume is relatively empty, since it copies every track, including tracks not containing data; labeling all tapes can be inconvenient

SCHEDULING BACKUPS

The following sample schedule has the basic features of a good backup schedule:

- Permanent total. Every fourth Friday, every file system is completely copied. The copies are saved permanently.

- Temporary total. Every Friday, except on days when a permanent total backup is done, every file system is completely copied. The copies are saved for four weeks.

- Incremental. Every working day, except on days when total backups are done, all files created or modified since the last total backup are copied. The copies are saved for a week.

- Backups occur at the end of the working day.

Some of the features of this schedule are arbitrary, some are not. Every four weeks may be too often for you to make permanent backups; but if you increase the time between permanent total backups, make the same increase in the time you keep temporary total backups. Total backups need not occur on Fridays, but should occur at the same time each week; backups need not occur at the end of the working day, but the time they do occur should not change from day to day.

The most important feature of this schedule is that it does not permit the loss of more than a day's work due to the complete loss of the fixed disk's files. It also protects files against accidental removal. The longer a file is left on a fixed disk, the harder it is to lose it permanently.

DOING BACKUPS

Most of the rest of this chapter describes a backup strategy using the copy program **cpio**. Certain procedures, described immediately below, are required for any strategy using any program. Other procedures are particular to a program or combination of programs, such as **find** and **cpio**.

PROCEDURES COMMON TO ALL BACKUP STRATEGIES

There are two distinct backup procedures:

- The total backup. Every file system is copied onto tape.
- An incremental backup. A list of files modified since the last total backup is prepared and each file on the list is copied onto tape.

The following steps are required to perform total and incremental backups on quarter-inch tape:

1. Retension each tape the first time you use it. To retension a tape, insert it in the drive and type

   tsioctl -c retension /dev/rmtØ

2. Log in as **root** to perform the backup.
3. If this is a total backup, generate a list of files backed up and register the time for the benefit of this week's incremental backups.
4. If this is an incremental backup, generate a list of files modified since the last total backup.
5. Use the appropriate program to do the actual backup.
6. Print out a log of the files backed up.

NOTE

It is possible to put more than one backup archive on a single tape. However, this chapter does not document the procedure for multiple-archive tapes because the procedure is difficult and prone to accidental erasure of backups. The practice is discouraged.

TOTAL BACKUPS USING cpio

The **find** and **cpio** commands accomplish a total backup of every file system.

NOTE

The strategy outlined here is to back up every file system with one invocation of **find** and **cpio**. **find** recursively descends the entire directory tree, beginning with **/**. If your file systems total more than, say, 300 Mbytes of data, you will find it easier to recover lost data if you have backed up your file systems individually for a total backup.

To back up an individual file system, unmount all file systems that are normally mounted on the file system you are backing up. When you type the **find** command, specify **/** to backup the root file system only, **/dir** to back up a file system normally mounted on the directory named **dir**.

To back up all file systems with one invocation of **find** and **cpio**,

1. Log in as **root**.

2. Make sure all file systems are mounted.

3. Insert the first tape cassette in the tape drive.

4. Type

```
find / -print | sort > /usr/adm/bulog/TOTAL
cpio -ocQ < /usr/adm/bulog/TOTAL > /dev/rmtØ
```

This set of commands performs the following:

- finds every file in every file system
- creates an alphabetically-sorted and time-stamped log file
- copies every file listed in the log file to tape

When **cpio** fills up a tape, the system prints a message telling you it is safe to remove the device and insert the next one in the series.

To print out the log file, type

```
lp /usr/adm/bulog/TOTAL
```

INCREMENTAL BACKUPS USING cpio

An incremental backup copies recently modified files from all file systems.

To do an incremental backup:

1. Log in as **root**.
2. Make sure all file systems are mounted in their normal place.
3. Insert the first tape cassette in the tape drive.
4. Execute these commands:

```
find / -newer /usr/adm/bulog/TOTAL -print |
sort > /usr/adm/bulog/INCd

cpio -ocvQ < /usr/adm/bulog/INCd > /dev/rmtØ
```

where

d is the number of days since the last total backup.

This set of commands performs the following:

- Finds all files that have been modified since the last total backup.
- Creates an alphabetically-sorted and time-stamped log of these modified files (**/usr/adm/bulog/INC***d*).
- Copies all files in the incremental log to tape.

To print out the incremental log, type

lp /usr/adm/bulog/INC_d_

where

_d_ is the number of days since the last total backup.

RESTORES

A mishap can destroy a disk's worth of data or just a few files. Restoring a large amount of data requires copying all or part of the last total backup, then copying all or part of the latest incremental backup. Restoring a few specific files means copying them from the latest backups that have them.

RESTORING FROM TOTAL BACKUP (cpio)

To completely restore all files on total backup, insert the first total backup tape cartridge and enter

cpio -iQcduvm < /dev/rmtØ

When **cpio** reads through a tape, it tells you it is safe to remove the tape and insert the next one in the series.

To restore all files in a specific directory on total backup, first locate the tape or tapes containing that directory, then copy the files. To get a list of all files on a particular tape, type

cpio -iQct < /dev/rmtØ

To restore files in a specific directory, type

cpio -iQcduvm 'dir/*' < /dev/rmt0

where

dir — is the name of the directory containing the files you want to restore.

RESTORING INCREMENTALLY BACKED-UP FILES

To completely restore all files on incremental backup, insert the first tape cartridge from the most recent incremental backup and type

cpio -iQcduvm < /dev/rmt0

When **cpio** reads through a tape, it tells you it is safe to remove the tape and insert the next one in the series.

To restore all files in a specific directory on the latest incremental backup, insert the cartridge containing those files and type

cpio -iQcduvm 'dir/*' < /dev/rmt0

where

dir — is the name of the directory containing the files you want to restore.

RESTORING SPECIFIC FILES FROM INCREMENTAL OR TOTAL BACKUP

Restoring specific files from an incremental or total backup is very similar to restoring all files in a specific directory.

Insert the cartridge containing those files in the drive and type

cpio -iQcduvm *list* **< /dev/rmtØ**

where

list is a list of files to be recovered.

Creating a text file for **cpio**. If you need to recover a large number of specific files, create a text file with all the file names in it, and use the following form of the **cpio** command instead of the one above.

cpio -iQcduvm `cat** *file***` < /dev/rmtØ

where

file is the file that contains a list of files to be recovered.

Note that the second form of the **cpio** command uses accents grave. Do not confuse them with single quotes (').

As with recovering the entire incremental backup, recovering specific files will require intervention each time **cpio** reads through a disk or tape.

13 SYSTEM MAINTENANCE AND TROUBLESHOOTING

This chapter describes how to

- set the date
- check the system console log and arrange to receive other system messages
- watch out for and correct some common events and situations that degrade system performance
- use the maintenance tape
- use the system activity package

SETTING THE DATE

It is important for you check everyday to see if the date is set correctly, especially if you are doing backups and restores.

If you change the date by more than one hour, first bring the system to single-user mode.

To set the date, type

date MMddhhmmyy

where

MM is the month of the year, Ø1 to 12.

dd is the day of the month, Ø1 to 31.

hh is the hour in a 24 hour system, ØØ to 23.

mm is the minute, ØØ to 59.

yy is the year, expressed as the last two digits. If omitted, the year defaults to the current date setting.

For example,

date Ø914ØØ2585

sets the date to September 14, 1985, 12:25 A.M.

CHECKING THE SYSTEM CONSOLE FILE

CTIX sends important messages about the system's activities, problems, and potential problems to the file **/etc/log/confile**. Examine this file periodically for information, such as the following:

- out of resource messages; for example, "disk out of space" or "running low on swap"
- system tables
- information about when the system entered single-user mode and when the system was booted

Use **more** or some other CTIX utility to look at the file:

more /etc/log/confile

RECEIVING OTHER SYSTEM MESSAGES

System messages from various error-logging utilities are normally sent by **mail**(1) to the users **root** and **adm**. It is a good idea to forward these messages to your own mailbox so that you can be sure to read them.

To forward root's and adm's mail messages, make an entry at the beginning of the **/usr/mail/root** file and **/usr/mail/adm** of the form

Forward to person

where

person — is the login name of the user to receive root's and adm's forwarded mail.

DETECTING AND CORRECTING EVENTS THAT DEGRADE SYSTEM PERFORMANCE

The paragraphs below describe several routine actions you should take to detect and correct common causes of system degradation:

- checking for and compacting oversized directories
- watching out for and trimming files that grow
- using the **ps** command to determine if there are an unreasonably large number of active processes

OVERSIZED DIRECTORIES

Directories larger than about 5 Kbytes are very inefficient because of file system indirection. A large **/usr/mail** or **/usr/spool/uucp** directory, for example, can really slow access to files in those directories. Simply removing files from an oversized directory does not make the directory smaller and increase access time.

Use the following command to find directories that are at least 5 Kbytes:

```
find / -type d -size +10 -print
```

Now compact the directory. The following command shows how to compact, for example, the **/usr/mail** directory:

```
mv /usr/mail /usr/omail
mkdir /usr/mail
chmod 777 /usr/mail
cd /usr/omail
find . -print | cpio -plm ../mail
cd ..
rm -rf omail
```

FILES THAT GROW

There are about a dozen CTIX system files that can grow large enough to fill up the **root** or **/usr** file systems. Some of these are system accounting files: these grow only if the accounting system is turned on. Other system files that grow are log files, like **/usr/adm/sulog** and **/usr/lib/cron/log**. The **/tmp** and **/usr/tmp** files can also become large enough to fill up their file systems.

The system initialization process trims most of these files by moving the current file to an oldfile and removing the old oldfile. If several weeks elapse between system boots, however, the size of some of these files can become a real problem.

Follow the steps below if you think the **root** or **/usr** file system may have run out of space:

1. Use the **df**(1M) command to check the number of free disk blocks.

   df -t

   This checks the number of **free** disk blocks in each file system and tells you the total block size of the file system.

2. If one of the file systems is out of disk space, either reboot the Mighty-Frame and see if that corrects the problem, or look for the oversized file. The following command looks for oversized files:

   find / -type f -size +20 -print

   If you locate an oversized file that is expendable (for example, a **/tmp** file), remove it, or reboot and let the system remove expendable files.

USING THE ps COMMAND

Occasionally a process hangs and ties up a tty line, for example. Or one or more processes begins filling up the process table with processes they have spawned.

Learn to use the **ps**(1) command and interpret its output. Follow the steps below if you suspect or know that a process has hung, or that processes are recklessly filling up the process table and slowing the system down:

1. Do a **ps -ef** to locate the problem process or processes and the process ID numbers.

2. Kill the processes that are causing trouble as follows:

   kill PID

USING THE MAINTENANCE TAPE

The MightyFrame CTIX maintenance tape is a bootable tape that contains certain essential system files and programs. When you are locked out of the system (for example, because you have forgotten root's password and you can not modify **/etc/passwd** without being root), the maintenance tape allows you to

- boot CTIX
- attempt to determine why you were locked out if you do not already know
- fix the problem that locked you out

If you have only SMD drives and the information on the EEPROM becomes corrupted, booting from the maintenance tape allows you to edit **/etc/system**, reload the EEPROM, and reboot from an SMD. This critical procedure is documented below.

NOTE

Access to the maintenance tape should be restricted, since it poses an obvious security risk.

When CTIX boots up from the tape, it reads the RAM file system into memory. (The RAM file system's device specification is **c2dØs1**.) The tiny root file system in memory contains these directories and programs:

/bin

- **cat**
- **chmod**
- **cp**
- **cpio**
- **ed**
- **ln**
- **mkdir**
- **ls**
- **mv**
- **rm**
- **sh**
- **stty**
- **swap**
- **sync**

/etc

- **fsck**
- **hinv**
- **icode**
- **icodel**
- **inittab**
- **iv**
- **ldeeprom**
- **mkfs**
- **mnttab**
- **mount**
- **passwd**
- **reboot**
- **splocate**
- **system**
- **umount**

/dev (many entries)

/mnt (mount point for **/dev/dsk/cØdØs1** and **/dev/dsk/cldØs1**)

/tmp

/usr/lib/iv (several disk description file entries)

Note that **vi** is not included: you must use **ed** to modify configuration files without doing a **chroot**.

Also note that while you are booted from the maintenance tape (that is, until you reboot from your normal boot device) the following is true:

- **c0d0s1** is the device specification for the root file system on the first ST506 drive.
- **c1d0s1** is the device specification for the root file system on the first SMD drive.
- **c2d0s1** is the device specification for the root file system on the RAM disk in memory.

When you have fixed the problem, **sync** and **reboot** the system as follows:

```
sync
sync
reboot
```

FIXING A CORRUPT ROOT FILE SYSTEM

If the root file system has become too corrupt to fix in the ordinary way (for example, if **fsck** itself has become corrupt or if **/unix** has been damaged), use the maintenance tape to **fsck** the root file system while it is unmounted. Here are the steps to do this:

1. Put the maintenance tape in the drive and press the **Reset** button.

2. Provide swap space for **fsck** and the other programs. The system prompts you to permit it to use **/dev/dsk/c0d0s2** or, in the case of an SMD, **/dev/dsk/cld0s2**, as swap (this is the normal fixed swap device). Either allow the system to use this slice or issue the **swap -a** command to add swap space.

   You can remove the tape when you get a shell prompt.

3. Run **fsck** on the root file system. If the root file system is on the first ST506, the command is

   fsck -b /dev/rdsk/c0d0s1

   If the root file system is on the first SMD, the command is

   fsck -b /dev/rdsk/cld0s1

REPAIRING IMPORTANT SYSTEM FILES

To edit or manipulate files in **c0d0s1** or **cld0s1**,

1. Mount the root file system on **/mnt**. If the root file system is on the first ST506, the command is

   mount /dev/dsk/c0d0s1 /mnt

   If the root file system is on the first SMD, the command is

   mount /dev/dsk/cld0s1 /mnt

2. **cd** to **/mnt**. Now you are in your own root file system, rather than the RAM file system.

3. Use any of the programs in the RAM file system's **/bin** or **/etc** to modify files in your root file system. Use **cpio** to copy a damaged file from backup.

