TRICEP

INSTALLATION GUIDE AND OVERVIEW

MORROW 

Manual No. 270-01150-00

DISCLAIMER

No representations or warranties, express or implied, are made with respect to the contents hereof, including, but not limited to, the implied warranty of merchantability or fitness for a particular purpose. Further, Morrow Designs Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation to notify any person of such revision.

Morrow
600 McCormick St.
San Leandro, CA 94577

IMPORTANT WARRANTY INFORMATION

LIMITED WARRANTY

Morrow Designs, Inc. warrants its products to be free from defects in workmanship and materials for the periods indicated below. This warranty is limited to the repair and replacement of parts only.

This warranty is void if, in the sole opinion of Morrow Designs, Inc., the product has been subject to abuse or misuse, or has been interconnected to other manufacturer's equipment for which compatibility has not been established in writing.

Circuit boards - Parts, including the printed circuit board, purchased as factory assemblies, are warranted for a period of ninety (90) days from the original invoice/purchase date.

Electro-mechanical peripherals - Peripheral equipment such as floppy or hard disk drives, etc., not manufactured by Morrow Designs, Inc., are included in the limited warranty period of 90 days from the original invoice date when sold as part of a Morrow system.

Exception - Expendable items such as printer ribbons, software media, and printwheels are not covered by any warranty.

Software/Firmware - Morrow Designs, Inc. makes no representations or warranties whatsoever with respect to software or firmware associated with its products and specifically disclaims any implied or expressed warranty of fitness for any particular purpose or compatibility with any hardware, operating system, or software/firmware. Morrow Designs, Inc. reserves the right to alter or update any program, publication or manual without obligation to notify any person of such changes.

LIMITATION OF LIABILITY: THE FOREGOING WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MORROW DESIGNS, INC. BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF MORROW DESIGNS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

WARRANTY RETURN PROCEDURE

Should a buyer experience a defect in either workmanship or materials during the warranty period, any Morrow Authorized Service Center will replace or repair the product at its expense only if the product is promptly returned to the dealer or Service Center with dated proof of purchase.

Should factory repair be necessary, the Service Center shall contact Morrow Customer Service for a Return Materials Authorization (RMA) number.

INTRODUCTION TO TRICEP

TABLE OF CONTENTS

Configuring TRICEP, continued

FOREWORD TO THE TRICEP MANUAL

Welcome. You have found the portion of the manuals that deals directly with your TRICEP system. The instructions for installing and configuring your system are in these pages. Other information that applies to your system hardware, the TRICEP, are found here.

The bulk of the documentation that you received was originally written by Bell Labs or the University of California, Berkeley. This material was edited by UniSoft so that it reflects UniPlus+ System V running on Motorola 68000 processors. This information is as correct as is humanly possible. It is also of little help to anyone new to the UNIX system.

Besides providing TRICEP-specific information, I will attempt to guide you to an understanding of the UNIX system. You will probably need more help than this manual provides if you are new to UNIX. There really is so much to UNIX that it takes several books just to explain most of the things available.

I will also try to convey to you some of the excitement I had when I started working with UNIX. Here, at last, was an operating system that could be controlled through files written in English (almost). Here was a wealth of commands, an easily organized file system and a flexibility of devices (printers and terminals). No more poking bytes into mysterious locations in memory. No more directories with 200 incomprehensible file names. I was really pleased.

So, here we go. I will try to remember what was mystifying when I first met UNIX and explain it to you. Some of these explanations are elementary, and not intended for UNIX experts. I don't want to lose anyone by too complex an explanation. I also don't want to put anyone off by being too simplistic, so please bear with me.

Finally, I realize that no one likes reading manuals. Manuals are the source of last recourse. Since no one reads entire manuals, this one is organized so that you can pick out and read what you need. Most topics are covered quickly in an overview, then beaten more or less to death in a wordy version. Diagrams are included where I felt they'd be helpful. There also is an index in the back that can help you find what you need quickly and a table of contents in the front.

Questions and comments about this manual can be mailed to me in care of the Documention Department at Morrow.

Much Good Fortune,

Rik Farrow
February 12, 1985

INSTALLING YOUR TRICEP HARDWARE

This section of the manual explains how to install and test the TRICEP computer and the first terminal (or "console"). Installation of additional terminals and a printer are covered on pages 49 and 58 respectively.

Installation Overview

Detailed instructions appear after this list.

1. Unpack and check for external damage; remove the cardboard slip from the floppy drive.

2. Choose a suitable location for the computer and its peripherals.

3. Plug the TRICEP and one terminal into grounded power receptacles.

4. Turn TRICEP on (turn the front panel keyswitch clockwise) -- are the fan and keyswitch pilot light on?

5. Set the terminal switches for 19,200 baud, 7 data bits, 1 stop bit and even parity. Turn it on -- can you see your cursor?

6. Connect a standard RS-232 serial cable between this terminal and the CONSOLE connector on TRICEP's rear panel.

7. Flick the TRICEP keyswitch to "RESET", and press the RETURN key when the colon (:) appears onscreen. This should boot the UNIX system (you will see the "#" prompt when this is done). Go to "First Time Use," page 17.

Step 1: Unpack and Inspect

Carefully remove the TRICEP computer from the largest carton and save all packing materials in case you ever want to ship the system again. Besides TRICEP, you should have also found:

- the UniSoft documentation (nine binders) and these instructions,

- a single diskette labelled "Standalone Floppy" or equivalent. This diskette is used only for repairing damaged file systems on the hard disk, for example, if the hard disk was excessively jarred during transit.

- a power cord, and

- a wide, flat cable for use with parallel or "Centronics-type" printers.

If you see obvious physical damage to TRICEP notify your dealer or the carrier immediately for filling out a damage claim.

Keep your carton and packing materials! They provide the best protection in case you ever ship your system. If it is ever returned to the factory, Morrow requires that it be shipped in its original container for insurance reasons.

NOTE:

There is a slip of cardboard in your floppy drive to protect its heads during shipping. Remove it before applying power to the system. Lift the door latch and slide it out. Leave the door open.

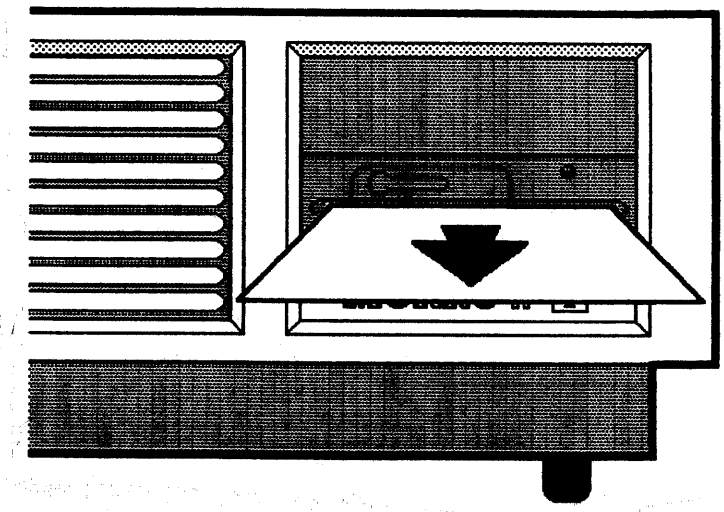


Figure 1. Remove the cardboard slip from the floppy drive.

Step 2: Select a Location

The choice of the location for your TRICEP is up to you. However, there are four requirements that must be followed. Your TRICEP should be installed where it will remain cool, level, stable and dry.

- Cool: room temperature below 85 degrees F (28 C). Allow 4" clearance (10 cm) on all sides for ventilation. Don't put anything underneath the TRICEP cabinet that could impede air flow.

- Level: to maintain proper operation of the disk drives.

- Stable: since a sharp jolt can be one of the worst things to happen to a hard disk drive.

- Dry: normal office or domestic environment is fine. There are occasional problems caused by static electricity when an environment is TOO dry. In these cases an anti-static rug in the immediate area is advised.

One final suggestion about location. Like a stereo system, all of the connections for your system are located on the back of the cabinet. If you can arrange the system so that you can stand behind it, at least while installing it, everything will be a lot easier. Once the physical installation is completed, you can move the system so that the back faces a wall. Remember to leave at least 4" (10 cm.) at the back and over the top of the cabinet for ventilation.

Step 3: Plug in the Power Cords
(After Checking TRICEP's Voltage Selection Switch)

First, be sure that the power switches to TRICEP and the terminal are OFF. The TRICEP has a keyswitch on the front panel. Turn it fully COUNTER-CLOCKWISE to ensure that your system is off.

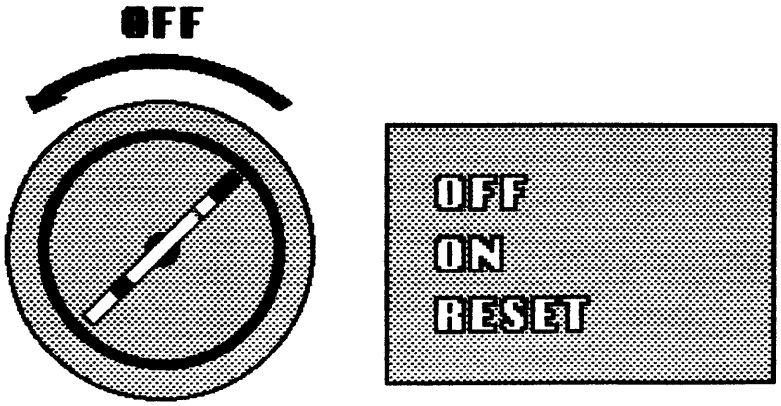


Figure 2. Be sure the keyswitch is off (fully counterclockwise).