To repair files in **/usr**,

1. Run **fsck** on the root file system and mount it as described above.

2. Run **fsck** on the **/usr** file system. If the **/usr** file system is on the first ST506, the command is

   fsck /dev/rdsk/c0d0s3

   If the **/usr** file system is on the first SMD, the command is

   fsck /dev/rdsk/c1d0s3

3. Mount the file system on **/usr**. If the **/usr** file system is on the first ST506, the command is

   mount /dev/dsk/c0d0s3 /mnt/usr

   If the **/usr** file system is on the first SMD, the command is

   mount /dev/dsk/c1d0s3 /mnt/usr

4. **cd** to **/mnt/usr**.

5. Use programs in the RAM file system's **/bin** and **/etc** to repair or restore files in **/usr**.

To use a program not in the RAM file system, use the **chroot** command. (The full pathname is **/mnt/etc/chroot**, and you change root to **/mnt**.) To use **vi**, for example, to delete root's encrypted password from **/etc/passwd**,

1. Mount **c0d0s1** and **c0d0s3**, or **c1d0s1** and **c1d0s3**, as described above.

2. Set and export the TERM variable; for example,

   TERM=pt; export TERM

3. Change root and edit the password file as follows:

/mnt/etc/chroot /mnt /usr/bin/vi /etc/passwd

RECOVERING FROM A CORRUPTED EEPROM

The paragraphs below document how to recover from a situation in which the EEPROM has become corrupted and you are no longer able to boot from your normal SMD boot device. The procedure outlined here assumes that the SMD that will boot **unix** has been previously initialized and has a valid bad block table in slice 0.

Follow the steps below to restore the EEPROM information:

1. Put the maintenance tape in the drive and press the **Reset** button.

2. When prompted if you want to add swap, answer **no**. You will get a warning message about not running **fsck**. Ignore the warning, since you will not be running **fsck** now and you can do without swap space until later in the procedure.

3. When a shell prompt appears, run **ed** on **/etc/system**. The **/etc/system** that appears on your screen is the **/etc/system** provided on the maintenance tape, not the **/etc/system** you previously configured.

   At the bottom of the file, there is a standard entry for an Interphase disk controller. Note that the slot number specified is **Ø**, the address is **C1ØØØ2ØØ**, and there is no initialization routine named.

   Edit this line if you are using a different address or a different slot number, but do not specify an initialization routine: you will continue to be booted from the maintenance tape until the end of the procedure.

   Write the **/etc/system** file when you have finished editing the VMESLOTS line. Since the RAM disk is short of swap space, you will get a warning when you first attempt to write the file. Ignore the "?" warning, and attempt to write again. After a couple of attempts, the write will succeed and you will be able to exit the editor.

4. Now run **ldeeprom**.

   ```
   # ldeeprom
   ```

   ldeeprom reads **/etc/system** and initializes the EEPROM with a checksum and the address of the controller. Wait for the shell prompt to return. The execution of the **ldeeprom** program can take up to about 1Ø minutes. (The front panel lights will blink rapidly while **ldeeprom** executes!)

5. When you have a shell prompt, run **hinv -p** (hardware inventory command) to verify that the system recognizes the presence of an SMD controller in the VME slot.

   The system message should tell you that there is a controller 1, drive Ø (this is the SMD drive) and a controller 2, drive Ø (this is the RAM disk).

6. Add some swap space from the SMD and delete the swap space you were using on the RAM disk, as follows:

   ```
   # swap -a /dev/dsk/cldØs2
   # swap -d /dev/dsk/c2dØs2
   ```

   Do a **swap -l** to verify that you now have your normal swap partition added as swap space. The system message should tell you that your path is **/dev/dsk/cldØs2**, that the device is **1,2** and that you have a large number of blocks available as swap.

7. Now run **fsck** as follows:

   ```
   # fsck
   ```

8. Mount your normal root file system on **/mnt**:

   ```
   # mount /dev/dsk/cldØs1 /mnt
   ```

9. Do a **chroot** to **/mnt**:

   ```
   # /mnt/etc/chroot /mnt /bin/sh
   ```

   Now you are in your own root file system rather than the root file system on the RAM disk.

10. Run **fsck** on the **/usr** file system:

    # fsck /dev/rdsk/cld0s3

11. Mount the file system on **/usr**:

    # mount /dev/dsk/cld0s3 /usr

12. Use **more** or some other text utility to examine **/etc/system** and verify its contents. (This is the old **/etc/system**, not the **/etc/system** you edited in step 3.) Edit the file if you need to make changes.

13. Reload the EEPROM with the contents of the **/etc/system** file you have just verified:

    # /etc/lddrv/ldeeprom

14. When **ldeeprom** has finished executing, do a **sync**, remove the maintenance tape, and reboot. You should now be able to boot successfully from the SMD.

USING THE CTIX SYSTEM ACTIVITY PACKAGE

OVERVIEW

The paragraphs below describe the design and implementation of the CTIX System Activity Package. The CTIX operating system contains a number of counters that are incremented as various system actions occur. The System Activity Package reports CTIX system-wide measurements including the following:

- central processing unit (CPU) utilization
- disk and tape input/output (I/O) activities

- terminal device activity
- buffer usage, system calls
- system switching and swapping
- file-access activity
- queue activity
- message and semaphore activities

The package provides four commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The five functions of the activity package are:

sar(1) command

This allows a user to generate system activity reports in real-time and to save system activities in a file for later usage.

sag(1G) command

This displays system activity in a graphical form.

sadp(1M) command

This samples disk activity once every second during a specified time interval and reports disk usage and seek distance in either tabular or histogram form.

timex(1)

This is a modified **time**(1) command that times a command and also optionally reports concurrent system activity and process accounting activity.

system activity daily reports

These procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The system activity information reported by this package is derived from a set of system counters located in the operating system kernel. These system counters are described in "System Activity Counters" below; the commands provided by this package are described in "System Activity Commands." The procedure for generating daily reports is explained in "Daily Report Generation."

NOTE

There is an administrative tool to start the system activity system and to turn it off. See the CTIX Administration Tools Manual for a description of the tool.

SYSTEM ACTIVITY COUNTERS

The CTIX operating system manages a number of counters that record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the **sysinfo** structure in **/usr/include/sys/sysinfo.h**. The system table

overflow counters are kept in the **syserr** structure. The device activity counters are extracted from the device status tables. The I/O activity of the following devices is recorded: GDØ-15, RS422, QICØ.

The following paragraphs describe the system activity counters sampled by the System Activity Package.

Cpu time counters

There are four time counters that may be incremented at each clock interrupt 6Ø times per second. According to the mode the CPU is in at the interrupt (idle, user, kernel, and wait for I/O completion), exactly one of the **cpu[]** counters is incremented.

lread and lwrite

The **lread** and **lwrite** counters are used to count logical read and write requests issued by the system to block devices.

bread and bwrite

The **bread** and **bwrite** counters are used to count the number of times the data is transferred between the system buffers and the block devices. The actual I/Os are triggered by logical I/Os that cannot be satisfied by the current contents of the buffers. The ratio of block I/O to logical I/O is a common measure of the effectiveness of the system buffering.

phread and phwrite

The **phread** and **phwrite** counters count read and write requests issued by the system to raw devices.

swapin and swapout

The **swapin** and **swapout** counters are incremented for each system request initiating a transfer from or to the swap device. Each request brings in or out at least one page (4K) of process information. The amount of data transferred between the swap device and memory are measured in 512-byte blocks and counted by **bswapin** and **bswapout.**

pswitch and syscall

These counters are related to the management of multiprogramming. **syscall** is incremented every time a system call is invoked. The numbers of invocations of **read**(2), **write**(2), **fork**(2), and **exec**(2) system calls are kept in counters **sysread, syswrite, sysfork,** and **sysexec,** respectively. **pswitch** counts the times the switcher was invoked, which occurs when under the following conditions:

1. A system call resulted in a road block.

2. An interrupt occurred, which results in the awakening of a higher priority process.

3. A 1-second clock interrupt occurs.

iget, namei, and dirblk

These counters apply to file-access operations. **iget** and **namei**, in particular, are the names of CTIX operating system routines. The counters record the number of times the respective routines are called. **namei** is the routine that performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special path. **iget** is a routine called to locate the i-node entry of a file (i-number). It first searches the in-core i-node table. If the i-node entry is not in the table, routine **iget** will get the i-node from the file system where the file resides and make an entry in the in-core i-node table for the file. **iget** returns a pointer to this entry. **namei** calls **iget**, but other file access routines also call **iget**. Therefore, counter **iget** is always greater than counter **namei**.

Counter **dirblk** records the number of directory block reads issued by the system. It is noted that the directory blocks read divided by the number of **namei** calls estimates the average path length of files.

runque, runocc, swpque, and swpocc

These counters are used to record queue activities. They are implemented in the **clock.c** routine. At every 1-second interval, the clock routine examines the process table to see if any processes are in core and in ready state. If so, the counter **runocc** is incremented and the number of such processes is added to counter **runque**. While examining the process table, the clock routine also checks if any processes in the swap device are in ready state. The counter **swpocc** is incremented if the swap queue is occupied, and the number of processes in the swap queue is added to counter **swpque**.

readch and writech

The **readch** and **writech** counters record the total number of bytes (characters) transferred by the **read** and **write** system calls, respectively.

Monitoring terminal device activities

There are six counters monitoring terminal device activities. **rcvint**, **xbmtint** and **mdmint** are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. **rawch**, **canch**, and **outch** count number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in "cooked" mode, such as terminals, are counted in both **rawch** and (as edited) in **canch**; but characters from raw devices, such as communication processors, are counted only in **rawch**.

msg and sema counters

These counters record message sending and receiving activities and semaphore operations, respectively.

Monitoring I/O activities

To monitor the I/O activity for a disk or tape device, four counters are kept for each disk or tape drive in the device status table. Counter **io_ops** is incremented when an I/O operation has occurred on the device. It includes block I/O, swap I/O, and physical I/O. **io_bcnt** counts the amount of data transferred between the device and memory in 512-byte units. **io_act** and **io_resp** measure the active time and response time of a device in time ticks summed over all I/O requests that have completed for each device. The device

active time includes the device seeking, rotating, and data transferring times, while the response time of an I/O operation is from the time the I/O request is queued to the device to the time when the I/O completes.

inodeovf, fileovf, textovf, and **procovf**

These counters are extracted from the **syserr** structure. When an overflow occurs in any of the inode, file, text, and process tables, the corresponding overflow counter is incremented.

SYSTEM ACTIVITY COMMANDS

The System Activity Package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity in the following instances:

- a controlled standalone test of a large system
- an uncontrolled run of a program to observe the operating environment
- normal production operation

The **sar** and **sag** commands permit a user to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command. The **sadp** command allows a user to establish a sampling period during which access location and seek distance on specified disks are recorded and later displayed as a tabular summary or as a histogram.

The sar Command

The **sar** command can be used in the following two ways:

- When the frequency arguments **t** and **n** are specified, it invokes the data collection program **sadc** to sample the system activity counters in the operating system every **t** seconds for **n** intervals and generates system activity reports in real-time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. The format of the data file is shown in **sar**(1M). In addition to the system counters, a time stamp is also included, which gives the time the sample was taken.

- If no frequency arguments are supplied, **sar** generates system activity reports for a specified time interval from an existing data file that was created by **sar** at an earlier time.

A convenient use of **sar** is to run it as a background process, save its samples in a temporary file but sending its standard output to **/dev/null**. Then an experiment is conducted after which the system activity is extracted from the temporary file. The **sar**(1) manual entry describes the usage and lists various types of reports.

The sag Command

sag displays system activity data graphically. It relies on the data file produced by a prior run of **sar**, after which any column of data or the combination of columns of data of the **sar** report can be plotted. A fairly simple but powerful command syntax allows the specification of cross plots or time plots. Data items are selected using the **sar** column header names. The **sar**(1G) manual entry describes its options and usage. The system activity graphical program invokes **graphics**(1G) and **tplot**(1G) commands to have the graphical output displayed on any of the terminal types supported by **tplot**.

The timex Command

The **timex** command is an extension of the **time**(1) command. Without options, **timex** behaves like **time**. In addition to giving the time information, it can also print a system activity report and a process accounting report. For all the options available, refer to the manual entry **timex**(1). It should be emphasized that the **user** and **sys** times reported in the second and third lines are for the measured process itself including all its children while the remaining data (including the **cpu user** % and **cpu sys** %) are for the entire system.

While the normal use of **timex** will probably be to measure a single command, multiple commands can also be timed either by combining them in an executable file and timing it, or by typing

timex sh -c "cmd1; cmd2;...;**"**

This establishes the necessary parent-child relationships to correctly extract the user and system times consumed by cmd1, cmd2, ...(and the shell).

The sadp Command

sadp is a user level program that can be invoked independently by any user. It requires no storage or extra code in the operating system and allows the user to specify the disks to be monitored. The program is reawakened every second, reads system tables from **/dev/kmem**, and extracts the required information. Because of the 1-second sampling, only a small fraction of disk requests are observed; however, comparative studies have shown that the statistical determination of disk locality is adequate when sufficient samples are collected.

In the operating system, there is an **iobuf** for each disk drive. It contains two pointers which are head and tail of the I/O active queue for the device. The actual requests in the queue may be found in the following three buffer header pools:

- system buffer headers for block I/O requests
- physical buffer headers for physical I/O requests
- swap buffer headers for swap I/O

Each buffer header has a forward pointer that points to the next request in the I/O active queue and a backward pointer that points to the previous request.

sadp snapshots the **iobuf** of the monitored device and the three buffer header pools once every second during the monitoring period. It then traces the requests in the I/O queue, records the disk access location, and seeks distance in buckets of 8-cylinder increments. At the end of the monitoring period, it prints out the sampled data. The output of **sadp** can be used to balance the load among disk drives and to rearrange the layout of a particular disk pack. This command is described in manual entry **sadp**(1M).

DAILY REPORT GENERATOR

The previous paragraphs described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. The following paragraphs describe the steps that you may follow to automatically produce a standard daily report of system activity.

Facilities

sadc The executable module of **sadc.c** which reads system counters from **/dev/kmem** and records them to a file. In addition to the file argument, two frequency arguments are usually specified to indicate the sampling interval and number of samples to be taken. If no frequency arguments are given, it writes a dummy record in the file to indicate a system restart.

sa1 The shell procedure that invokes **sadc** to write system counters in the daily data file **/usr/adm/sadd** where **dd** represents the day of the month. It may be invoked with sampling interval and iterations as arguments.

sa2 The shell procedure that invokes the **sar** command to generate daily report **/usr/adm/sa/sardd** from the daily data file **/usr/adm/sa/sadd**. It also removes daily data files and report files after seven days. The starting and ending times and all report options of **sar** are applicable to **sa2**.