SPECIAL INSTRUCTIONS FOR 220 VOLT SYSTEMS

TRICEP comes configured from the factory for 115 volt operation. Configuring it for European locations consists of:

- Resetting the voltage selection switch from 115V to 230V
- Replacing the 8 amp fuse with the 4 amp fuse taped to the power supply
- In most cases, acquiring a European-style power cord or replacing the plug on the Morrow-supplied cord

See Figure 3 for locating the switch and fuse holder.

If this TRICEP is being used with a 220 volt supply, perform the three steps above before proceeding.

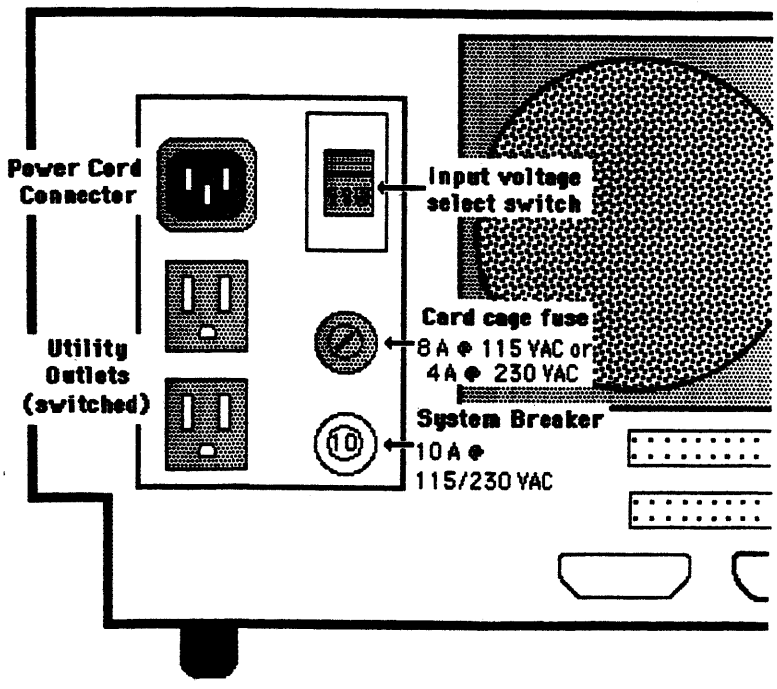


Figure 3. Rear panel power supply details

Regardless of whether you're using 110 or 220 volts:

Just to be absolutely sure that your available voltage matches what the computer needs, check the switch on the rear of TRICEP (Figure 3). The number showing on the switch is the voltage level the computer expects. Never change this switch when the power to TRICEP is on.

Plug the female end of the power cord into the power cord connector. Plug the other end of the power cord into the outlet you have selected. Your TRICEP can be plugged into any grounded (three-prong) outlet.

If grounded receptacles are not available:

Use a three-prong to two-prong adapter to plug the power cord into the wall, and plug accessories, such as your terminal and printer, into the utility outlets on the back of the TRICEP. When using an adapter, be sure to fasten the metal tab or wire from the adapter to the screw that secures the socket cover to the wall.

If you need more outlets:

Get a power strip and plug everything into the power strip. Computer systems and their accessories work best if all components are using the same ground. If you are going to have terminals that are located a distance away from the TRICEP, use serial (RS-232) cables that include a ground on pin 7. (Morrow cables supply this).

Voltage fluctuations:

Besides a shared ground, TRICEP and its peripherals (terminals, printers, etc.) require a steady power supply. The minimum suggested voltage for TRICEP is 105 volts, and the maximum is 125 volts. For 230 volt systems, the tested voltage range is 208 to 265 volts. In the United States and most of Europe, the power companies are pretty reliable at delivering electricity in this range, so most of you won't have trouble with this.

One problem that you need to avoid is voltage variations due to the startup of large electrical appliances like air conditioners, space heaters, and freezers. If an electric lamp connected to the outlet you want to use dims noticeably when one of these appliances turns on, you may be plagued with erratic performance from your computer. The only solutions are running a dedicated power line from your main service panel to either the computer or the appliance, or connecting an uninterruptable power supply between TRICEP and its outlet.

Step 4: Powering Up The First Time

For future reference: Later, when you have terminals, printers, and external disk drives connected to TRICEP, you should turn them on first, before turning TRICEP on.

Apply power by turning the keyswitch one click clockwise. The power is now ON to TRICEP.

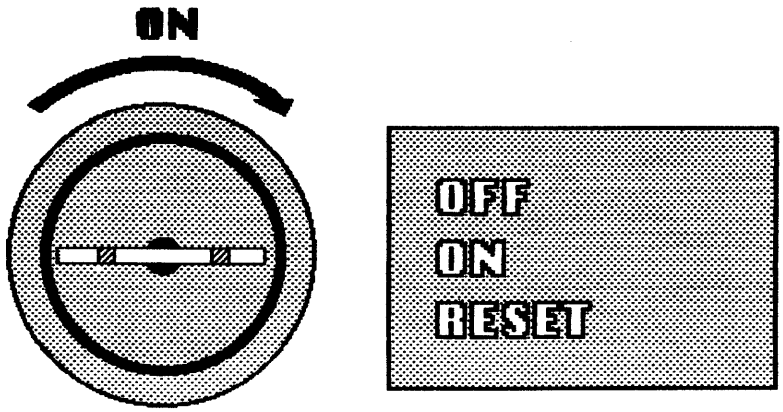


Figure 4. Turn the keyswitch to the ON position.

Checkpoint:

A pilot lamp inside the keyswitch panel should now be glowing. The fan at the back of cabinet should also be on. You can hear noises from the hard disk (inside the cabinet, under the floppy drive) as it comes up to speed. The two grounded power outlets on the rear panel of TRICEP now have power applied as well. If everything checks out here, go on to the next step, Setting Up the First Terminal.

Troubleshooting:

If neither the pilot light nor the fan came on, check:

1. The connection of the power cord to the back of TRICEP.
2. The power outlet. Plug a desk lamp into the outlet to make sure that you have power.
3. The circuit breaker on the back of the TRICEP. It is of the push-button type seen on stereo receivers and speakers. Push it all the way in and release with a snap.

4. The fast-blow fuse above the circuit breaker. It is inside a screw-in screw-out socket. Check it visually or with an ohmmeter. If blown, it may indicate a defective TRICEP power supply. Replace it once -- if the new one blows, report it to your dealer. 110 Volt Systems: 8 amp (Buss ABC 8) 220 Volt Systems: 4 amp (Buss ABC 4).

5. The power switch itself. Flick the keyswitch a couple of times.

If you still have problems, contact your dealer.

If the fan comes on, but the pilot light doesn't, TRICEP may well be operational with the exception of the lamp itself. If the fan is not working, do NOT operate TRICEP as it will overheat. Report it to your dealer.

Step 5: Setting Up the First Terminal ("The Console")

This section covers attaching ONLY the first terminal. If you're going to have more terminals, these procedures apply in principle, but there is further information under "Connecting Additional Terminals" on page 49. For now, let's concentrate on getting just the first one working.

The first terminal connected to a TRICEP system is called the console. On large computer systems, the console is used by the system administrator. TRICEP's console is used both by the system administrator and ordinary users. When you first bring up your TRICEP system, the console is also the only terminal that you can use. In fact, whenever the system is in "Single User mode," only the console will be operative. (Making other terminals usable is known as "going multiuser".)

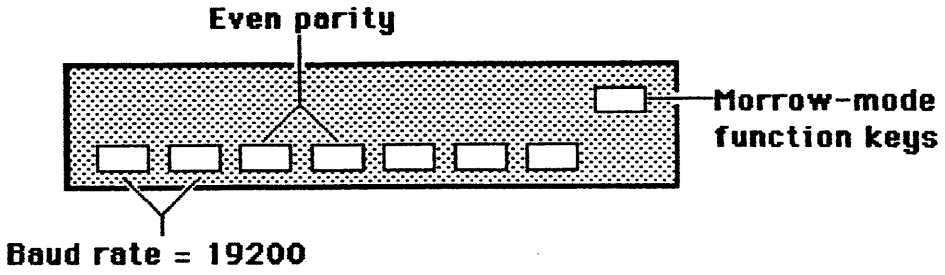
Terminal settings for models other than the Morrow MT-70

Terminals have many switch-selectable features, of which only a few concern us at present. The first, and most important, is the baud rate, that is, the transmission rate in bits per second. TRICEP expects the console terminal to be set to 19200 baud. If the terminal that you plan to use will not work at 19200 baud, you have a problem. TRICEP requires a terminal set to 19200 baud at this point.

IF YOU'RE USING A MORROW MT-70 TERMINAL:

Contrary to the illustration on page 2-4 of the MT-70 User's Guide, you should set your terminal configuration switches as shown below. (The MT-70 manual was printed before TRICEP was developed.)

When the switches are set correctly, you can skip ahead to Step 6, Connecting the Console Cable. Those of you with other terminals will probably have to dig into its manual for instructions on setting its baud rate, etc.



Look up baud rate selection in the operation manual for the terminal you are using, and set the terminal for 19200 baud. While you are looking at this section of your terminal's manual, also set the switches for:

- full duplex - since the TRICEP echos the characters you type back to your screen, if you see two of everything later, you should double check this setting;
- 7 data bits - the number of bits used to send a character;
- 1 stop bit - the pattern that marks the end of a character;
- even parity - parity is an option that is used for checking transmission accuracy; and
- on line - as opposed to local, means that the keys you type are transmitted to the computer; sometimes this switch will be on the keyboard, or may not be present at all.

Once again, the most likely cause of trouble here is the baud rate. The usual sign of using the wrong baud rate is that the startup messages will appear as a nonsensical collection of assorted characters and punctuation ("garbage"). You can later change TRICEP's software to communicate more slowly than 19200 baud if you so desire, but you have to use a terminal communicating at 19200 to do this.

Step 6: Connecting the Console Cable

By the "console cable", we mean the standard serial cable that should be packaged with the terminal. In the case of the Morrow MT-70, it will be white and about 4 ft. long. It doesn't matter which end is which, and you can't connect it upside down.

Using the Morrow MT-70 as an example, refer to the illustration below. Be sure to connect the cable to the proper connector on TRICEP.

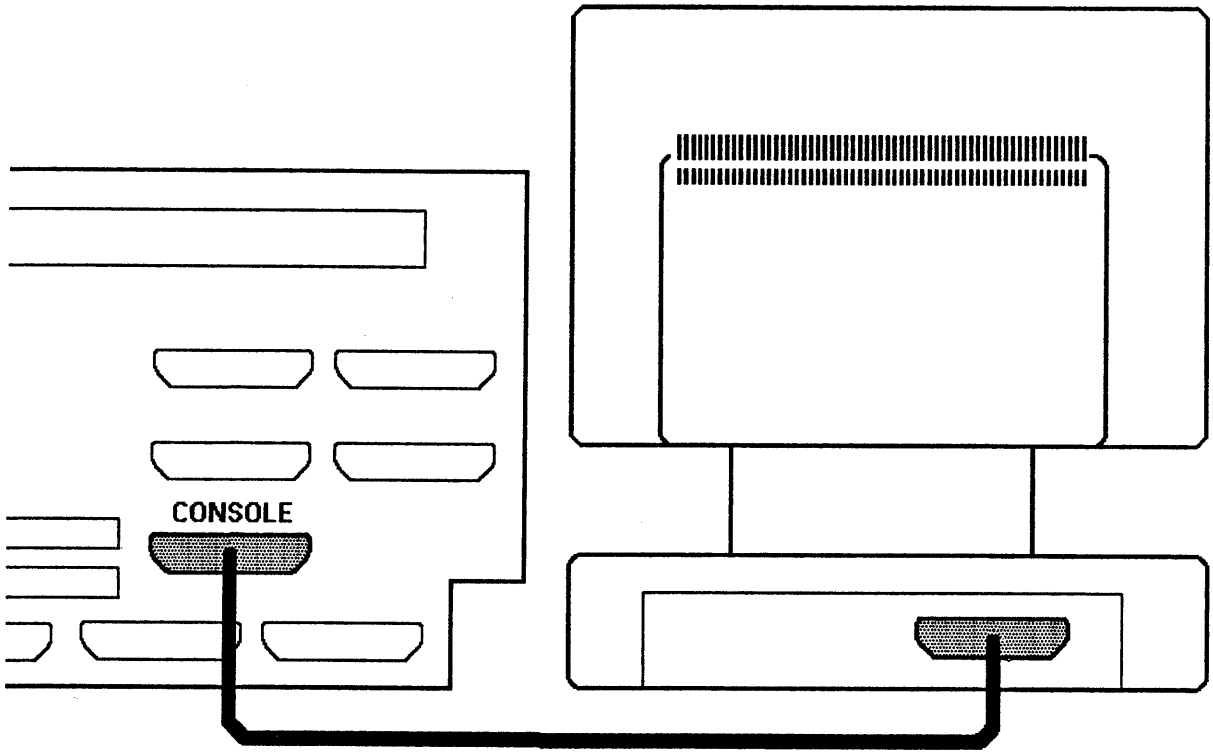


Figure 5. Morrow MT-70 connected as the console

MT-70: Connect The Keyboard Too

The Morrow MT-70 (like many others) has a detached keyboard that connects to the main part of the terminal with a coiled cord. If connecting the keyboard cable is not immediately obvious, refer to the MT-70 User's Guide, or your different terminal's manual.

Other Terminals:

There will be one or more female 25-pin connectors on your terminal's rear, the same size and shape as TRICEP's console connector. If there is only one connector on the back of the terminal, great. If there are two, use the one labeled "modem", "RS-232", "main port", or "P1", not the one labeled "printer" or "aux". The second connector is not designed to be connected to a computer, and you don't need it.

Connect the terminal's cable between TRICEP's CONSOLE connector and the correct connector on the back of your terminal.

Plug in the terminal and turn it on.

You can use one of the two utility sockets on the back of TRICEP to plug in your terminal, although we suggest that you use these sockets for extra hard disks or other peripherals. Use a grounded socket, preferably the twin to the one TRICEP is plugged into.

When the terminal is turned on, it may "beep". After warming up, (5 seconds or so), you should see a marker on the screen of your console called the cursor. The cursor marks the place where the next character will appear on the screen.

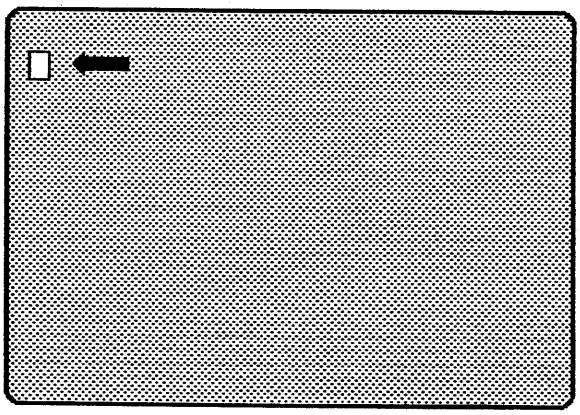


Figure 6. The cursor

If you don't see the cursor on the screen, you may need to adjust your terminal's brightness control. Before you leave this paragraph, find your cursor. If you can't find the cursor, something is wrong with your terminal, so you might as well just mark your place in this manual, and return when your terminal is okay. Thanks.

Step 7: Booting, The First Time

Everything ready? Grasp the keyswitch and turn it clockwise to "RESET" and release it.

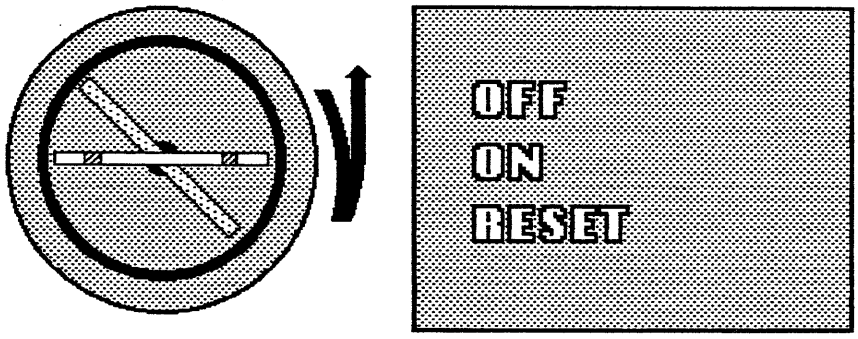


Figure 7. Flick the keyswitch to RESET and release.

The following message should appear on your screen:

```
Morrow TRICEP System  (triboot ###)
Testing memory.
```

(If you don't see this, but you do see something, you have terminal problems. Gibberish on the screen indicates a bad baud rate. Set your baud rate to 19200 and try RESET again. Review the instructions under Step 5. If you don't see anything besides the cursor, doublecheck the cable connection, as described in Step 6.)

Testing memory takes between 10 and 30 seconds to complete. Then, several more messages will be sent to the console:

```
Memory OK
######## bytes of memory

5 inch floppy not ready.

8 inch floppy not ready.

Press RETURN to continue
     Standalone boot

: []
```

Parenthetically, what the messages mean:

The memory test and the attempts to access the "not ready" floppy disks are part of a program stored in the PROM chips on the processor board. The amount of memory found is given as the decimal number of bytes.

The attempted floppy disk accesses allow booting the system without using the hard disk. If a floppy disk has been inserted in the first 5 1/4 or 8 inch floppy drive and the door closed, the system will ask if you want to boot from the floppy. This is used for repairing file systems on the hard disk and will be explained later (page 101).

After the PROM program has checked the floppy drives, it loads the "Standalone boot" program from the hard disk and prints the message on the screen. The CPU, memory, hard disk and disk controller have checked out okay if you reach this point successfully. If you have other messages on your screen, please jump ahead to Troubleshooting on page 13.

Proceeding with booting: Press RETURN

When the colon appears, the Standalone boot program is waiting for you to press the RETURN key. Please press the RETURN key now. A new set of messages will be added to the console:

```
mw(0,0)unix
Loading at 0x400:  87708+15404+133916
(C) Copyright 1983 UniSoft
```

This message reports the successful loading of the UNIX kernel (the essential part of UNIX that must always be in memory) and the initializing of its data areas.

The final set of messages report the copyright notice, the version creation date and amount of memory remaining after loading the kernel. The system then checks the SIO4 (terminal controller) circuit board(s) and finally places the superuser prompt on the screen:

```
#
```

```
**********************************************************************
*                                                                    *
*    If you have this symbol (#) at the bottom of your               *
*    screen, you have successfully installed your TRICEP             *
*    system. The UNIX operating system is running in single          *
*    user mode. You should skip over the next section, and           *
*    move on to FIRST TIME USE (page 17).                            *
*                                                                    *
**********************************************************************
```

Troubleshooting Booting Problems

This section explains what problems might occur while booting, what the symptoms of these problems are and how to cure them. This information is not only useful for solving problems the first time you boot your TRICEP, but anytime you have a problem booting.