Suggested Operational Setup

It is suggested that **cron**(1M) control the normal data collection and report generation operations. For example, the following sample entries in **/usr/spool/cron/crontab/sys** would cause the data collection program **sadc** to be invoked every hour on the hour:

```
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
0 * * * 0,6 /usr/lib/sa1 &
0 18-7 * * 1-5 /usr/lib/sa/sa1 &
```

Moreover, depending on the arguments presented, **sadc** writes data to the data file one to three times at every 20 minutes. Therefore, under the control of **cron**(1M), the data file is written every 20 minutes between 8:00 and 18:00 on weekdays and hourly at other times.

Note that data samples are taken more frequently during prime time on weekdays to make them available for a finer and more detailed graphical display. It is suggested that **sal** be invoked hourly rather than once every day; this ensures that if the system crashes, data collection will be resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** with no frequency arguments within **/etc/rc** when going to multiuser state:

```
/bin/su - sys  -c "/usr/lib/sa/sadc
/usr/adm/sa/sa`date +%d` &"
```

cron(1M) also controls the invocation of **sar** to generate the daily report via the shell procedure **sa2**. One may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if the following is an entry in **/usr/spool/cron/crontab/sys**, **cron** will execute the **sar** command to generate daily reports from the daily data file at 20:00 on weekdays:

```
0 20 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd
```

The daily report reports the CPU utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In the case of a shortage of disk space or for any other reason, these data files and report files can be removed by the superuser. The manual entry **sar**(1M) describes the daily report generation procedure.

ACCESS TO PERIPHERAL DEVICES

CTIX provides a standard way for programs to use peripheral devices. Each special file represents a particular way to access a particular peripheral. A special file appears in the file system (by convention, in **/dev**) and ordinary input/output operations on special files have standard meanings for the peripheral.

Special files are either block or character. Block special files identify kernel routines that are most efficient with input/output operations precisely 1024 bytes long. Character special files identify kernel routines that do not prefer any particular size operation. Some peripherals are represented by both block special files and character special files.

MightyFrame CTIX is normally set up with 32 special files for each disk drive, providing a block special file and a character special file for each possible slice. The name of a disk's special files takes one of two forms:

/dev/dsk/cc#dd#ss#

/dev/rdsk/cc#dd#ss#

where

/dsk specifies that you are accessing the slice as a block special file.

/rdsk specifies that you are accessing the slice as a character special file ("raw" device).

c# is the disk controller: Ø for the first controller, 1 for the second controller, and so on. The first controller corresponds to the first entry in the kernel configuration file, the second to the second entry, and so on.

d# is the disk: Ø for the disk in the first disk drive position, 1 for the disk in the second disk drive position, and so on.

s# is the slice: Ø for the first (reserved) slice, 1 for the second slice, and so on; numbering is hexadecimal (A through F for slices 1Ø through 15).

If a disk has fewer than 16 slices, it is an error to use the special files for the nonexistent slices.

SECTORS AND BLOCKS

A block is the basic unit of disk input/output. There are two kinds of blocks:

- Physical sector, a physical entity 512 bytes long. A disk drive's basic access to the disk reads or writes a physical sector.

- Logical block, a conceptual entity 1Ø24 bytes long. A CTIX input/output operation involves 1 or more (never a fraction) logical blocks. Using double-size logical blocks improves performance.

The utilities that initialize the disk, create the file systems, and report on disk sizes consider blocks synonymous with physical sectors. But most CTIX programs, including **fsck**, consider the basic unit to be the logical block. In the remainder of this appendix, a block is a logical block.

DIRECTORIES

A directory is simply a file that only the CTIX kernel can write to. Each directory entry consists of the file name and the file's i-number. A file can have more than one directory entry (link). The number of directory entries that refer to a file is that file's link count.

FILE SYSTEM FORMAT

A CTIX file system is a storage area (normally a disk slice) with the following structures:

- A block reserved for use in booting CTIX.
- The super block, containing data structures that describe the file system.
- The i-list, a sequence of records, called i-nodes, that describe CTIX files. The size of the i-list is fixed when the file system is created. Each i-node has an i-number that gives the i-node's place in the i-list. All file status information is in the i-node, as are the direct and indirect pointers to the file's data blocks.
- The free list, a linked list of blocks not used by any file and not contained in the bitmap. Each element of the free list is a block that contains pointers to 50 additional blocks.

- The bitmap, a data structure containing information about blocks not used by any file. Each bitmap can represent up to 16384 blocks. Each time a file system is created with **mkfs**, one bitmap is created. CTIX takes blocks from the free list and puts them in the bitmap. A block cannot be in both the free list and bitmap.

The CTIX program that creates these structures also creates a directory that is the first file on the file system. This directory is the root of the file system.

Two structures in an i-node are important for you, as system administrator, to understand: the link count and the disk addresses.

The link count is an integer value, set at Ø when the i-node is not in use. Creating a file sets the link count to 1. Each additional directory entry (link) for the file increments the link count; each removal of a directory entry decrements the link count. If the link count returns to Ø, the blocks for the file are returned to the free list, and the file is removed.

There are 13 disk addresses in the i-node. The first 1Ø point to the first 1Ø blocks of the file (the direct blocks). If the file is more than 1Ø blocks long, the 11th address points to a block that has pointers to the next 256 blocks of the file (the indirect blocks). If the file is more than 266 blocks long, the 12th address points to a block that points to up to 256 blocks containing pointers to the next 65,536 blocks of the file (the double-indirect blocks). If 65,8Ø2 blocks is not enough, the 13th address provides access to triple-indirect blocks.

The Link Count and disk address data structures can become inconsistent through incomplete input/output operations, usually those caused by a power failure some other improper shutdown procedure. One of your responsibilities is to repair file system data structures using the maintenance programs described in Chapter 8, "Disks."

A mount places a file system on the CTIX file system hierarchy. A mount specifies an empty directory and the special file that holds the mountable file system. A mount informs CTIX that any reference to the specified directory is actually a reference to the root directory of the file system. The directory on which a file system is mounted must be on a mounted file system; however, the parent file system must be mounted first.

The root file system (the file system whose root is /) is, in effect, always mounted. It is the only file system that has no parent file system.

The term file system actually has two uses, both of them the same on CTIX as on UNIX System V. CTIX and UNIX documents refer to a file system both as the organization imposed on a single slice ("the file system mounted on **/a**") and the whole disk hierarchy of files ("the CTIX file system"). Context should make clear which is meant.

CAUSES OF FILE SYSTEM CORRUPTION

File system corruption is caused by incomplete or garbled input/output instructions, caused by any of the following circumstances:

- Improper shutdown. All input/output must be complete before the processor is halted. To ensure completeness of input/output, kill all user processes and perform two **syncs**. All these procedures are contained in the shell script **/etc/shutdown**.

- Use of a corrupt file system. This causes further errors because of the incorrect file system structures.

- Hardware failure.

fsck AND THE FILE SYSTEM

fsck detects errors in three areas:

- the superblock

- the i-nodes

- directory data

fsck checks the following in the superblock:

- File system size and i-list size. The file system must be bigger than the superblock plus the i-list. Must not exceed 65,534 i-nodes.

fsck relies heavily on these two data. Except to check that they are reasonable values, there is no way to confirm their correctness. All other checks depend on the correctness of the file system and i-list sizes.

- Free block list and bitmap. The first block in the free list is pointed to by the superblock. Each block in this list contains pointers to additional free blocks. Each block's count of pointed-to blocks must not be less than Ø or greater than 5Ø. Each block pointer must not point past the end of the file system or before the first data block. No block in the free list or bitmap can be in **fsck's** list of blocks claimed by the i-nodes.

  CTIX files may also have a bitmap of free blocks. This bitmap has a bit set for each block in a 16384 block section of the file system. The blocks in the bitmap are omitted from the free list. **fsck** treats blocks in the bitmap exactly as if they were in the free list.

  If **fsck** finds errors in the free list or bitmap, or if it cannot account for every block in the file system, it will request permission to reconstruct the free list or bitmap. The new free list or bitmap will include all blocks not claimed by any i-node. In the absence of any other serious errors, rebuilding the free list or bitmap is always safe.

- Free block count. If this count does not agree with the actual number of free blocks, **fsck** requests permission to reset the count.

- Free i-node count. If this count is not the same as the size of the i-list minus the number of i-nodes in use, **fsck** requests permission to reset the count.

fsck checks the following fields in each i-node:

- Format and type. These fields specify the kind of file (ordinary, directory, block special, character special) and the i-node status (allocated or unallocated). Invalid values indicate that bad data has been written into the i-list. **fsck** prompts for permission to clear the i-node, which is unavoidable.

- Link count. This value must equal the number of directories that actually list the i-node. An inconsistency here indicates a failure to update a directory or the i-node; this is always a minor error.

  If the link count of the i-node and the number of links are nonequal and both are nonzero, **fsck** requests permission to correct the i-node link count.

  If the i-node link count is nonzero and the actual link count is zero, **fsck** requests permission to provide a link in the file system's **lost+found** directory.

- Duplicate blocks. These are blocks claimed by more than one i-node. **fsck** spots duplicate blocks as it builds its list of allocated blocks; this condition requires a second pass of the i-list to find the first i-node. Then **fsck** suggests which i-node should be cleared, (usually the one with the earlier modify time).

A large number of duplicate blocks probably indicates that CTIX failed to physically write out a block of pointers to indirect blocks. **fsck** requests permission to clear both i-nodes.

- Bad blocks. These are blocks that cannot be found because their addresses are invalid.

  If an i-node has a large number of bad blocks, CTIX probably failed to write out a block of pointers to indirect blocks. **fsck** asks for permission to clear the i-node.

- File size. Two kinds of errors can appear here: block allocation inconsistency and improper directory size.

  fsck computes the number of blocks required to accommodate a file of the indicated size. If this value does not match the number of blocks the file actually has allocated, **fsck** prints a warning. Note that this condition may be the result of a program seeking past the end of a file before writing to the file, a perfectly valid action.

  If the file is a directory, the file size should be a multiple of 16. If it is not, **fsck** prints a warning but takes no action.

fsck looks for the following errors in directory data:

- Reference to unallocated i-nodes. This is probably the result of CTIX's failure to write out a modified i-node. **fsck** requests permission to remove the directory entry.

- Invalid i-number. This is probably the result of bad data output to the directory. **fsck** requests permission to remove the directory entry.

- Incorrect . and .. entries: . must be the first entry in the directory and have an i-number equal to the i-number for the directory itself; .. must be the second entry in the directory and be a link to the directory's parent directory. If these entries are incorrect, **fsck** asks for permission to correct them.

APPENDIX B: SYSTEM INITIALIZATION PROGRAMS AND ADMINISTRATIVE FILES

This appendix describes the system initialization process and administrative files:

- boot sequence
- start-up programs **init**, **getty** (and **uugetty**), and **login**
- configurable and nonconfigurable files used by, or called from, these start-up programs
- miscellaneous other administrative files

It is important for you to understand the system initialization process for several reasons: first, because there are files you must modify to add users and the terminals and modems they will be using; and second because, to a large extent, CTIX allows you to customize the whole process.

All of the programs referenced by this appendix are described in detail in Volume I of your CTIX operating system manual; the files are described in Volume 2, Section 4, of the manual.

THE BOOT SEQUENCE

Each time you turn on the power to or reset the MightyFrame system, you initiate the following sequence of events:

1. The hardware is initialized to power on state (68020 boot).

2. The MightyFrame begins running software from the boot prom. Hardware diagnostics (for example, memory tests) are performed; disks are recalibrated; the loader is loaded from tape, disks, or VME.

3. The loader begins execution. It searches for **/unix** on the tape, the first ST506 disk, and then the first SMD disk. It loads and executes **/unix**. The MightyFrame is now running UNIX.

4. **/unix** initializes the hardware. It allocates system tables and initializes them. It mounts the root file system. The CTIX kernel "handcrafts" (creates from bits it locates in system tables) process 0, the scheduler.

5. The CTIX scheduler forks the **init** process, which oversees and controls the rest of the process spawning required to complete the system start-up. The end result of **init**'s work as initial process spawner in a successful start-up is the following:

   - CTIX running in multiuser state

   - cleanup or removal of old or possibly invalid system files

- subsystems, like **cron** and **lpsched**, started
- loadable device drivers loaded
- outgoing terminal lines set correctly and users able to login

The rest of this appendix describes the details and implications of **init**'s process spawning.

init, getty (and uugetty), AND login

The start-up programs **init** and **getty** are invoked every time the system is reset; **uugetty** is a special form of the **getty** program for dial-in/dial-out modems. If specified in **/etc/inittab**, **getty** and **uugetty** are invoked every time the system is reset. **login** is the program that is invoked every time a user tries to login. Together they keep **/etc/wtmp**, an accounting file that is available to other processes on the system. The **who** program, in particular, uses data in **/etc/wtmp** to make information about current and expired processes, logged-in users, and so forth, available to all users.

init

The **init** program serves as both initial process spawner and general process spawner. As initial process spawner, **init**'s forks other processes, principally **gettys**, to initiate terminal lines and otherwise bring the system to a fully operational state. As general process spawner, **init** remains alive after CTIX is completely started and waits for events that cause it to awaken and take action: for example, to kill one of its direct descendents or to spawn an additional process.

init uses the file **/etc/inittab** as both a script and a data base for its actions. **/etc/inittab** defines run-levels for the processes **init** spawns: a run-level is a configuration that allows a select group of processes (and their descendents) to exist. Run-levels are thus defined by the set of entries in **/etc/inittab**, which consists of the processes for **init** to spawn and their specified values for "rstate" (system states during which the processes are allowed to exist). Once **init** has entered a particular run-level, it spawns only those processes allowed to run at that level.

init changes run-levels when a privileged user runs the **init** or **telinit** program (descendents of the original **init**) and specifies a different run-level: in that event, the original **init** awakens, scans its data base, kills all processes that are not defined to run at the new level, and spawns the processes that are defined to run at the new level.

There are eight run-levels available: **0-6** and **S** or **s**. All run-levels except **s (S)** are soft -- that is, completely configurable. However, it is recommended that you do not change the definitions of **2** (general multiuser state) or **6** (administrator mode prompting for root login to bring the system to single user mode).

If an entry in **/etc/inittab** has no value for "rstate," the process can exist in all run-levels. **s** or **S** is not a legal value for the rstate field.

As initial process spawner, **init** first looks for the entry **initdefault**. This entry specifies a numeric level (for example, **2**) for **init** to go to as the first run-level. (If there is no entry for **initdefault**, CTIX prompts for an initial run-level.)

Before **init** actually executes any of the scripts defined to run at the default run-level, it executes the scripts of the type boot, bootwait, and wait: in the file **/etc/inittab** as distributed, these are the scripts **/etc/bcheckrc**, **/etc/wtmpclean**, **/etc/brc**, **/etc/drvload**, and **/etc/rc**. (See "System Initialization Files" later in this appendix.)

When all initialization scripts have been executed, **init** forks the processes indicated for the default run-level. Normally, the default run-level is **2**, and **init** spawns the **getty** and **uugetty** processes on all the indicated tty lines (entries of type "respawn").

As general process spawner, **init** respawns continuing processes and manages the changing of run-levels. **init** scans its data base **/etc/inittab** at least once every five minutes. In addition to its periodic review of the data base, **init** reads the data base on these occasions:

- in boot state
- when a power failure occurs
- when a child process of **init** dies
- when it receives a user signal (invocation of **init** or **telinit**)

When you make a change to **init**'s data base **/etc/inittab**, the fastest way to have **init** act on your change is to run the **telinit** program. The following command instructs the original **init** to read **/etc/inittab**.

```
telinit q
```

getty and uugetty

getty is the program responsible for

- establishing communication with a terminal
- printing the login message on the terminal screen
- reading the user name when it is entered
- calling the **login** program

uugetty is a special form of the **getty** program that performs all of the above functions, but relinquishes control of the line to **init** when a dial-out connection is made by **uucico**, **cu**, or **ct** (**uucp**-related programs). **uugetty** allows dial-in/dial-out connections to be made on a line without having to banish **getty** when an outgoing connection is to be established. (This avoids the situation where multiple run-levels had to be specified in **inittab**, one-level to allow dial-in connections, the other level to allow dial-out connections.)

getty and **uugetty** are normally invoked by **init**: this occurs most commonly during system initialization or when a login session is terminated and the login shell dies. When invoked, **getty** or **uugetty** scans its data base, **/etc/gettydefs**, for information about how to set up the terminal line.

getty can take from one to six arguments (these are documented in **getty**(1M)). **uugetty** can take from one to seven arguments; these are documented in **uugetty**(1M). (Note that in the case of **uugetty**, if the modem is an intelligent modem, or if there is a **uugetty** at the other end of the line also, you must specify the **-r** argument to **uugetty**

in the **inittab** entry: this causes **uugetty** to wait to receive a character from the terminal that will get the login message, and thus prevents two **uugettys** from looping. Note also that the **-t** option to **uugetty** specifies a timeout and is recommended.)

An entry for **getty** or for **uugetty** in **/etc/inittab** that has a corresponding entry in **/etc/gettydefs** must have at least two arguments:

- The name of a tty line in **/dev**.
- A label (usually the speed, but it can be any string of characters) to match the tty line to the appropriate entry in **/etc/gettydefs**. The label in **/etc/inittab** must be identical to the label for the corresponding entry in **/etc/gettydefs**.

The most important information for **getty** or **uugetty** to obtain from **/etc/gettydefs** is the baud rate to use to attempt communication with the terminal. **getty** or **uugetty** sets the baud rate for the "first try" value, and if it encounters a **Break** character, tries the "next try" value. This search sequence makes it possible for you to construct a circular linked list of entries in **/etc/gettydefs**; the user selects the correct entry by pressing the **Break** key until a login message appears.

getty and **uugetty** set up the line in two stages:

- The initial stage uses the "initial options" to establish communication, print the login message and read the user name. **getty** or **uugetty** puts the terminal into "raw" mode during this stage. This allows **getty** or **uugetty** to interpret each character as it is entered and infer

certain things about the terminal. For example, if all upper case letters are entered by the user, **getty** or **uugetty** assumes the terminal is uppercase only and maps uppercase alphabetics to lowercase so that CTIX can understand the input. During this stage **getty** or **uugetty** responds to a **Break** character by trying the "next try" baud rate value.

- The final stage uses the "final options" to establish terminal options for the rest of the login session or until a different set of options is specified when the login shell is invoked. "Final options" is normally a more exhaustive list than initial options and includes options like software tabs, any character to restart output, and so forth.

Options are discussed in more detail in "**/etc/gettydefs**" later in this appendix.

Whenever you change a **gettydefs** entry, you should check the validity of the new entry by running **getty** with the **-c** option. For example, the following command finds and reports any errors in the **gettydefs** file.

```
/etc/getty -c /etc/gettydefs
```

(Note that **uugetty** can be invoked similarly with the **-c** option; however, **getty -c** serves the same purpose and can be used to check any **gettydefs** file.)

login

The **login** program is called by **getty** or **uugetty** with the user name as the only argument. **login** can also be called from the shell, as described in **login**(1). **login** performs the following functions, in this order:

1. It checks the password file **/etc/passwd** to determine if the user name is valid and if the user requires a password. (If the user name is invalid, **login** prompts for the password anyway; this is a security precaution.) If a password is required, it prompts the user, and then checks the password entry against the encrypted entry in **/etc/passwd**.

2. It does a **cd** to the user's home directory.

3. It modifies **/etc/utmp** for process accounting purposes.

4. It sets up a basic environment for the user, giving initial values to the HOME, PATH, LOGNAME, SHELL, and MAIL environment variables.

5. It execs the program specified in the last field of **/etc/passwd**, normally the login shell. The default login shell is the Bourne shell (**sh**), but if C shell (**csh**) is specified, the **login** program execs **csh**. The program can be one that does not result in a shell, for example, **tdl**, the terminal download program.

6. If the program is **sh**, the shell executes **/etc/profile**, which sets up a default environment that is common to all users whose login shell is **sh**. Then the shell executes the user's own profile, **/$HOME/.profile**, if there is such a file. A user's **.profile** sets up a specific environment for that user.

7. If the program is **csh**, the shell executes **/etc/cprofile**, which sets up a default environment that is common to all users whose login shell is **csh**. Then the shell executes the user's own **.login**, if there is one, and the user's own **.cshrc** file, if there is one. Normally, the **/$HOME/.login** file sets up the **csh history** mechanism and sets other specific environment variables; the **/$HOME/.cshrc** file normally specifies **csh** aliases.

CONFIGURABLE FILES INVOKED BY THE START-UP PROGRAMS

The configurable files referenced by the **init**, **getty** (and **uugetty**), and **login** programs are described below. For each file, the following information is presented:

- the file format, if there is a specific format
- the distributed version of the file, if there is one; this includes annotations regarding file contents and appropriate modifications
- guidelines for creating the file, if there is no distributed version of the file

/etc/inittab

This file provides the script and data base for **init**'s process spawning activities.

Each entry in **/etc/inittab** takes the form

id:rstate:action:process

where

id is a string of one to four characters that uniquely identifies an entry.

If the entry concerns a tty channel and an invocation of **/etc/getty** or **uugetty**, the id should be the tty number; for example, **ØØ2**. If the entry concerns a terminal that is to be active in administrator mode (run-level **6**), it must have a separate entry for administrator mode, and id must begin with "C"; for example, **CØØ2**. Thus, a terminal that is to be active in both multiuser and administrator modes has two **inittab** entries.

rstate is the run-level at which this process is allowed to exist. Values are **Ø** through **6**; **s** or **S** is not a valid value. Multiple run-levels may be specified for a given process. When **init** enters a new run-level, it (1) takes the indicated action for all processes defined to run at that level and (2) kills all processes that were running at the previous level but are not defined to run at the new level.

If an entry contains an empty rstate field, the process is allowed to exist at all run-levels.

Run-levels **2** and **6** are defined in the distributed version of **/etc/inittab** as normal multiuser and administrator mode, respectively. It is recommended that you do not change these definitions. No other run-levels are defined in **/etc/inittab** as distributed.

action instructs **init** what to do with respect to a specific process at a given run-level. action is one of 11 possible values; see **inittab**(4).

Processes that **init** invokes at system start-up (other than **gettys** or **uugettys**) are normally of the type **boot**, **bootwait**, or **wait**.

gettys and **uugettys** are of the type **off** (tells **init** not to invoke **/etc/getty** or **/usr/lib/uucp/uugetty**) or **respawn** (in the case of **getty**, tells **init** to invoke **/etc/getty** if the process does not yet exist, and to invoke it again when the process dies; in the case of **uugetty**, tells **init** to invoke **/usr/lib/uucp/uugetty** if the process does not exist and the line does not have an outgoing connection already established, and to invoke it again when the process dies unless the line is being used for a dial-out connection).

process is the command to be executed.

If process is an invocation of **/etc/getty** or **/usr/lib/uucp/uugetty**, the command must have the following two arguments in this order: the tty number (for example, **tty00**2) and a label to associate it with a definition in **/etc/gettydefs** (for example, **9600**). (The label is sometimes referred to as the "speed label", but it can be any string of characters.) Thus, all entries in **/etc/inittab** having the same label refer to the same unique definition in **/etc/gettydefs**.

If the entry defines a terminal in administrator mode, the label argument to **/etc/getty** must have a "C" in front of it; for example, **C9600**.

Figure B-1 presents the distributed version of **/etc/inittab**. Notice that only one tty line, **tty000**, has a **getty** running on it. Notice also that there is an entry at the end of the file for tty000 to be active in administrator mode.

When To Modify /etc/inittab

You modify **/etc/inittab** in the following circumstances:

- When you add terminals and modems to the system.
- When you want to make terminals active in administrator mode.
- When you remove terminals and modems from the system.
- When you attach a serial device other than a terminal or modem (say, a printer) to a port that has a **getty** on it.
- When you define new run-levels.

```
: The following states are used :-
:   s = single user
:   2 = multiple user
:   6 = administration console
:
:   To make a terminal eligible to be the system console in the
:           event of system file system failure, duplicate the
:           terminal entry and place a 'C' in front of the tag
:           and speed label and change run-level to state 6 thus:
:   000:2:respawn:/etc/getty tty000 9600
:   C000:6:respawn:/etc/getty tty000 C9600
:
(1) is:2:initdefault:
[bt::bootwait:rm -f /etc/mnttab > /dev/console #remove /etc/mnttab
bl::bootwait:/etc/bcheckrc > /dev/console 2>&1 #bootlog
(2) wt::bootwait:/etc/wtmpclean > /dev/console #wtmpclean
bc::bootwait:/etc/brc > /dev/console #bootrun command
drv::bootwait:/etc/drvload >/dev/console 2>&1 # loadable drivers
[rc::wait:/etc/rc > /dev/console 2>&1 #run com
 pf::powerfail:/etc/powerfail >/dev/console 2>&1 # power fail!

(3) [000:12345:respawn:/etc/getty tty000 9600
    [001:2:off:/etc/getty tty001 9600

    : 232 board #1 ports 2 - 11
    [002:2:off:/etc/getty tty002 9600
    003:2:off:/etc/getty tty003 9600
    004:2:off:/etc/getty tty004 9600
    005:2:off:/etc/getty tty005 9600
    006:2:off:/etc/getty tty006 9600
    007:2:off:/etc/getty tty007 9600
    008:2:off:/etc/getty tty008 9600
    009:2:off:/etc/getty tty009 9600
(4) 010:2:off:/etc/getty tty010 9600
    011:2:off:/etc/getty tty011 9600

    : 232 board #2 ports 12 - 21
    012:2:off:/etc/getty tty012 9600
    013:2:off:/etc/getty tty013 9600
    014:2:off:/etc/getty tty014 9600
    015:2:off:/etc/getty tty015 9600
    016:2:off:/etc/getty tty016 9600
    017:2:off:/etc/getty tty017 9600
    018:2:off:/etc/getty tty018 9600
    019:2:off:/etc/getty tty019 9600
    020:2:off:/etc/getty tty020 9600
    021:2:off:/etc/getty tty021 9600
```

(1) This specifies that the default run-level is 2.

(2) These scripts are executed every time the system is booted.

(3) These entries correspond to the two serial ports on the main CPU board; the **9600** label corresponds to a label in the **/etc/gettydefs** file.

(4) Change "off" to "respawn" to attach additional terminals to specific ports.

Figure B-1. The /etc/inittab File (Sheet 1 of 3)

```
: 232 board #3 ports 22 - 31
022:2:off:/etc/getty tty022 9600
023:2:off:/etc/getty tty023 9600
024:2:off:/etc/getty tty024 9600
025:2:off:/etc/getty tty025 9600
026:2:off:/etc/getty tty026 9600
027:2:off:/etc/getty tty027 9600
028:2:off:/etc/getty tty028 9600
029:2:off:/etc/getty tty029 9600
030:2:off:/etc/getty tty030 9600
031:2:off:/etc/getty tty031 9600

: 232 board #4 ports 32 - 41
032:2:off:/etc/getty tty032 9600
033:2:off:/etc/getty tty033 9600
034:2:off:/etc/getty tty034 9600
035:2:off:/etc/getty tty035 9600
036:2:off:/etc/getty tty036 9600
037:2:off:/etc/getty tty037 9600
038:2:off:/etc/getty tty038 9600
039:2:off:/etc/getty tty039 9600
040:2:off:/etc/getty tty040 9600
041:2:off:/etc/getty tty041 9600

: RS422 terminals
: line 0 - #'s 0 - 7
256:2:off:/etc/getty tty256 RS422
257:2:off:/etc/getty tty257 RS422
258:2:off:/etc/getty tty258 RS422
259:2:off:/etc/getty tty259 RS422
260:2:off:/etc/getty tty260 RS422
261:2:off:/etc/getty tty261 RS422
262:2:off:/etc/getty tty262 RS422
263:2:off:/etc/getty tty263 RS422

: line 1 - #'s 8 - 15
264:2:off:/etc/getty tty264 RS422
265:2:off:/etc/getty tty265 RS422
266:2:off:/etc/getty tty266 RS422
267:2:off:/etc/getty tty267 RS422
268:2:off:/etc/getty tty268 RS422
269:2:off:/etc/getty tty269 RS422
270:2:off:/etc/getty tty270 RS422
271:2:off:/etc/getty tty271 RS422

: line 2 - #'s 16 - 23
272:2:off:/etc/getty tty272 RS422
273:2:off:/etc/getty tty273 RS422
274:2:off:/etc/getty tty274 RS422
275:2:off:/etc/getty tty275 RS422
276:2:off:/etc/getty tty276 RS422
277:2:off:/etc/getty tty277 RS422
278:2:off:/etc/getty tty278 RS422
279:2:off:/etc/getty tty279 RS422

: line 3 - #'s 24 - 31
280:2:off:/etc/getty tty280 RS422
281:2:off:/etc/getty tty281 RS422
282:2:off:/etc/getty tty282 RS422
283:2:off:/etc/getty tty283 RS422
284:2:off:/etc/getty tty284 RS422
285:2:off:/etc/getty tty285 RS422
286:2:off:/etc/getty tty286 RS422
287:2:off:/etc/getty tty287 RS422

: administrator consoles
C000:6:respawn:/etc/getty tty000 C9600
```

(5) (6) (7)

620-016

Figure B-1. The /etc/inittab File (Sheet 2 of 3)

⑤ RS-422 terminal entries begin here: change "off" to "respawn" for the number of terminals connected to a particular channel. For example, if your are using five terminals on line 0, change "off" to "respawn" for 256-260. Refer also to the description of **/etc/system** for information about configuring RS-422 lines and drops.