As you begin to follow these instructions, we have assumed that you have the power on to both your TRICEP computer and your console terminal, you can see your cursor and you have connected your terminal to TRICEP's CONSOLE connector. This was explained earlier in Steps 5 and 6.

We will also assume that you have RESET the computer by turning the key in the keyswitch clockwise, and releasing it. The TRICEP system also performs an automatic RESET when you turn it on.

1. Nothing on screen after RESET

Most of the information that you need for diagnosing problems with your system comes from your console terminal. This poses a problem if you turn your system on and nothing appears on the console. Is the problem in the terminal or the computer?

The solution to this is to listen to your computer. TRICEP makes certain noises if it is able to follow the PROM program. If you can hear these noises, the computer is working and the terminal is not.

Immediately after resetting, the PROM program runs a memory check. This is silent, and takes from 10 to 30 seconds, depending on how much memory you have. (Don't forget to wait for this!) Next, the program tries to read a diskette in the 5 1/4" drive. This involves loading the heads, and that makes a clicking sound in the floppy drive mechanism.

Reading from the floppy disk will fail (unless you have inserted a formatted diskette and closed the drive door). The PROM program then tries to read the Standalone boot program from the innermost tracks of the hard disk. You should be able to hear the drive's stepper whirr as it moves its heads into this position. This is not easy to hear, so you may need to hold your head near the hard disk, which is located just below the floppy drive.

Did you hear the disk drive noises? If you did, you have some problem with your terminal. Go back to Steps 5 and 6 and check out your terminal's settings and the connection between the terminal and the computer. It's easy to use the wrong connector, on either the TRICEP or the terminal.

The problem might also be in the terminal's cable itself. TRICEP looks for a hardware signal that comes back from the terminal via pin 20. This signal is known as Data Terminal Ready. If your cable doesn't provide for a connection between pin 20 of the terminal and pin 20 of the TRICEP console connector, nothing will be transmitted.

If you don't hear any noises from the disk drives, you may have a hardware problem in the computer. See Appendix A, "Checking for Hidden Damage". The problem may be loose pc boards or cables. If this doesn't help, you should contact your dealer for assistance.

2. Garbled characters appear onscreen

This is an easy one. You have set your terminal to the wrong baud rate. You MUST set your console terminal to 19200 baud for booting TRICEP. This fact is carved in stone. Go back to Step 5 for more details.

3. Messages about memory failures appear

TRICEP requires at least 512 kilobytes of RAM memory to be able to boot. The number of bytes of memory available is counted during memory testing. If no memory at all is found, or memory is incorrectly addressed, the message:

RAM is missing or incorrectly addressed.
Please refer to the installation manual for further instructions.

A second message is displayed if less than 512 kilobytes of RAM are found:

Warning: Insufficient system RAM available.
UNIPLUS System V requires a minimum of 1/2 megabyte of RAM.

The proper response to both of these messages is to follow the instructions given in Appendix A, "Checking for Hidden Damage".

4. "Can't Boot" message

This message may appear after memory has been checked, and the floppy disk drives accessed. It may be preceded by the word "Time-out." What has happened is that the PROM program has been unable to access the hard disk. Occasionally, this is caused by insufficient power to the hard disk during power on, and can be cured by turning off the system. Wait 15 seconds or so, and try powering up again. This may be a reoccurring problem during power on, but it doesn't seem to affect system performance after booting.

The other possibility is that a board or cable is loose. Once again, please follow instructions in Appendix A, "Checking for Hidden Damage".

5. System doesn't respond to pressing RETURN

This is a manifestation of a simple problem. The correct messages have appeared on the terminal's screen, but pressing RETURN does not continue the Standalone boot. The problem is that data can get from the computer to the terminal, but not from the terminal to the computer. This is caused by one of three things.

First, you may have your terminal in "Local" mode, allowing it to receive but not send. Pressing RETURN causes the cursor to move to the left margin, but not to the next line. Check your terminal's settings.

The second cause is that your terminal may have received the "lock keyboard" character. This has the unpleasant side effect of making a system appear to be crashed if it occurs after a successful boot. The solution to this is to turn the terminal off, wait a few seconds, the turn it back on again.

The other cause for this is an incorrectly wired terminal cable. The pins that carry transmitted data from the terminal to received data at the computer are not connected. Pin 2 on each end should connect to pin 3 on the other. Pin 20 should be connected on both ends. This can be verified with an ohmmeter, or with a visual inspection of the cable's wiring.

6. System doesn't start after pressing RETURN

The hard disk has succeeded in loading memory with the UNIX kernel. This has involved the cooperation of the CPU board, the PROM, the console terminal and memory. If the system fails at this point, the file system on the hard disk may have been corrupted or damaged, possibly through shipping or incorrect shutdown. Please follow the instructions in the chapter "Using the Standalone Floppy".

7. Loading fails

This is much harder to diagnose because once you see the message

```
mw(0,0)unix
Loading at 0x400:  86268+...
```

most of the system has been used successfully. One possibility is that the hard disk is still not fully up to speed. This is the same cause of failure mentioned earlier in the "Can't boot" case. Simply flick the keyswitch to RESET and try again. If this doesn't work after two or three attempts, contact your dealer for assistance.

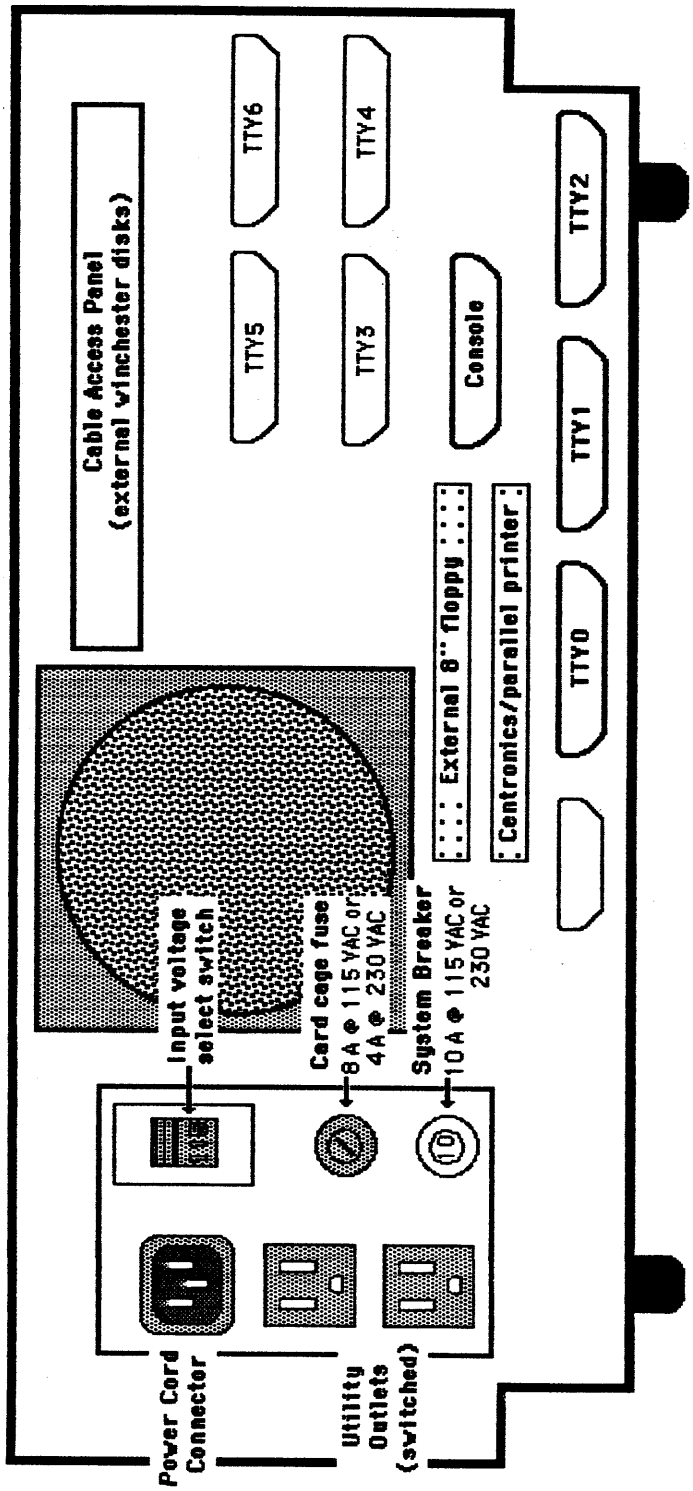


Figure 8. TRICEP rear panel

FIRST TIME USE

This section of the installation manual is broken into two parts: 1) testing and backup of TRICEP's UNIX file system and 2) a quick tour of the file system. Details on Configuration follow in the next chapter.

The testing and backup will take several hours, but is necessary for a reliable system. During this time, you may, if you like, read ahead and familiarize yourself with your UNIX system.

The quick tour of the file system provides a map of the file system and the commands used to move around in it. You may want to skip this section if you have worked extensively with UNIX systems in the past. However, if you are responsible for configuring this system, understanding the arrangement and organization of directories will be of immense assistance to you.