⑥ This entry makes tty000 eligible for use in administrator mode. Note that both the ID field and the label are prefixed by "C".

⑦ Add additional entries for other terminals to become active in administrator mode here; if there are to be RS-422 terminals active in administrator mode, make all RS-422 terminals eligible and hard-code the RS-422 cluster driver into the CTIX kernel. (See Chapter 7, "CTIX Device Drivers.")

Figure B-1. The /etc/inittab File (Sheet 3 of 3)

/etc/gettydefs

/etc/gettydefs is the data base for the **getty** and **uugetty** programs.

Each entry in **/etc/gettydefs** takes the form

label#ioptions#foptions#message#next

where

label — identifies the entry. The only strict rule is that label be unique in the file. By convention, a multiuser mode entry is labeled with its baud rate, and an administrator mode entry is labeled with **C** followed by the baud rate.

getty and **uugetty** match entries in **/etc/inittab** to entries in **/etc/gettydefs** by comparing the labels. Therefore, to associate a terminal or modem with a definition in the **/etc/gettydefs** file, make sure the labels are identical.

ioptions — is a list of communications options for **getty** or **uugetty** to apply when it first opens the terminal. Specify options with the symbolic constants described under **termio**(7) in your CTIX operating system manual. Since the terminal is put in "raw" mode when first opened, many of the possible options are ignored. (These options are not ignored: in c_iflag, ICRNL and IXON; all c_cflag; all c_oflag; in c_lflag, only ECHO, ECHONL, and NOFLSH options.) Symbolic constants are separated from each other by spaces or tabs.

foptions is a list of communications options for **getty** or **uugetty** to apply before calling **login** (that is, after a user first enters a login name). foptions contains the same kind of information as ioptions, although foptions is usually a more exhaustive list.

message is text to print when the terminal is first opened. The text should end with "login:".

next indicates another entry to use if **getty** or **uugetty** receives a break while it is using this entry. If you do not know one or more of a terminal's communications options in advance (most often the speed), use the next fields to form a circular linked list of entries. A user can then select the right entry by pressing the **Break** key until a login message appears.

To include nongraphic characters in the entry, use one of the following sequences:

\n	newline
\t	tab
\v	vertical tab (Control-K)
\b	backspace
\r	carriage return
\f	form feed
****xxx	where xxx is a 1- to 3-digit octal number

A backslash (\) followed by any character not mentioned above stands for the second character. Thus you enter \\ to get a \.

There should be only one difference between a multiuser mode entry and the corresponding administrator mode entry: The administrator mode entry should have a message that reminds the user that the system is in administrator mode.

Always check the correctness of the new **/etc/gettydefs** after modifying it. The following command finds errors:

```
/etc/getty -c /etc/gettydefs
```

As distributed, **/etc/gettydefs** contains the following six entries:

- A **300** entry, which defines communications options suitable for an RS-232-C 300-baud terminal and a multiuser mode login message; its next field is **1200**, enabling a user to get the 1200 baud setting by pressing the **Break** key.

- A **1200** entry, which defines communications options suitable for an RS-232-C 1200-baud terminal and a multiuser mode login message; its next field is **300**, enabling a user to get the 300 baud setting by pressing the **Break** key.

- A **9600** entry, which defines communications options suitable for an RS-232-C 9600-baud terminal and a multiuser mode login message.

- A **C9600** entry, which defines communications options suitable for an RS-232-C 9600-baud terminal and an administrator mode login message.

- An **RS422** entry, which defines communications options suitable for an RS-422 terminal and a multiuser mode login message.

- A **CRS422** entry, which defines communications options suitable for an RS-422 terminal and an administrator mode login message.

When To Modify /etc/gettydefs

You modify **/etc/gettydefs** in the following circumstances:

- When you connect a terminal or modem that requires a different set of communications options from those contained in the distributed file's definitions.
- When you want to change the login message.

Figure B-2 presents the **/etc/gettydefs** file, as distributed.

```
300# B300 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXON
OPOST ONLCR B300 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY TAB
3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\nlo
gin: #1200

1200# B1200 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B1200 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\
nlogin: #300

9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IXO
N OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
TAB3 #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\
nlogin: #9600

RS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL I
XON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO ECH
OE #MightyFrame CTIX (tm: Convergent Technologies) User Mode\n\nl
ogin: #RS422

C9600# B9600 CLOCAL IXON ECHO OPOST ONLCR # BRKINT HUPCL ICRNL IX
ON OPOST ONLCR B9600 CS8 CREAD ISIG ICANON ECHO ECHOE ECHOK IXANY
 TAB3 #MightyFrame CTIX (tm: Convergent Technologies)\n********\n
ADMIN MODE\n**********\nlogin: #C9600

CRS422# B9600 CLOCAL IXON ECHO OPOST ONLCR # IGNBRK IGNPAR ICRNL
IXON IXANY OPOST ONLCR B9600 CS8 CREAD CLOCAL ISIG ICANON ECHO EC
HOE #MightyFrame CTIX (tm: Convergent Technologies)\n********\nAD
MIN MODE\n**********\nlogin: #CRS422
```

620-017

Figure B-2. /etc/gettydefs

/etc/bcheckrc

/etc/bcheckrc is called from **/etc/inittab** and is executed every time the system is booted.

The purpose of **/etc/bcheckrc** is to check the date and to check file systems by running **fsck**.

By default, **/etc/bcheckrc** performs these checks automatically and without your intervention. If you want to run **fsck** interactively each time you reboot the system, change the line

```
CONSOLE=ABSENT
```

to

```
CONSOLE=PRESENT
```

If you make this change to **/etc/bcheckrc**, you will be prompted to (1) verify that the system date is correct, or (2) to enter the correct date if it is not correct. You will also be prompted to answer whether you want to check the file systems; if so, the system invokes **fsck** in a manner that requires your intervention if a file system needs repair.

It is recommended that you always check the file systems when you reboot your MightyFrame, whether you do it manually, by specifying CONSOLE=PRESENT, or whether you let the system do it automatically.

If you allow the MightyFrame system to do its own check of the date and of the file systems and it encounters inconsistencies in a file system, the MightyFrame enters administrator mode (run-level **6**), and prompts you to run **fsck** manually.

When To Modify /etc/bcheckrc

You modify **/etc/bcheckrc** in the following circumstances:

- If you want to interactively check the date and the file systems every time you reboot the MightyFrame.
- If you want to perform any other kind of system check before file systems are mounted.

Figure B-3 presents the **/etc/bcheckrc** file, as distributed.

```
# ***** This file has those commands necessary to check the file
# system, date, and anything else that should be done before
# mounting the file systems.
# NOTE: this script does not have any implicit input device!

TZ=`cat /etc/TZ`
PATH=:/bin:/etc:/usr/bin:/usr/local/bin
HOME=/
export HOME PATH TZ

# If you want to run date and the query for fsck interactively
# at boot then place CONSOLE=PRESENT on the next line, else if
# you just want to run fsck without the query, place
# CONSOLE=ABSENT on the next line.

(1) CONSOLE=ABSENT

# set current date
date -

if [ $CONSOLE = PRESENT ]
then
    # this will link console to /dev/syscon
    IOCTL=`/etc/conlocate -r`
    export IOCTL
    (
    stty $IOCTL

    # **** Check date

    while :
    do
            echo "Is the date `date` correct? (y or n) \c"
            read reply
            if
                    [ "$reply" = y ]
            then
                    break
            else
                    echo "Enter the correct date (mmddhhmmyy): \c"
                    read reply
                    date "$reply"
            fi
    done

    # ***** Auto check, if necessary

    while :
    do
            echo "Do you want to check the file systems? (y or n) \c"
            read reply
            case "$reply" in
                    y )
                            ;;

                    n* )
                            break
                            ;;

                    * )
                            echo "Invalid input. Try again."
                            continue
                            ;;
            esac
            trap "echo Interrupt" 2
            /etc/fsck -b -p -q || /bin/sh
            trap "" 2
            break
    done
    ) </dev/syscon >/dev/syscon 2>&1

else        # CONSOLE = ABSENT
(2) /etc/fsck -b -p -q || telinit 6
fi
# put out booted date
echo "System Booted on \c"
date
```

620-018

Figure B-3. /etc/bcheckrc (Sheet 1 of 2)

① Change ABSENT to PRESENT if you want to set the date and run **fsck** interactively with each system boot.

② If **fsck** is run automatically and a file system problem is detected, the system changes to run-level **6**.

Figure B-3. /etc/bcheckrc (Sheet 2 of 2)

/etc/wtmpclean

/etc/wtmpclean checks the size of the **/etc/wtmp** file; if it is too large (greater than 10 physical blocks), it moves **/etc/wtmp** to **/etc/OLDwtmp** and creates a new **wtmp** file.

When To Modify /etc/wtmpclean

You probably will never need to modify this file. The only change you might want to make is to the threshold at which the file should be cleaned out.

Figure B-4 presents the **/etc/wtmpclean** file, as distributed.

```
# The purpose of this file is to keep /etc/wtmp from getting to large.
if test -f /etc/wtmp
then
    # sz: size of /etc/wtmp in blocks.
    sz=`du -s /etc/wtmp | sed 's/    .*//'`
    test $sz -gt 10 && mv /etc/wtmp /etc/OLDwtmp && >/etc/wtmp
    chown adm /etc/wtmp
    chgrp adm /etc/wtmp
fi
```

620-019

① If **/etc/wtmp** is greater than 10 physical blocks, this moves **/etc/wtmp** to the OLD file and creates a new **/etc/wtmp** file.

Figure B-4. /etc/wtmpclean

/etc/brc

/etc/brc is called from **/etc/inittab** and is executed each time the system is booted. It is a short script that removes the binary file **/etc/mnttab**, if it exists, and replaces it with a new **/etc/mnttab**.

/etc/mnttab contains information about mounted file systems and is used by the **mount** command. After a system crash, the information in **/etc/mnttab** is unreliable and must be discarded. **/etc/brc** creates an **/etc/mnttab** that describes the root file system as the only mounted file system.

When To Modify /etc/brc

NEVER! This file should be left completely intact.

Figure B-5 presents the **/etc/brc** file, as distributed.

```
# ***** This command file's function is to remove the mnttab

if
    [ -r /etc/mnttab ]
then
    rm -f /etc/mnttab
fi
# put root into mnttab
devnm / | setmnt
```

620-020

Figure B-5. /etc/brc

/etc/drvload

/etc/drvload is called from **/etc/inittab** and is executed each time the system is booted. **/etc/drvload** loads all the loadable drivers you want to have running on your system. For example, to have the Kernel Debugger loaded each time you reboot, uncomment the line containing the command.

/etc/drvload is also a good place to put **swap -a** commands to increase your swap space. (See Chapter 8, "Disks.")

When To Modify /etc/drvload

Modify **/etc/drvload** in the following circumstances:

- When you want any loadable driver, except IOP and RS-422, to be loaded with each system boot.
- When you want to increase your swap space by invoking **swap -a**.

Figure B-6 presents the **/etc/drvload** file, as distributed.

```
#
#   called from /etc/inittab to load drivers & to add any swap devices
#    requires:
#    /etc/master      (with entries for loadable devices)
#    /etc/lddrv       (directory)
#    /etc/lddrv/lddrv
#    /etc/lddrv/mkifile
#    .o files for loadable drivers in /etc/lddrv
#
# set compiliation environment
CENVIRON="CPU=`/etc/hinv -c`"
export CENVIRON
> /etc/drvtab
cd /etc/lddrv
echo creating unix.sym
./mkifile /unix unix.sym

./mktunedrv && ./lddrv -av tunevar && echo "Tuneable Variables Driver Loaded"

#./lddrv -av prf && echo "Kernel Profiler Loaded"

(1) #./lddrv -av debugger && echo "Kernel Debugger Loaded"

./lddrv -av sxt && echo "Shell Layers Loaded"

./lddrv -av plp && echo "Parallel Printer Loaded"

# check if have VME before loading EEPROM and VME boards
if /etc/hinv vme
then
    # Load the eeprom before any VME boards
    if /etc/hinv eeprom
    then
            : eeprom is ok
    else
            if [ ! -d /usr/tmp ]
            then
                    mkdir /usr/tmp
            fi
            ./ldeeprom
    fi
    # now load in VME boards
fi

# load in network code after the VME board is loaded

#./lddrv -av pts && echo "Pseudo-terminals Loaded"

#./lddrv -av enp10 && echo "Sockets Loaded"

(2) if /etc/hinv 422
    then
        ./lddrv -av cluster tsp tsy tst && echo "Cluster Loaded"
    fi

(3) if /etc/hinv iop
    then
        ./lddrv -av tiop && { ./tiopld iop && echo "Terminal Accelerator Loaded" ; }
    fi

echo Drivers loaded

(4) # add any swap devices
    # swap -a /dev/dsk/cXdXsX
```

620-021

Figure B-6. /etc/drvload (Sheet 1 of 2)

① These lines may be commented or uncommented, depending on which of the drivers you want to load; this file shows "Shell Layers" and "Parallel Printer" loaded automatically every time the system is booted.