Much of the material presented here is covered by the UniSoft manuals, and by books on UNIX. The difference between these other sources and this manual is that here you are given advice specific to your TRICEP, rather than for some generic UNIX box. We can't and won't try to duplicate Bell Lab's documentation, but we will tell you the best way to handle TRICEP-specific operations, like backing up and configuration.

Orientation: The Superuser Prompt (#) is on your Console Screen

This is where you stopped at the end of the first chapter. If necessary, go back and follow through the booting steps until the # prompt appears on the left side of your screen.

CHECK YOUR FILE SYSTEM!

Just because you can see the superuser prompt, #, on your console doesn't mean everything is is perfect with the system. What we are concerned with now is that there may be subtle damage to the information pre-recorded on your hard disk.

Morrow has shipped you a complete 12 megabyte-or-so UNIX System V, adapted to the TRICEP hardware by UniSoft, ready to go on your hard disk. But, is it all there, complete and undamaged? You will check this in three stages, with these commands:

badblks - a surface check for damaged sectors ("bad blocks"),

fsck - "file system check", for consistency, and

tar - "tape archiver", for the creation of your master backup copy of the hard disk on diskettes.

We can't overstress the importance of following these preliminary file system safeguards. It is time-consuming, we admit, but there may come a time when you'll be very glad you took the time.

Checking for Bad Blocks

With your cursor next to the superuser prompt (#), type the following command at the console. Follow every command with the RETURN key.

```
# badblk /dev/rmwØhwØ
```

That will take a while to run, so you can relax, and read ahead. Badblk starts out by reporting the number of badblocks previously discovered on your disk. Then, it attempts to read every sector on your hard disk. This takes about 28 minutes for a 34 megabyte hard disk. Badblk reports any bad (unreadable) sectors (blocks) it finds, assigns an alternate sector and updates the badblock map in block 0.

When badblk is finished running, write down the numbers that appear on the screen. These are damaged sectors that have appeared since your system was tested. If you have several damaged sectors that aren't being used, they will not affect your file system. Badblocks with block numbers greater than 27000 are probably in the empty (unused) portion of the file system.

What if badblk discovers bad blocks with numbers less than 27000? If you have many (more than 20) of these bad blocks, you have a serious problem with your hard disk and should report to your dealer the number of unreadable blocks. If there are less than 20 bad blocks, you can probably continue to use your system. The files that have the bad blocks can be discovered through normal system use, and restored from your dealer's master copies.

For example, if you typed the command **cal** and the error message

```
mw0a error bn=1234 sr=0x4
```

appeared on the system console screen, you would know that the file **/bin/cal** had a bad block, number 1234, in it. Then, you can add this block number to the badblock list with the command

```
mwbad
```

and replace the file **/bin/cal** with a backup from your vendor. One bit of funny business about using **mwbad** is that it believes in 512 byte blocks. TRICEPs shipped or rebuilt after February 5, 1985 have 1kbyte sectors and logical blocks. This means that you take the block number reported by the error message, 1234 in our example, multiple it by two to get 2468, and use this number, and the next, 2469, with **mwbad**. This converts the new logical

block size to two block numbers in the old block size still "believed in" by **mwbad.**

If you don't like this hit or miss type of approach, you can try reading the entire file system, file by file, to discover the files with bad blocks with the command

/etc/vchk -kc > vchk.results

We suggest that you run fsck (see next section) first. Then run vchk -kc. This will take about 34 minutes, and will attempt to read every file in the file system, and produce an error report for every file with an unreadable block. Carefully note the block numbers and files where errors occur. Add the bad blocks to the bad blocks list and replace the files from backups.

Vchk, or version checker, checks a file system and compares it to a special list, called a V-check tree. We supplied your TRICEP with a V-check tree (/etc/vchk_tree) that tests to see that all necessary files, directories, ownership and permissions are intact. The -k option causes vchk to do a checksum on files in the vchk_tree. One option of vchk allows it to copy (take) missing files from a host machine. We will talk more about the vchk utility later.

File System Check

The badblk command attempts to read every block in the file system. The ability to read a block says nothing about the validity of the information found there. The file system check program, fsck, understands the structure of file systems. Fsck not only understands file systems, it can repair them. This makes fsck an indispensable tool in preventing file system problems. You should use fsck every day. To start fsck the first time, you can enter its name, followed by the RETURN key:

fsck

Fsck will look in the file "/etc/checklist" for the names of the file systems to check.

During file system checking, the fsck program will produce reports about what it is doing on the system console. If problems are discovered, you will be prompted before any action is taken. Unless you are already experienced with UNIX file systems, you should always answer "Y" for yes.

If the message "BOOT UNIX (NO SYNC!)" appears, reset your TRICEP by flicking the keyswitch to the RESET position. Reboot with the RETURN key, then re-enter the fsck command.

Fsck takes about 4 minutes to complete on a 34 megabyte drive. If fsck cannot check your file system, (for example, the message "Can not stat root" appears), you have a serious problem and should contact your dealer.

Further information about running fsck is given in the section "Every Day Procedures" (page 111), and also in the UniSoft System Administrator's Guide.

First Backup

The check programs have been pretty simple. You simply started the program, and perhaps provided a yes response or two. You'll need to provide a bit more assistance for the first backup operation.

The standard (and least expensive) form of backup medium is floppy diskettes. The down-side is that it's also relatively slow and error-prone. The section that follows describes first time backup procedures for floppy diskettes.

This procedure requires about four hours of time and 40 formatted diskettes. You, the operator, need to be standing by with your supply of formatted diskettes. Other than removing and inserting diskettes, your task will be essentially mindless, allowing you to continue reading through this manual (or whatever).

The Diskettes

As mentioned, you need to have 40 diskettes, and hopefully, 40 blank labels. TRICEP uses

5 1/4" double sided, soft-sectored diskettes

Please use only diskettes that are certified double-sided.

Formatting the diskettes

To format a diskette, insert it in the floppy disk drive, oblong opening first and seamless side up. Close the latch and type

diskformat /dev/rdjØ

for each diskette. This takes about one minute per diskette. Label these diskettes, after they are formatted, with the date and the words "MASTER TRICEP BACKUP". Number the diskettes sequentially from 1 to 40.

Backing up

Now you are ready to begin the backup process. Insert the first formatted diskette (number 1) in the floppy drive, close the latch, and type

tar cvfB 78Ø /etc

NOTE: The normal device specification **/dev/mt1** is the default for this command. Morrow has linked /dev/mt1 to the floppy device **/dev/djØa**. In case you have software that is missing this linkup, you'll receive a "can't open" error message with the command above. Use the command **tar cvfb /dev/djØa 78Ø /etc** instead.

This begins to copy the file system to floppy diskettes in tar (tape archive) format. After each file is copied, the filename and number of blocks (its size) will be reported. The tar command can only copy entire files to a diskette, so when a file is too large to fit in the space remaining on a diskette, it will request that you insert another:

Change disks and press return.

Put in the next diskette and press the RETURN key. The amount of storage required on floppies will be greater than the number of blocks of files on the hard disk because some space will be wasted at the end of each floppy, and by tar's padding.

You should write the names of the first and last files that are stored on each diskette on its label. You can also include the names of directories on each diskette. Directory names consists of the list of letters between the first and last slash (/). For example, in the name /usr/src/sys/config.c, the directory name is /usr/src/sys, dropping the part after the last slash. This will help you find the correct diskette if you need to replace a single file. Otherwise, you will have to go through all the diskettes sequentially until you find the correct file.

After you have backed up the /etc/ directory, go on to /bin, then /lib, and /usr, as in

```
tar  cvfB  78Ø  /bin
tar  cvfB  78Ø  /lib
tar  cvfB  78Ø  /usr
tar  cvfB  78Ø  /users /lost+found .cshrc .profile .setup .login /unix
```

Remember to have your supply of formatted, numbered and labeled diskettes handy BEFORE you begin. If you are one diskette short, or try to use an unformatted diskette, you will have to start the tar command over from the beginning. (Ahhhrrgg!!!) Go back to the diskette that began with that particular tar command and start over (with more diskettes in hand).

It takes tar about seven minutes to fill a diskette. You can use this time to copy down the first and last filenames, and the directory names, that are being copied to each numbered diskette. If you later lose a file, say, /usr/lib/lpadmin, you can insert the correct diskette, and type

```
tar  xvf  /usr/lib/lpadmin
```

Otherwise, you'll have to put the diskettes in one at a time, until you have searched through most of them to find the right one.

NOTE TO EXPERTS: The floppy device for using tar has a special designation: /dev/djØa as opposed to /dev/djØ. **tar** overwrites essential format information in the diskette's boot block unless the special device dj0a with a 1k offset is used. This leaves 78Ø 512 byte blocks per diskette.

If you have an 8" drive with its own power supply that you'd like to use with **tar**, you can connect the 50 pin cable to the 8" drive connector, just below the fan (see illustration, page 16). The red stripe on the cable should be toward the left (looked at from the back). The command

```
diskformat  /dev/rdj4
```

will format single or double-sided 8" disks with 1K sectors (with a single density boot track). Then,

```
tar  cvfB  /dev/dj4  `devsize  /dev/dj4`  names
```

archives the files or directories "names" on the first 8" drive.

Quick Tour of the File System

Still working on that first backup? This section gives you a quick look around your new home. The UNIX file system has a well-deserved reputation as an elegant solution for organizing many files on huge disks. This system is not hard to follow once you get a grasp of it, and can certainly work to your advantage if you bother to understand it.

If you have worked as a system administrator or programmer on a UNIX system before, this section may seem pretty old hat to you. System V's file system is laid out essentially the same as the file systems in Versions 6 and 7, and in System III. The obvious changes to UNIX are in init, getty and login, and many new commands.