② These lines check for the presence of an RS-422 board; if a board is present, the driver is loaded.

③ These lines check for the presence of an IOP board; if a board is present, the Terminal Accelerator driver is loaded.

④ Edit this to add swap space.

Figure B-6. /etc/drvload (Sheet 2 of 2)

/etc/rc

/etc/rc is a general-purpose initialization script that is called from **/etc/inittab** when the system is booted and each time the system changes run-states. **/etc/rc** performs many functions, including the following:

- sets the system node name for UUCP
- updates the **/etc/mnttab** file
- mounts file systems if the system is changing from single-user to multiuser state
- moves the old **sulog** and **cronlog** files to OLD files (removing the old OLD files), and creates new **sulog** and **cronlog** files
- starts all system processes and demons; for example, system accounting, **lpsched**, and **cron**
- removes all temporary files
- sets parallel printer options

The **/etc/rc** file, as distributed, contains markers to show you where a currently commented out command to start a system process begins and ends. The beginning marker is of the form BXXX, and the ending marker is of the form EXXX, where XXX is the name of the process. To start a process, simply remove the comment signs (#) in the left margin between the markers. (Do not remove the comment sign in front of a marker.)

CAUTION

Changing or removing markers, removing comment signs from the left margin, or altering the form of a command to start a system process may interfere with the proper functioning of the administrative tools software.

When To Modify /etc/rc

Modify **/etc/rc** in the following instances:

- if you are not using the administrative tools software and you want to enable the process accounting subsystem, the system activity subsystem, or the **lp** scheduler **lpsched**
- when you configure your MightyFrame to be a node in the UUCP or Ethernet network
- when you define a new multiuser state in **/etc/inittab** that is not state 2 or state 3
- when you want to set the parallel printer port options from this script
- when you want to add scripts of your own: for example, to remove old log files that are not removed by any other automatic process, and so forth

Figure B-7 presents the **/etc/rc** file, as distributed.

620-022

① This sets and exports the time zone variable.

② This sets and exports a default path.

③ Edit this line if this MightyFrame is a UUCP or Ethernet node.

④ Leave this alone.

⑤ This gives information about the previous run-state.

Figure B-7. /etc/rc (Sheet 1 of 2)

⑥ If the last run-state was **S** and you are going to **2** or **3**, this mounts the file systems in **/etc/mountable**; if you define other multiuser states, you will have to modify this line.

⑦ Lockfile clean-up.

⑧ In the event of an abnormal shutdown, this sends mail to users telling them that there are copies of **ex** and **vi** buffers.

⑨ Uncomment these lines to start system accounting system.

⑩ Uncomment to start system activity programs.

⑪ This begins new **sulog**.

⑫ This begins new **cronlog**.

⑬ This starts **cron**.

⑭ This executes **sync** every 30 seconds.

⑮ This deletes tmp files.

⑯ These are the **lp** spooler lines: uncomment these when you configure **lp**.

⑰ These are the **lpr** spooler lines: uncomment these when you configure **lpr**.

Note that only one spooler can be used.

⑱ Uncomment and edit this to set printer attributes on the parallel printer channel.

Figure B-7. /etc/rc (Sheet 2 of 2)

/etc/powerfail

/etc/powerfail, called from **/etc/inittab**, is executed whenever the MightyFrame system receives a powerfail signal.

/etc/powerfail runs the **halt** program, which flushes system buffers and brings the system to a graceful shutdown.

When To Modify /etc/powerfail

You will probably never have a reason to modify **/etc/powerfail**. You may, however, want to modify **halt** and run that version at **powerfail**.

Figure B-8 presents the **/etc/powerfail** file, as distributed.

```
# power hit - we have ~ 2 minutes to stop system
halt </dev/syscon >/dev/syscon 2>&1
```

620-022

Figure B-8. /etc/powerfail

/etc/passwd

/etc/passwd, used by **login** and other programs, is a data base containing information about valid login names.

A login name is a string of characters in the first field of the **/etc/passwd** file. Entering a login name after the login prompt in multiuser mode can ultimately cause the program specified in the last field of the password file to be executed; this program is normally a shell. A login name can be the name or initials of a user who is an actual person; or it can be the owner of system files and programs, for example, **root** and **sys**. Thus, some logins have special privilege: it is very important to password-protect these login names.

Chapter 9, "Adding and Supporting Users," documents how to run the **passwd** program to add new users and to password-protect system file owners.

Each entry in the password file is a line of the form

name:pass:uid:gid:fullname:home:program(shell)

where

name — is the user's login name.

pass — is user's encrypted password. Leave this field blank; it will be filled in when you run the **passwd** program.

uid — is the user's numeric user ID. This must be a decimal number greater than or equal to 100 and unique for each user.

gid — is the user's initial numeric group ID. To implement groups, see **group**(4). If the user is not associated with any group, set this field to 100.

fullname — is, by CTIX convention, the full name of the user. It can in fact hold any information; for example, name and office location. Several CTIX scripts, for example, **lp** interface programs, print the contents of this field on their output.

home — is the name of the user's home directory.

{program / (shell)} — is normally the full path name (not the command name) of the user's shell, the program that is executed when the user logs in. If this field is empty, **login** uses the Bourne shell, **/bin/sh**. program can also be a CTIX program (for example, **tdl**, the terminal download program) that does not produce a shell.

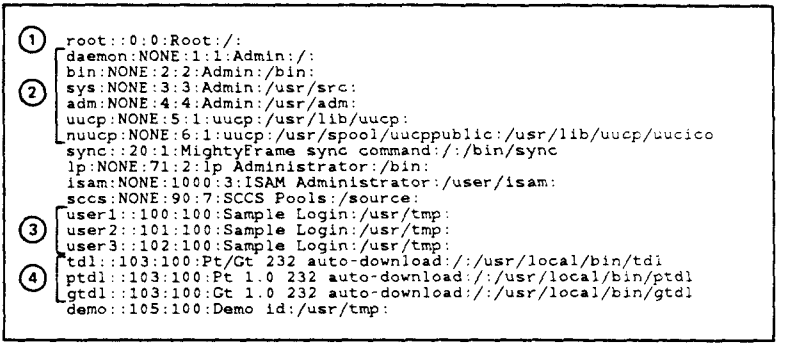
When To Modify /etc/passwd

Modify **/etc/passwd** in the following circumstances:

- when you add or remove a user; this includes users like **lp** and **bin**

- when you want to update information relating to any of the fields in the **passwd** file other than the password itself; for example, to assign a different home directory, or a different **gid**

Figure B-9 presents the **/etc/passwd** file, as distributed.



620-023

① Root needs a password!

② These, and other privileged users, need passwords if their logins are used: with a text editor, remove "NONE" from the password field, and run the **passwd** program.

③ Make entries for actual users.

④ These are logins for the terminal download programs.

Figure B-9. /etc/passwd

/etc/profile

/etc/profile, called by the shell during the login process, sets up a default environment for the user. This environment can be modified in the user's own **.profile** file.

As distributed, **/etc/profile** performs the following functions:

- Sets a default PATH and other environment variables, including the compilation variable CENVIRON.
- Invokes the **tset** command. **tset** sets the TERM environment variable according to entries in **/etc/ttytype**. (See Chapter 5, "Terminals and Modems.") If you do not use **tset** to set the TERM variable, set TERM explicitly, either in this file, or in the user's **.profile** file.
- Prints the message of the day, news, notice of mail, and so forth.
- Handles certain events relating to the system changing from multiuser to single-user mode. You should not modify commands involving state changes.

When To Modify /etc/profile

You will need to modify **/etc/profile** in the following circumstances:

- When you set the **umask** (CTIX file creation mode mask). **umask** masks out file creation privileges. With no mask, files and directories are created with a mode of 666 (rw-rw-rw-). When the **umask** is set to 22, files are created with a mode of 644 (rw-r--r--). This is recommended for better system security.

- If you invoke **stty** for all users: for example to set **erase**, **kill**, and **interrupt** characters. If users have different types of terminals, you may need to do this in **.profile** files.

- When you want to alter the default environment.

- When you want to enable the window manager. If all users have PTs and GTs, uncomment the lines relating to **wm** in the distributed file.

Figure B-10 presents **/etc/profile**, as distributed.

```
    trap '' 1 2 3
    SHELL=/bin/sh ; export SHELL
    TZ=`cat /etc/TZ`
(1) PATH=:/bin:/usr/bin:/usr/local/bin
    export TZ LOGNAME PATH
    # set compiliation environment
(2) CENVIRON="CPU=`/etc/hinv -c`"
    export CENVIRON
    # set file creation mask
    umask 022
    cmd=$0
    if [ -s /etc/utmp ]
    then
        set `who -r`
        if [ "$3" = "S" ]
        then
                if [ "$9" = "6" ]
                then
                        cmd=singleadmin
                else
                        cmd=singleuser
                fi
(3)     elif [ "$3" = "6" ]
        then
                cmd=administrator
        fi
    else
        echo /etc/utmp not readable
        if [ "$LOGNAME" != "root" ]
        then
                echo 'Sorry: system available only to root'
                sleep 5
                exit
        fi
    fi

    case $cmd in
    -sh | -rsh)
        trap : 1 2 3
    #   In the case of a remote login do not set TERM and print motd
        if [ "$TERM" = "" ]
        then
        #       The following line will set TERM (see ttytype)
(4)             TERM=`tset - -Q`
        #       The following may be used in place of above if
        #       /etc/ttytype is inaccurate
(5)     #       TERM=`tset - -Q '?dumb'`
(6)             export TERM
(7)             echo "       "NODE: `uname -n`, VERSION:
                CTIX`uname -v`, RELEASE: `uname -r`, DATE: `date +%D`
                cat /etc/motd
        fi
        trap '' 1 2 3
(8)     if mail -e
        then echo you have mail
        fi
```

620-024

Figure B-1Ø. /etc/profile (Sheet 1 of 3)

① This sets a default path for all users; if users have their own **.profile**, you do not need to modify the value for PATH in this file.

② Leave this alone unless you are cross-compiling: this variable contains the correct value for your hardware.

③ Do not modify this: if CTIX is in administrator mode, only root can log in.

④ This invokes the **tset** command to set the TERM variable; **tset** will know about this terminal only if you have made an entry for it in **/etc/ttytype**.

⑤ This is an alternate form of **tset**: queries user for terminal type.

⑥ This prints node message on terminal.

⑦ This prints message-of-the-day.

⑧ If a user has mail, this prints message "you have mail."

Figure B-10. /etc/profile (Sheet 2 of 3)

```
⑨      if [ $LOGNAME != root ]
       then
               news -n
       fi
       #To enable the window manager for PT's remove the
       #following comments:
       #if [ -r $HOME/.profile ]
       #then
       #       . $HOME/.profile
       #fi
       #trap 1 2 3
⑩      #exec /usr/local/bin/wm
       ;;
   -su)
       :
       ;;
   singleuser)
       sync
       PATH=/etc:/bin:/usr/bin:/usr/local/bin;export PATH
       killall 2>/dev/null
       mount|sed -n -e '/^\/ /d' -e 's/^.* on\(.*\) read.*/umount
               \1/p' | sh -
       mount|sed -n -e '/^\/ /d' -e 's/^.* on\(.*\) read.*/umount
               \1/p' | sh -
       echo "Entered Single User Mode on \c" >/dev/console
       date >/dev/console
       # get console record
⑪      cat /dev/console | tee -a /etc/log/confile
       sync;sync;sync
       echo "Ok To Stop Or Reset Processor"
       sync
       ;;
   administrator)
       telinit s || exit 1
       exit 0
       ;;
   singleadmin)
       # get console record so far
       cat /dev/console
       echo "File Systems Must Be Fixed Before Start-up Can Complete"
       echo "TYPE 'fsck -b' to check all file systems"
       echo "When Finished checking ALL file systems, TYPE reboot"
       ;;
   esac
   trap 1 2 3
```

⑨ If a user is not root, this prints news, if there is any.

⑩ This execs the window manager: uncomment this if all terminals are clustered PTs and GTs; alternatively, the window manager can be invoked from a user's own **.profile**.

⑪ Leave this alone: it involves events that occur when the system changes from multiuser to single-user mode (unmounts file systems, flushes buffers, allows only root to log in when the system is in state 6).

Figure B-10. /etc/profile (Sheet 3 of 3)

$HOME/.profile

A user's **.profile** file, if present, is called by the Bourne shell at the end of the login process, and after **/etc/profile**. Each user can create his or her own **.profile** to establish an individual environment. Typically, the following environment variables are set in **.profile**: PATH, MAIL, PS1 (sets primary prompt), TERM, LPDEST (**lp** default destination printer), EXINIT (sets **ex** and **vi** environment).

Below is an example of a **.profile** file for a user named "karen". Her search path begins at the current directory, then moves from her binary directory to **/bin**, and so forth. She specifies that her mail be put in **/usr/mail/karen** and that her default destination printer be "diablo". She does an **stty** to make the kill character, **Control-x**, and the interrupt character, **Control-**. Her TERM variable has been set in **/etc/profile**.

```
PATH=:$HOME/bin:/bin:/usr/bin:/usr/local/bin:/etc

MAIL=/usr/mail/karen

LPDEST=diablo

export PATH MAIL LPDEST

stty kill '^x' intr '^\'
```

/etc/ttytype

/etc/ttytype is used by the **tset** command to establish the value for the TERM environment variable. Each line in **/etc/ttytype** is of the form

terminal ttyXXX

where

terminal is the terminal type specified as a code in **/etc/termcap**. Entries in **/etc/termcap** consist of fields separated by vertical lines; any field in the correct **termcap** entry is an acceptable code to use in **/etc/ttytype**.

XXX is the terminal number.

When to Modify /etc/ttytype

You will need to modify **/etc/ttytype** when you add a terminal, unless it is a PT connected to Channel 0 on the RS-422 expansion board: **/etc/ttytype**, as distributed, contains entries for PTs on this channel.