Trees: the Basic Philosophy

This section provides an overview to the discussion that follows in the next section. The UNIX file system is usually compared to an upside down tree, with the root at the top and branches heading downward. The root directory, represent as / (slash) is the beginning of the file system. Ten directories branch off from the root, providing the basic subdivisions of the rest of the file system. These are:

bin the available commands

dev the device connections

etc the system administration commands and files

lost+found a place used by fsck to hold files and directories that have lost their names and their "parents"

t,f hooks for mounting file systems

tmp the place for temporary or scratch files

users the place where user directories will be located

usr the beginning of a large subtree, with more commands, more libraries, spooling directories, administration and accounting files

There are several files in the root directory: unix (the kernel program loaded by the PROM booting program) and the superuser's shell setup files (like .profile).

Why Trees?

This is a story of organization. The problem is simple: a 34-megabyte hard disk has the capacity for tens of thousands of files. If you are used to working with floppy disks, organization is simple because only a handful of files will fit on each floppy. The limited capacity of floppy disks provides a natural division and keeps directories small. You can keep all the files you are currently working with on one or two diskettes. When a diskette gets full, you copy the files you need to a new diskette, and the old diskette becomes the backup.

The problem is that it takes a long time to fill a hard disk. Suddenly faced with as much as 100 times the capacity of a single floppy, you can go a long time before running out of room. Directories can grow impossibly large, so that it can take five minutes just to scan through them. Then, suddenly, you have a serious problem. The disk is full and you don't know which files to delete.

The UNIX file system helps you in three ways. First, you create directories and use them as if they were file folders. Groups of related files fit in each directory. If a directory has too many files, you create a subdirectory and move files into it.

Second, you use easy to understand names for directories and files. The UNIX systems allows names of up to 14 characters, so you can have real names for filenames. This makes it easier to find files.

Third, the UNIX system makes entries in the inode (file descriptor) every time it accesses (reads) or modifies (writes) to a file. This can be used in locating the most recent version of many similar files.

Hints on file and directory names

I suggest that you use short names for directories. You will have fewer directories than files, so their names will be easier to remember. Directory names become part of filenames when used in commands, and having several long directory names can make a filename very long very quickly. Having to retype long filenames like "/usr/rik/textfiles/systemV/tour" gets old very quickly. I use "/usr/rik/Txt/SV/tour" and save myself a dozen keystrokes. Using capital letters for directory names makes them stand out in directory listings.

Moving Through the File System -- Made Easy

Okay, enough of the reason for being. Now, let's learn how to move around and see what's here. There are three basic commands for moving around the file system:

cd "change directories" moves you from one directory to another

pwd "path way to directory" tells you the name of the directory you're in, and by implication, how you got there

ls7 shows you what's there (a directory listing of files and subdirectories)

The Office of the Future

For this tour of the file system, I will be using the allegory of "the office of the future". Instead of describing the file system as it really is, I will pretend that is an office space that corresponds exactly to the file system. This will help those of you who are visually oriented get a feel for the file system.

The tour starts in the reception area, also known as the root directory. As you walk into the room, the receptionist asks you for your name (your login) and your reason for being there (your password). Then, she busies herself with setting up your visit, so you have a chance to look around.

The room you are in is almost devoid of detail (minimalist interior decorating, lots of chrome, subdued colors). The feature that catches your eye immediately is that there are nine doors leading from the room, each of them labeled: bin, dev, etc, etc. In the corner, there is an ordinary looking file cabinet labeled "unix". At that moment, the door labeled "usr" opens, and I step out, tall, serious and somewhat goodlooking.

We greet each other, and I launch into my introductory spiel:

"Welcome to the UNIX file system. As you may have already noticed, we are here in the root directory, the beginning of the file system. Later on, when you have your own HOME directory, that will be the beginning of your file system.

"There is a simple way to find out where you are in the file system at anytime by typing 'pwd'. Let's try it now."

Leaning over the secretary's desk, I pick out the keys p, w, d on her terminal, and press the RETURN key.

```
pwd
/
```

"The slash is the symbol representing the root directory. As we move through the doors that represent other directories, the directory names will be added after the slash of root. **pwd** stands for 'path way to directory'. Obviously, the path way to a directory will get much longer the further we move away from root."

Wanting to impress you immediately, I start moving you toward the door labeled "bin". To move through the "bin" door, it is necessary to type:

```
cd bin
```

meaning, "change directory to bin". The "bin" door opens and we step into a very large, dark room. To get a look around, you type:

```
ls7
```

turning on the lights and displaying the names of some 230 files. The files appear to be tools, each different from the next, all with some controls on top, and places where something goes in and the result comes out.

"This is the bin room, the workplace of most of the famous UNIX tools. There are tools for creating files, editing files, altering files and making new tools. Each tool is designed so that it can be hooked together with other tools. This way, new tools can be improvised simply by temporarily connecting them together.

"You won't actually be visiting here very often. When you use a command, this room . . . er, directory, will be automatically searched for the tool. To help you remember the name, bin stands for binary, and binary is short for "binary executable files", that is, programs. Look, there's pwd. Let's try it, even though we know where we are."

```
pwd
/bin
```

"You can see that the door we went though to get to this directory has been added to the slash for root. Okay, let's go back to the root and check out another directory."

cd /
cd dev
ls7

With these three commands, we leave bin, return to the root, walk over to the door labeled "dev", and walk in. This room has a very different character to it. There are about forty files here, arranged around the walls of the room. Some files have small, character-sized connectors, resembling phone jacks. Others have a large, squarish, block sized openings in the front. The files with the small connectors are attached to terminals, printers and even a clock. The files with the big block-sized openings are hooked into quietly humming hard disks and idle floppy disks.

One of the files with block-sized openings, labeled "mwØa", is connected to a transparent tube that disappears into the ceiling. Even as we watch, blocks flash through the transparent tube, coming and going to some place above the ceiling.

"This is the 'dev' room. dev is short for device. This is the directory where the connections between the file system and devices are made. You may have noticed that each file here represents only one physical device. You, or the system administrator, can control which connections are made to devices. A few connections, like the system console and the root file system, mwØa, are designed into the kernel and can't be changed.

"You could, for example, connect your terminal to someone else's terminal and type right onto their screen. There are, however, much more polite ways of talking to people than taking over their personal terminals. Now, let's look at the etc directory."

cd etc
No such file or directory

"Whoops, we made a mistake. Since we're in the dev directory, and the door into etc is located in the ROOT directory, our change directory command failed. We can take a shortcut, however. We can specify to the cd command that we want to get to etc AND we need to start at the root by typing:"

cd /etc
ls7

This room is filled with a bunch of office workers and a couple of long-haired programmers. There are several filing cabinets around the walls, with names like passwd, group, motd and gettydefs. There is also an organizational chart on the wall labeled with the funny name "inittab".

"This is the etc directory. The easiest way to remember the name is, of course, to think of et cetera. And so on.

"This directory houses the files and programs that configure your UNIX system. For example, the passwd file will contain the login names and passwords of all the users you will have on your system. That chart on the wall, inittab, determines what programs will be run initially. The gettydefs file contains login speeds for terminals, and motd has any message you want everyone to read after logging in.

"Those long-hairs over there represent your resident system programmers, fsck and fsdb, that check your file systems and repair damage to them. You'll quickly become familiar with fsck, because you'll use him every day. fsdb is a bit harder to communicate with, but can also be helpful."

cd /
ls7

"We're back in the root directory. I wanted to point out to you that several of the doors currently have empty rooms behind them. The t and f directories is used for attaching temporary file systems on other disks to the root directory. The tmp directory is for temporary files. If you make a habit of creating your temporary files in tmp, you won't have any trouble remembering which files you can safely erase because they are temporary."

"The lost+found directory is used by fsck. If fsck finds any files or directories that aren't listed in any directory, fsck manufactures a name for them and leaves them in lost+found. Makes sense, doesn't it?

"There's lots of empty space behind the 'users' door. The /users directory is where you will be creating "home directories" for your system users. Keeping your users together makes backing up simpler.

"Now, let's check out the last door, usr."

cd usr
ls7

Behind the door labeled usr is a long corridor with more doors. Two of the doors have familiar names on them, bin and lib. Other doors are labeled spool, admin and man.

"The usr directory is the entry point into the rest of the file system.

"The bin and lib directories have more tools and libraries within them, respectively. The man directory has all the manual entries stored in a format used by the nroff program. Before we finish the tour, let's move into the guest room so that I can make a point regarding pathways to directories."

```
cd users
cd guest
pwd
/users/guest
```

"Okay, our pathway to directory is now /users/guest. We started in root, /, moved into users, then into guest. Additional slashes are used to separate directory names. You don't have to use entire path names to specify a file, because you can form a file name relative to your current directory, that is, the room that you are currently visiting. Now, back to the root."

```
cd
```

Using the cd command without specifying a pathway to a directory means "change the directory to this user's home." In this case, the user is the root user, or "superuser", so we flash back to the root directory. Home directories are further covered on page 55.

The secretary now logs you out, and you step back into the duller reality of this manual. Is your backup finished yet? Tar will tell you it is complete and the # prompt will return.

Making Room: "Unnecessary" Files

Owners of TRICEP systems with 16 megabyte hard disks are immediately faced with a shortage of disk space. The UNIX system and swap space use up almost the entire hard disk. The solution is to remove some files from the file system. Since you have completed your backup, these files are still available if you find you need them later.