Figure B-11 shows **/etc/ttytype**, as distributed.

```
dumb        tty000
dumb        tty001
pt          tty256
pt          tty257
pt          tty258
pt          tty259
pt          tty260
pt          tty261
pt          tty262
pt          tty263
pt          tty264
pt          tty265
pt          tty266
pt          tty267
pt          tty268
pt          tty269
pt          tty270
pt          tty271
pt          tty272
pt          tty273
pt          tty274
pt          tty275
pt          tty276
pt          tty277
pt          tty278
pt          tty279
pt          tty280
pt          tty281
pt          tty282
pt          tty283
pt          tty284
pt          tty285
pt          tty286
pt          tty287
pt          tty288
```

620-025

Figure B-11. /etc/ttytype

/usr/spool/cron/crontabs/*

The four **crontab** files, **root**, **sys**, **adm**, and **uucp** reside in **/usr/spool/cron/crontabs**.

Each entry in a **crontab** file, whether it is one of the administrative files or a user crontab file, consists of 6 fields, separated by spaces. These fields are

minute

hour

day of the month

month of the year

day of the week

program

See **cron**(1) for a specification of the values contained in each field.

cron looks at each of the **crontab** files once per minute and executes any program scheduled for the current time.

As distributed, **/usr/spool/cron/crontabs/root** does the following:

- runs the **errstop** program every day at 5:00 A.M. to stop error-logging; at that hour it also cleans out error files and the console log file, starts error-logging again, issues an error-report based on the previous day's file, and mails output to **adm**

- runs daily accounting at 2:00 A.M., if you uncomment the script at the marker
- removes old **ex** and **vi** buffer contents each morning
- three times a week, removes files named **a.out**, core dump files, **ed** buffers, and dead letter files not accessed for five days
- three times a week, removes **/tmp** and **/usr/tmp** files that have not been accessed for 10 days
- turns games off each weekday morning at 9:00
- turns games on each weekday evening at 6:00
- each morning at 1:00 A.M., looks at each user's home directory for a calendar file, and makes sure reminders get sent
- makes a date entry each 15 minutes in the console log file

When To Modify root's crontab File

Modify **root's crontab** in the following circumstances:

- when you want root to execute any program at a regular time
- when you use the accounting system: uncomment the relevant line
- when you want to change the hours that games are turned on and off
- when you want to change the frequency with which files named **a.out** or core dump files are removed: every 5 or 6 days may be too frequent

As distributed, **/usr/spool/cron/crontabs/sys** generates system activity data, if you uncomment the three relevant lines.

When To Modify sys's crontab File

Modify **sys's crontab** file when you use the system activity package: uncomment the only three lines in this file.

As distributed, **/usr/spool/cron/crontabs/adm** performs the following functions:

- processes daily accounting file and generates accounting reports, if you uncomment these lines
- processes monthly accounting file, if you uncomment this line
- processes system activity report according to the specified sampling mechanism, if you uncomment this line

When To Modify adm's crontab File

Modify **adm's crontab** file in the following circumstances:

- when you use the accounting system: uncomment the three relevant lines
- when you use the system activity package: uncomment the relevant line. You may also want to change the way **sar** does its sampling. See **sar**(1)

As distributed, **/usr/spool/cron/crontabs/uucp** runs the UUCP demons.

When To Modify uucp's crontab File

Modify **uucp's crontab** file under the following circumstances:

- when you do not want to poll other systems
- when you want to change the number of times per day **uudemon.admin** is run

```
#   @(#)root        1.1
0 5 * * 1 /etc/errstop ;/bin/mv /usr/adm/errfile /usr/adm/oerrfile; /bin/mv /etc
/log/confile /etc/log/oconfile; /usr/lib/errdemon ; /usr/bin/errpt /usr/adm/oerr
file | /bin/mail adm
# BACCT (marker for scripts)
#0 2 * * 1-6 /usr/lib/acct/dodisk
# EACCT
15 4 * * * find /usr/preserve -type f -mtime +7 -a -exec rm -f {} \;
0 1 * * 1,3,5 find / -atime +5 -a \( -name a.out -o -name core -o -name nohup.ou
t -o -name ed.hup -o -name dead.letter \) -a -exec rm {} \;
0 1 * * 1,3,5 find /tmp /usr/tmp -atime +10 -a -exec rm {} \;
0 9 * * 1-5 /usr/games/turnoff > /dev/null
0 18 * * 1-5 /usr/games/turnon > /dev/null
0 1 * * * /usr/bin/calendar -
0,15,30,45 * * * * /bin/date >/dev/console


#   @(#)sys 1.1
# BSAR (marker for scripts)
#0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3 &
#0 * * * 0,6 /usr/lib/sa/sa1 &
#0 18-7 * * 1-5 /usr/lib/sa/sa1 &
# ESAR


#   @(#)adm 1.1
# BACCT (marker for scripts)
#0 4 * * 1-6 /usr/lib/acct/runacct 2> /usr/adm/acct/nite/fd2log
#0 * * * * /usr/lib/acct/ckpacct
#15 5 1 * * /usr/lib/acct/monacct
# EACCT
# BSAR (marker for scripts)
#5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -ubcwydaqvm &
# ESAR


#   @(#)uucp        1.1
# BUUCP (marker for scripts)
54 * * * * /usr/lib/uucp/uudemon.poll > /dev/null 2>&1
56 * * * * /usr/lib/uucp/uudemon.hour > /dev/null 2>&1
0 4 * * * /usr/lib/uucp/uudemon.clean > /dev/null 2>&1
48 8,12,16 * * * /usr/lib/uucp/uudemon.admin > /dev/null 2>&1
# EUUCP
```

620-026

Figure B-12. The Four crontab Files in /usr/spool/cron/crontabs

/etc/system

The **/etc/system** system configuration file is used by several CTIX programs, including **mktunedrv** and **ldeeprom**. (**mktunedrv** and **ldeeprom** are called from **/etc/drvload**.)

The **/etc/system** file is a series of sections, each beginning with a section header. Each section header occupies a single line and consists of a word in capital letters, preceded by an exclamation point (!). The format of a section is specific to that section and depends on the format required by the program using the information. There are currently four sections:

- !FILENAMES defines the names of files used by the programs reading **/etc/system**. Currently, there are three file names defined:

  - PROM_IFILE is the name of the file to be used by the loader when assembling the contents of the EEPROM. Do not change this entry.

  - EEPROM_FILE is the name of the file to which EEPROM data will be written. If the file name is **/dev/vme/eeprom** (this is the default), the contents of this special file is output to the EEPROM. If the file name is something other than **/dev/vme/eeprom** (for example, **EEPROM.out**), the contents is not written directly to the EEPROM: instead, it is written to the specified file. (For example, this is useful for testing a new VME initialization routine.)

  - INIT_CFILE is the name of the tuneable variables driver file.

- !TUNEABLES specifies a set of system tuning parameters to be read by the tuneable variables driver **mktunedrv**. The variables are commented out in the distributed version of the file; to change a default value, uncomment the variable and change its value.

  - **cl_deflines** and **cl_defdrops** set the number of cluster lines and terminals per line respectively. (See Chapter 5, "Terminals and Modems," for more information on setting these variables.)

  - **cl_defdrive** specifies the full 16-bit device number of the controller and drive that contains the RS-422 download images. The device numbers are right-shifted four bits: thus cØdØ is Ø, cldØ is 16, cldl is 17, and so forth.

  - **net_mbsize**, **net_tcpdebug**, and **vt_defcnt** specify maximum buffer size, trace buffers, and number of virtual terminals for an Ethernet configuration.

  - **diriosize** specifies the I/O "kick-in" value (that is, at what point the system stops caching and begins doing direct I/O).

- !VMESLOTS contains data to be put into the EEPROM by **ldeeprom**. The data specifies slot, type, starting address, and address length of VMEbus devices. If the device is a boot device, its entry contains the name of an initialization

routine: for example, **initVs32** for an SMD. See Chapter 7, "CTIX Device Drivers," and **ldeeprom**(1M) for more information about the !VMESLOTS section and how to configure it.

- !VMECODE specifies the files containing the executable code of the initialization routines to be loaded into the EEPROM. This section is required only if a bootable initialization function is specified in the !VMESLOTS section.

When To Modify /etc/system

Modify **/etc/system** in the following circumstances:

- in a development environment, when you want to test a new EEPROM file
- when you want to tune the cluster lines
- when you want to tune the Ethernet configuration
- when you want to change the point at which caching stops and direct I/O takes over
- when you add an Ethernet controller board
- when you add an SMD controller board
- when you want to add a 1/2-inch tape controller
- when you want to make an SMD controller a boot device

Figure B-13 presents the **/etc/system** file, as distributed.

```
* The following "system" file is an example file - it should be personalized
* for your system.
*
* ---
*
* The format of this file is a series of sections each with a
* section keyword header (a word with a ! in front of it) followed
* by the section definitions.  The file is intended to be used by
* a number of programs to read system parameters. Programs should ignore
* the contents of all sections other then the ones headed by their
* specific section headers.
*
*
!FILENAMES
PROM_IFILE=/etc/lddrv/EEPROM.ifile
*To output to a file use
*EEPROM_FILE=EEPROM.out
*To output directly to vme use
EEPROM_FILE=/dev/vme/eeprom
*The name of the tuneable variables driver
INIT_CFILE=tunevar.c
!TUNEABLES
* NOTE: these are example/default values.
* NOTE2: tuneables MUST be set BEFORE the associated driver is loaded.
* disable the kernel debugger
*dsabldbg=1
* # of lines and drops, and default boot drive for 422 cluster comm
*cl_deflines=1
*cl_defdrops=8
*cl_defdrive=0
* # mbufs, # trace buffers, and # virtual terminals for networking
*net_mbsize=64*1024
*net_tcpdebug=0
*vt_defcnt=0
* direct i/o 'kick-in' value
*diriosz=2048
!VMESLOTS
* The following section describes the VME boards
*   (data to be stuffed into eeprom)
*
* The slot field is the slot in the vme bus.
* The type field is the board type:
*   1 - CMC Ethernet board
*   2 - Interphase SMD disk controller board
* The address field is the board's address.
* The length field is the address space size of the board.
* The optional initialization function name is an init function
*   called by the prom/loader (disks and ethernet boot only).
*   The initialization name only needs to be specified once.
*
* slot      type    address         length  [ Initialization-routine-name ]
*
* (one CMC ethernet controller - each one requires 128K of A24 space)
*0  1       C0DE0000        131072
*
* (Interphase disk controller)
*1  2       C1000000        512     loadvs32
*
* The following section describes the code to be stuffed into the eeprom.
*   It consists of a list of files
*
!VMECODE
* (VME disk controller) - commented out for now
*/etc/lddrv/DISKVS32.o
```

620-027

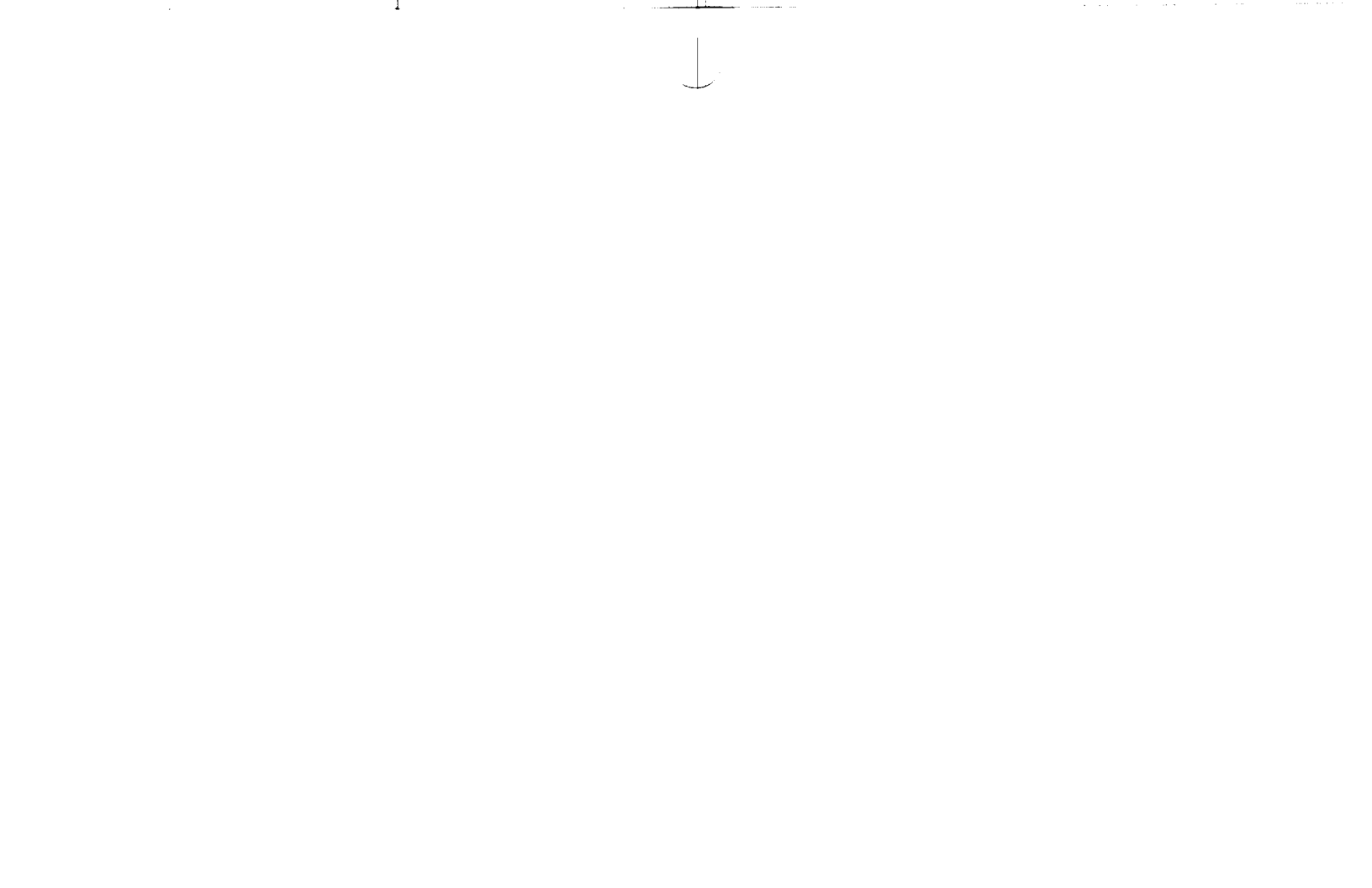
Figure B-13. /etc/system (Sheet 1 of 2)

① These specify the number of RS-422 lines and drops. Change the values of these parameters if you add or remove RS-422 terminals. Note that the tty numbers associated with a particular channel can vary according to the number of drops specified per channel. See Chapter 5, "Terminals and Modems."

② Make an entry in this section if you add an Ethernet controller.

③ Make an entry in this section if you add an SMD controller.

Figure B-13. /etc/system (Sheet 2 of 2)



This appendix explains the following information:

- which CTIX tuning parameters are autoconfigurable and what formulas CTIX uses to automatically configure these parameters
- how you can change autoconfigurable parameters to override the autoconfiguration process

AUTOCONFIGURATION PARAMETERS AND FORMULAS

By default, CTIX automatically configures **procs**, **regions**, **clists**, **inodes**, **files**, and **buffers**. It does so according to formulas based upon number of users, amount of physical memory on your system, number of available tty lines, and so forth.

Note that the number of users is defined by the version of CTIX you installed: either a 16-user or 32-user operating system. Note also that the number of available ttylines is calculated as the number of RS-232-C channels on your system plus the number of available RS-422 drops (as configured in **/etc/system**).

Autoconfiguration is turned on if the parameter has a value of **0** in the kernel configuration file used to make your version of the CTIX kernel. Autoconfiguration is turned off if the parameter has a value other than **0**. To turn off autoconfiguration and assign a fixed value to the parameter, you must remake the CTIX kernel, as described below.

Table C-1 shows the formulas CTIX uses to automatically configure a MightyFrame system.

Table C-1
AUTOCONFIGURATION FORMULAS

Parameter	Description	Formula
procs	maximum number of processes allowed on system	(maximum number of users * 150) / 16 and if odd, subtract one
regions	total number of regions in system	(maximum number of processes * 3) + 50 and if odd, subtract one
clists	small buffers to allocate to tty-like devices	(number of available ttylines * 6)+ 40
files	maximum open files allowed	(maximum number of processes * 2) + 100 and if odd, subtract one
inodes	maximum open inodes allowed	maximum open files allowed + (maximum number of users * 2) and if odd, subtract one
buffers	number of 1024-byte file system caching buffers	15% of real memory, minimum of 16 buffers

* indicates multiplication operator
/ indicates division operator

OVERRIDING AUTOCONFIGURATION

To assign a fixed value to one or more auto-configurable parameters, modify **dfile** and remake the CTIX kernel as follows:

1. **cd** to **/usr/sys/cf**.

2. Locate the version of **dfile** (kernel configuration file) that generated your current version of **/unix**. If you have SMDs on your system or you have for some other reason modified the CTIX kernel, look for a **dfile** in **/usr/sys/cf** that matches your current configuration. (See Chapter 7, "CTIX Device Drivers.") If you have not modified the kernel or added SMDs, use **/usr/sys/cf/dfile**.

3. Make a copy of the **dfile** you will modify. Give the copy a unique name to identify it (for example, **dfilemorebufs**, if you are increasing buffer size allocation).

4. Change the entry for one or more auto-configurable parameters in the **dfile** copy, assigning a fixed value to the parameter or parameters.

   For example, to specify 100 buffers, edit the copy of **dfile** so the following entry appears in the last section of the file:

   ```
   buffers    100
   ```

   (Make sure this is the only buffers entry in the file.)

5. Rebuild the CTIX kernel. Enter

   make VER=yourversion **DFILE=**dfilename

   where

   yourversion — is the CTIX version name you are creating: for example, **5.1Ømorebufs**.

   dfilename — is the filename of the **dfile** copy you have modified.

   The command

   make VER=5.1Ømorebufs DFILE=dfilemorebufs

   creates a file called **CTIX5.1Ømorebufs** in the directory **/usr/sys**. Always make a different version number from the standard CTIX release.

6. Bring the system to single-user mode and remount **/usr** (see Chapter 8).

7. Move **/unix** to **/OLDunix** so you have a backup of the current kernel:

   mv /unix /OLDunix

8. Make the new CTIX kernel the boot default by entering

   cp /usr/sys/CTIXyourversion **/**
   ln /CTIXyourversion **/unix**

   For example, if yourversion is **5.1Ømorebufs**, the command is

   cp /usr/sys/CTIX5.1Ømorebufs /
   ln /CTIX5.1Ømorebufs /unix

9. Sync and reboot the system. (Before you type the **reboot** command, always wait for the disks to become quiescent.) The command is

```
sync
sync
sync
reboot
```

1Ø. If the system does not reboot properly, or if any other catastrophic event (like a system crash) occurred after you moved **/unix**, boot the system with the maintenance tape, run **fsck**, move **/OLDunix** to **/unix**, find the problem, and start over.

administrator mode. Administrator mode refers to **init** run-level **6**. When CTIX detects a serious file system problem at system start-up, it automatically goes in administrator mode. When root (superuser) logs in to an eligible terminal, CTIX goes to single-user mode. See also **init**, **run-level**, and **single-user mode**.

administrative tools software. The CTIX administrative tools software package is a set of compiled programs and shell scripts that allow you to perform a number of system administrative functions without using shell commands.

autoconfiguration. Autoconfiguration is the process by which CTIX automatically configures procs, regions, clists, inodes, files, and buffers. Autoconfiguration can be overridden by specifying fixed values for these parameters in the kernel configuration file and remaking the CTIX kernel.

baud rate. Baud rate is the speed at which data is transmitted over certain types of serial channels; for example, RS-232-C. Baud rate is usually measured in bps (bits per second).

bitmap. In a file system, the bitmap is a data structure containing information about blocks not used by any file. Each bitmap can represent up to 16384 blocks. Each time a file system is created with **mkfs**, one bitmap is created: CTIX takes blocks from the free list and puts them in the bitmap. (A block cannot be in both the free list and bitmap.)

block. A block is the basic unit of disk I/O. There are two kinds of blocks: a physical sector (512 bytes long) and a logical block (1Ø24 bytes long). See also **sector** and **logical block**.

block special file. A block special file identifies kernel routines that are most efficient with I/O operations precisely 1Ø24 bytes long. **/dev/dsk/cØdØs1**, for example, is a special file name for the root file system accessed as a block device. See also **special file** and **character special** file.

channel. A channel is a data transmission path between the MightyFrame host and an auxiliary device, such as a terminal. On MightyFrame I/O boards, physical ports are labeled "Channel."

character special file. A character special file identifies kernel routines that do not prefer to access the device in 1Ø24-byte blocks. **/dev/rdsk/cØdØs1**, for example, is a special file name for the root file system accessed as a character (sometimes called "raw") device.

cluster line. A MightyFrame cluster line is an RS-422 channel. A maximum of eight terminals can be supported on a single cluster line.

console. A console is an imaginary terminal that receives important system status messages. On a MightyFrame computer system, console messages are sent to **/etc/log/confile.**

controller. A controller board is a hardware interface that manages an I/O device, such as a disk drive, Ethernet port, tape drive, and so forth.

CTIX. CTIX is an operating system derived from UNIX System V. CTIX runs on MegaFrame, MightyFrame, and MiniFrame computer systems.

CTIX modes. CTIX modes refer to categories of **init** run-states: single-user, multiuser, and administrator. See also **single-user mode**, **multiuser mode**, and **administrator mode**.

demon. A demon is a program that normally runs in the background: for example, **cron** (the clock demon), **uudemon.clean** (the demon that executes **uucleanup** and other programs for UUCP), and so forth.

device driver. A device driver is a software program that manages a peripheral device and handles communication between the device and the CTIX kernel. All CTIX device drivers except those that manage disks are "loadable." See also **loadable drivers** and **kernel**.

EEPROM. The MightyFrame EEPROM is the electrically erasable programmable read-only memory on the VME interface board. During a system boot, it supplies initialization parameters for MightyFrame hardware. The contents of the EEPROM can be altered by using the **ldeeprom** program. (**ldeeprom** generates a VME description file, and outputs it to the EEPROM.)

file system. A file system consists of CTIX files plus the data structures the CTIX kernel requires to support file uses. Normally all the slices on a disk except slice Ø (the reserved area) and any swap areas contain file systems. See also **slice**.

free list. A free list is a linked list of blocks not used by any file in the file system and not contained in the file system's bitmap. Each element of the free list is a block that contains pointers to 5Ø additional blocks. See also **file system** and **bitmap**.

getty. **getty** is the CTIX program that monitors channels for attempted logins. To "put a **getty** on a channel" means to cause the **/etc/getty** program to monitor the channel for attempted logins. To "take the **getty** off the channel" means to cause the channel to be free from interference by **/etc/getty** (for example, when you attach a printer to a channel). See also **uugetty.**

home directory. A user's home directory is the directory a user is pathed to when the user first logs in. A user's home directory is specified in **/etc/passwd.**

i-list. The i-list is a sequence of records, called i-nodes, that describe CTIX files. The size of the i-list is fixed when the file system is created.

init. **init** is the CTIX program that serves as a process spawner. As initial process spawner, **init's** role is to fork other processes, principally **gettys**, to initialize terminal lines and otherwise bring the system to a fully operational state. **init** remains alive after CTIX is completely started and waits for events that cause it to awaken and take action: for example, to kill one of its direct descendents or to spawn an additional process. **init** uses the file **/etc/inittab** as both a script and data base for its actions. **/etc/inittab** defines run-levels for the processes **init** spawns. See also **getty**, **uugetty**, and **run-level.**

i-node. An i-node is a record describing a file or directory. Each i-node has an i-number that gives the i-node's place in the i-list. All file status information is in the i-node, as are the direct and indirect pointers to the file's data blocks. See also **i-list.**

kernel. The kernel is the part of CTIX that manages the resources of the MightyFrame system: for example, I/O and memory. Kernel software resides in the file **/unix**.

loadable driver. A loadable driver is a device driver that can be loaded dynamically with each system boot according to an entry in **/etc/drvload**. All supported CTIX drivers, except those that manage disks, are loadable. See also **device driver**.

local printer channel. A local printer channel is an RS-232-C channel on a Programmable Terminal or Graphics Terminal. A device (normally a printer) can use a local printer channel only when the PT or GT is communicating with a MightyFrame computer via an RS-422 channel. See also **lp**.

logical block. A logical block is a conceptual entity that is 1Ø24 bytes long (the size of two physical sectors). See also **block** and **section**.

lp. **lp** is a group of programs and commands that configure the **lp** print spooling system and execute print requests. The **lp** (line printer) spooling system is a more powerful spooling system than **lpr**, in that it allows multiple printers and multiple queues for print requests. A MightyFrame computer system can use only one type of spooling system, **lp** or **lpr**. See also **lpr** and **local printer channel**.

lpr. **lpr** is a simple print spooling system that can be used if there is only one printer to execute print requests. **lpr** is incompatible with **lp**. See also **lp**.

mount point. A mount point is an empty directory on which a file system is placed in the CTIX file system hierarchy.

multiuser mode. Multiuser mode is the normal operating mode for CTIX.

ordinary file. An ordinary file is a file that is not a directory and not a special file. See also **special file.**

Release Notice. A Release Notice is a piece of documentation accompanying a specific release of software. A Release Notice describes new features of the software release, installation instructions, bug fixes, and known bugs.

Reserved Area. The Reserved Area is slice Ø on a CTIX disk. At minimum, the Reserved Area contains the Volume Home Block and a bad block table. The Reserved Area can also contain a dump area, a download area, a loader area, and a program area. See also **Volume Home Block.**

root. The root user is the user whose numeric ID in **/etc/passwd** is **Ø**; this identifies root as the superuser. See also **superuser.**

root file system. The root file system occupies slice 1 on the boot device and holds the most essential system directories and files, including **/unix**, **/bin**, **/etc**, and so forth.

run-level. A run-level is a configuration allowing a select group of processes (and their descendents) to exist. A run-level is thus defined by a set of entries in **/etc/inittab** consisting of processes for **init** to spawn and its

specified value for "rstate" (system state during which the processes are allowed to exist). Once **init** has entered a particular run-level, it spawns only those processes allowed to run at that level. See also **init**.

sector. A sector is a physical entity on a disk and is 512 bytes long. A disk drive's basic access to the disk reads or writes a physical sector.

shell. The shell is the CTIX command line interpreter. CTIX supports two command line interpreters: the Bourne shell and the C shell.

single-user mode. Single-user mode is the CTIX mode that allows only one terminal to be in use. The single user in single-user mode has superuser status. Single-user mode is used for procedures that require an absence of normal disk activity.

slice. A slice is a section of a disk that is used as a unit. (A slice is sometimes called a partition.) A CTIX disk can have one to 16 slices. More than one slice can occupy a particular area of a disk.

special file. A special file is an entry in the **/dev** directory that represents a way to access a peripheral device (for example, a serial channel, a slice on a disk, and so forth). A special file is either block or character. Some kinds of peripherals are represented by both block special files and character special files. See also **block special file** and **character special file**.

superblock. The superblock is physical block 1 in a file system and contains data structures that describe the file system. See also **file system**.

superuser. Superuser status is conferred on a user whose user id is Ø. Superuser status removes important CTIX restrictions: the superuser can write to or read from any ordinary or special file; the superuser can also execute certain commands that no other user can execute. See also **root**.

swap space. Swap space consists of one or more areas on a disk that function as auxiliary storage for internal memory in a virtual memory operating system. Transparent to a user or a user program, pages of virtual memory are swapped into real memory from disk when needed and subsequently swapped back to disk to make room for other memory pages. The conventional area for CTIX swap space is slice 2 on the boot device. Add-on swap space (swap space additional to the main swap device) can be specified using the **swap** command.

umask. The umask is the CTIX file creation mode mask. It masks out file protection bits when new files and directories are created on the system. (With no mask, files and directories are created with a mode of 666.)

uugetty. **uugetty** is a special form of the **getty** program for use by bidirectional and other smart modem connections. The **uugetty** program allows dial-in calls, and when the line is free, allows other programs (for example, **uucico**, **cu**, and **ct**) to use it for dialing out. See also **getty**.

Volume Home Block. The Volume Home Block (VHB) occupies logical block Ø in Slice Ø of a CTIX disk. The VHB is a data structure describing the disk.