There are two major candidates for removal: the games and the manuals. The /usr/games directory is the sixth largest in the file system, and is not absolutely necessary. You may wish to compromise by leaving some games and their supporting files.

The /usr/man directory contains copies of the documentation produced by UniSoft. You have, on line, the equivalent of many inches of paper. The advantage is that you can use the man command to review manual entries. The disadvantage is twofold: the man command is painfully slow and doesn't allow you to see the beginning of an entry after it has passed. It would be nice if there were a better way of viewing manual pages, but there is not. A compromise here is to remove everything but /usr/man/u_man/man1 and /usr/man/a_man/man1. These two subdirectories contain the most often used manual entries.

There are several other candidates for removal. The benchmark shell scripts in /usr/guest/bnchmrks are not necessary. The /bin/efl program, Enhanced Fortran Language, is useless unless you have a Fortran compiler. There are also spelling dictionaries (/usr/lib/spell and /usr/dict) on line that you may wish to dispense with.

If you are interested in uncovering the largest files and directories in your file system, you can use the following commands to produce a list sorted with the largest file first:

```
du -a / | sort -rn > /tmp/filesizes
```

Please be certain that you have backed up before your remove any file. Sometimes removing a file will have unexpected side-effects, and you will need to restore the file from backups.

```
rm -r /usr/games
```

will remove the entire /usr/games directory.

```
rm -r /usr/man/u_man/man4
```

removes the "file formats" section of the online user's guide.

CONFIGURING YOUR TRICEP

Your TRICEP system runs the UniPlus+ operating system. This is essentially Bell Lab's System V UNIX with some enhancements. The documentation provided by UniSoft was written by Bell Labs and edited by UniSoft to be somewhat more applicable to Motorola 68ØØØ systems. UniSoft also added entries for the Berkeley UNIX programs that are included, such as the C-shell (csh), and exchanged the name UniPlus+ with the name UNIX. Unfortunately, the Bell Labs documentation is a bit thick for anyone that doesn't ALREADY know what they are doing with UNIX.

We will do our best to fill the gap between the generalized information provided by the UniSoft manuals, and the procedures that will work best on your TRICEP.

Human-Readable Configuration Files

The UNIX system has a simply wonderful feature. Most of its activities can be controlled by editing HUMAN-READABLE FILES. This accounts for some of the enthusiasm that people feel for UNIX. This chapter explains how the files that control your TRICEP fit in, and how to edit these files.

There is, of course, a small problem with the human-readable files. Your TRICEP must be able to read them also. This entails rigid structuring for these files. You must compromise with the machine, which is neither as clever or as creative as you are. You have access to all the flexibility offered, but only if you obey the rules. We will carefully explain the rules to you, so that you and your TRICEP will get along splendidly.

The human-readable files are introduced in the same order as they are used by TRICEP. Most of these files are involved in the process of going multi-user. We will start with an overview from boot to the appearance of the user's prompt ($ or %). Then, we will talk about each of the files used in this process, all in the /etc directory:

inittab - Controls which terminals are allowed to login, starts background processes, like the print scheduler, and the daemons, through /etc/rc;

gettydefs - Controls initial terminal baud rate and set up;

passwd - Establishes login names, holds passwords, user and group id's, home directory and shell program name;

profile - Works with the shell (not csh) to initialize every shell user's environment;

cshrc - Works like profile for csh.

We will also fit in connecting terminals, modems, additional disks and printers. The **lp** commands will be explained, so that you will be able to use the print spooling facilities of UNIX.

From Boot to Multi-User

The diagram on the next page outlines the process your TRICEP goes through to become a multi-user system. This diagram has been simplified somewhat by leaving out the scheduler (created by the kernel) and the single user state (the superuser shell started by init).

The first several programs, the PROM, Booter code and kernel, are not alterable simply by editing files. These form an invariant system initialization pattern that can be modified only by editing source files and remaking the kernel, PROM or Booter.

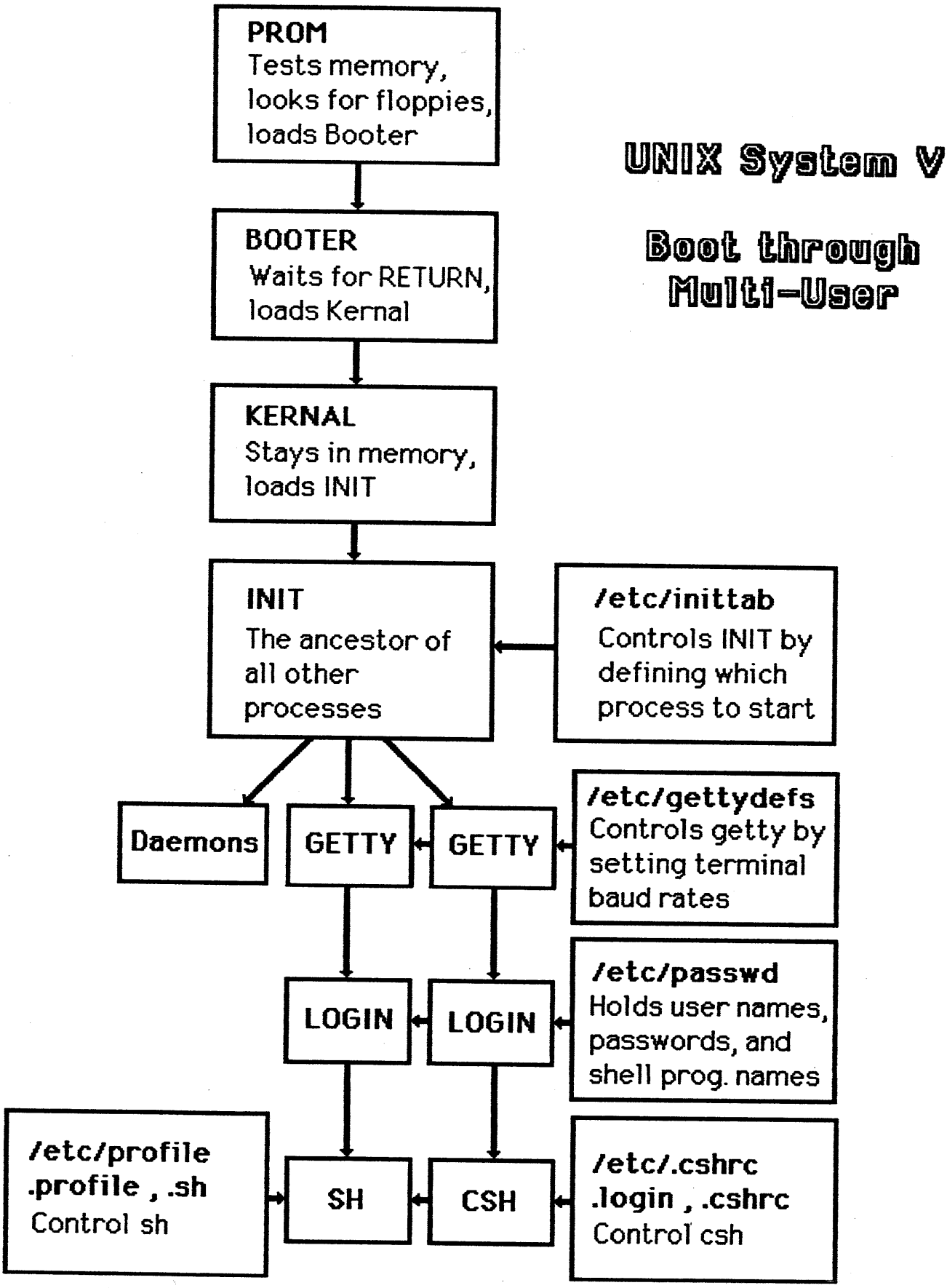
The PROM code has three tasks: check memory, check for floppy disks and to load the Booter code. The memory tester determines whether you have enough working memory to boot. The floppy disk checker looks at the first 5 1\4" and 8" drives. The PROM code will attempt to boot from any floppy disk that is inserted in a drive with the door closed. If there is enough memory (512 K) and no floppy disk is ready, the PROM code loads the Booter code from the hard disk. The Booter code is located on 18 blocks in the alternate block area of the hard disk.

The Booter code understands how the file system works and how to load the kernel or a standalone program. When it is ready to go, it prints a colon (:) prompt and waits for either a RETURN or a file name to boot. Typing RETURN enters the file name "unix" from the root device (mw(0,0)) as the file to boot. You can enter the name of a kernel program that you are testing here.

The Booter code locates the file name in the appropriate directory, looks up the inode (file descriptor) and loads the file into memory. The Booter then waits for another RETURN before starting to execute the kernel or standalone program.

The kernel is the part of the operating system program that interfaces between user software and system hardware. User programs can access memory in their own code and data areas, but make requests (system calls) to the kernel to communicate with disks, terminals or other memory. The kernel is never swapped out (it's what performs the swapping).

The kernel initializes itself, prints copyright information, looks for the serial ports (SIO-4 boards) and looks for hard disks. The message "drive timeout" means that the kernel did not receive a response from a hard disk within a reasonable amount of time. Missing hard disks are discovered this way. The kernel then enables interrupts for the first time.


PROM
Tests memory, looks for floppies, loads Booter
BOOTER
Waits for RETURN, loads Kernal
KERNAL
Stays in memory, loads INIT
INIT
The ancestor of all other processes
/etc/inittab
Controls INIT by defining which process to start
Daemons
GETTY
GETTY
/etc/gettydefs
Controls getty by setting terminal baud rates
LOGIN
LOGIN
/etc/passwd
Holds user names, passwords, and shell prog. names
/etc/profile
.profile , .sh
Control sh
SH
CSH
/etc/.cshrc
.login , .cshrc
Control csh
UNIX System V
Boot through Multi-User

The kernel next creates the zeroeth process, the scheduler. The scheduler, as its name suggests, determines which process will be next to run or swap. The scheduler process, not shown in our drawing, runs when the system is idle, when a process goes to sleep and every clock tick. The scheduler, like the kernel, doesn't get swapped out of memory.

The next process to be created is the **init** process. The init process controls the birth of the next level of processes. The processes that init will create (through fork and exec system calls) are defined in the file /etc/inittab. Init looks through inittab (initialization table) every time it is awakened, and initiates or kills processes according to entries in inittab.

When init wants to initiate a login process, it begins by starting **getty**, the program that establishes a connection with a terminal (get tty, get it?). Getty uses the file /etc/gettydefs (getty definitions) to determine the baud rates for terminal lines and line condition information for the terminal. Gettydefs is set up so that getty will change to a different baud rate if getty sees a BREAK or framing error while trying to make contact. Getty also looks for lower case characters. Logging in with all upper case characters causes getty to consider your terminal as upper case only, and sets a flag that converts lower to upper case characters, and vice versa.

If getty succeeds in reading a name, it passes the name it read to the login program. The login program uses the file /etc/passwd to determine valid login names. The login program also requests passwords for login names with password entries. Then, the login program starts a shell program in the home directory for valid login names.

The two most common shells, the Bourne shell (sh) and the C-shell (csh), are also set up by certain files. The Bourne shell uses the /etc/profile file to set up conditions that are shared by all Bourne shell users. Then, the .profile and .sh files belonging to the individual user finish setting up the local environment.

The C-shell uses /etc/cshrc to initialize all C-shells, and the .login and .cshrc files to initialize the local environment. The files /etc/profile, /etc/cshrc, .profile and .login are executed immediately after logging in, and not during subsequent invocations of shells.

Inittab Controls INIT

The /etc/inittab file controls the init program through its line entries. Each line in inittab (short for init table) controls one process that may be initiated by init. These lines contain four types (fields) of information: the name of the entry (id), the run level for this entry, the type of action to be performed and the name of the process to be executed. The line that starts the console shell running in multi-user mode is shown below:

```
co:2:respawn:/etc/getty console co_19200
```

The name of this entry is "co". This is the id that appears in the /etc/wtmp file, processed by the **who** command.

Colons (:) separate each field of information. The second field contains a "2", meaning that this line will be active when the run level is 2. The third field is respawn, meaning that when this line is active (according to the run level), the action taken will be to create the process if it doesn't already exist. The last field is the process to be executed, getty.

To get a better idea about how inittab works, let's look at a diagram that represents the information available to init. The three boxes labeled BOOT, NORMAL and POWERFAIL, in the INIT diagram (next page) represent the three states that init can be in. Init can only be in one state at a time. Init starts out in the BOOT state when it's first loaded by the kernel. Init leaves BOOT state for the NORMAL state when it enters any of the numbered run levels. In the event of a power failure, init temporarily enters the POWERFAIL state, then returns to NORMAL after carrying out any "power" or "powerwait" actions.

Init also keeps track of the current run level and the new run level. The run levels are the numbers from 0 to 6, and the letter s. Only the single user run level, represented by the "s", has a predefined meaning. The system administrator defines the meanings of the numbered run levels when he or she edits /etc/inittab, so their significance is a matter of organization. The system administrator can use one numbered run level to wake up one group of terminals, and use a different run level to include more terminals for login.

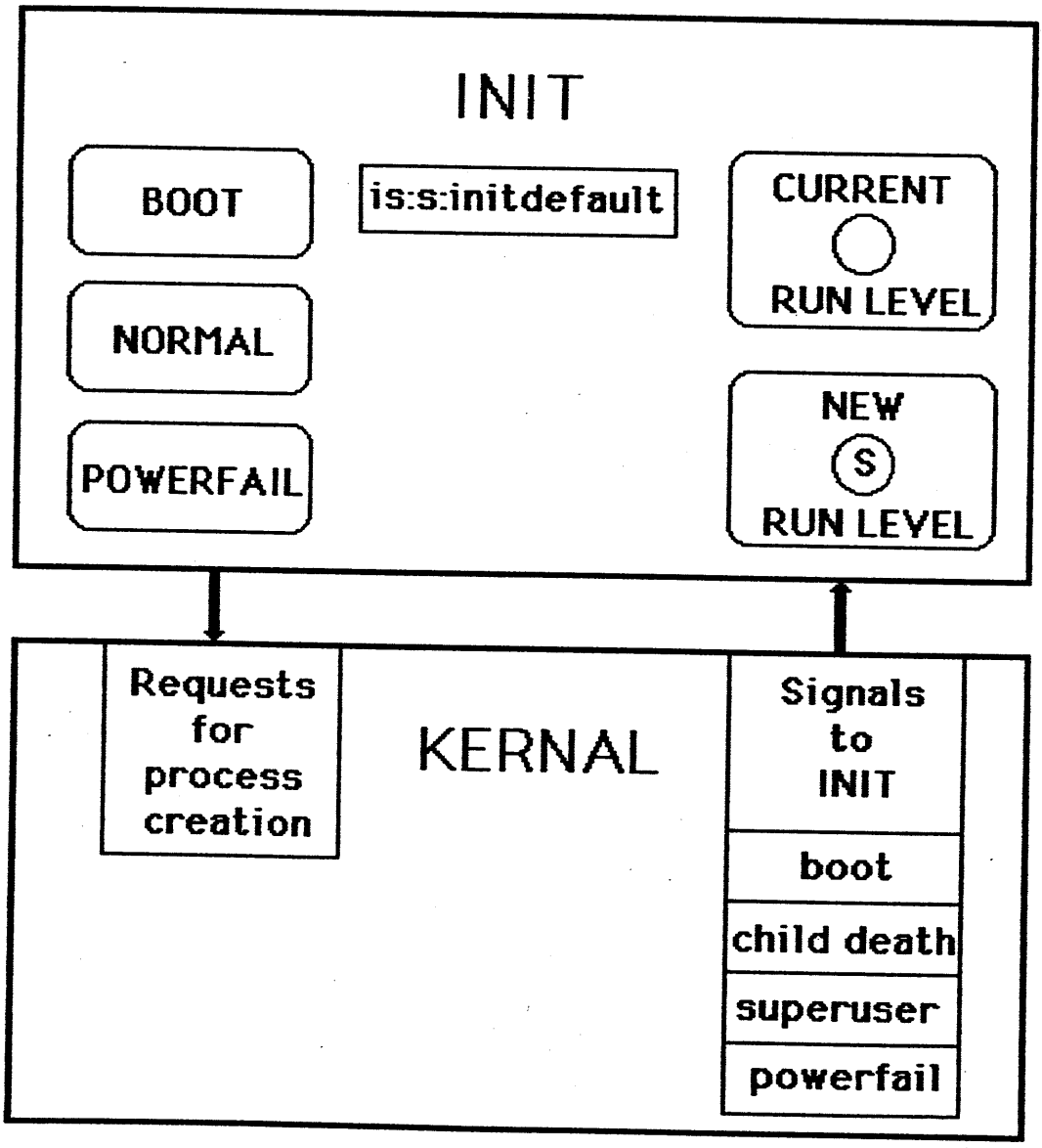
The first line of the file inittab appears in a window in our illustration of init. When inittab is awakened, it examines each line in inittab, always starting with the first line. Init compares the run level field with the new run level. If the new run level matches any of the run levels in the run level field, init follows the action described in the action field. If the run level field does not match the new run level, init signals any living process associated with this line that it has 20 seconds to live, and then kills any process still around at the end of 20 seconds. Then, init looks at the next line in inittab.

The actions that init will take depends on its current state. There are two actions possible while in BOOT state. These actions are triggered when init leaves the BOOT state and goes into a numbered run level the first time:

boot Create the process named in the process field;

bootwait Create the process named in the process field, and wait until it completes (dies) before looking at the next line inittab.

Since init can only leave the boot level once after booting, boot and bootwait actions will only be done once.


INIT
BOOT
is:s:initdefault
CURRENT
RUN LEVEL
NORMAL
NEW
S
RUN LEVEL
POWERFAIL
Requests for process creation
KERNAL
Signals to INIT
boot
child death
superuser
powerfail

After init has entered the NORMAL state, there are four actions that it can follow. Remember, that actions are ignored if the run level for the line doesn't match:

off Kill the process if it exists;

once Create this process once when entering the run level;

wait Create this process once when entering the run level, and wait until it completes before reading the rest of inittab;

respawn Create this process, and restart it every time it dies.

The respawn action is used for initiating the login process. So, when a user logs off, his shell has died and init respawns a new getty-login-shell sequence.

There are two actions that init can perform when entering POWERFAIL state:

powerfail Create this process once; and

powerwait Create this process once, and wait until it dies before continuing.

Your TRICEP system cannot not detect power failure. Even if it could, the loss of power will have erased most of memory anyway. The powerfail state is designed for larger systems where some parts are more sensitive than others to loss of power.

There is one other name that can appear in the action field, called **initdefault**. By including this action in the inittab, init knows which run level to enter when it is started the first time. For example,

```
is:s:initdefault:
```

causes init to start in single user run level, s. This entry appears in the inittab file that comes with your TRICEP, and is the reason why your system comes up in single user mode. You can make your system come up multi-user instead by simply editing inittab and changing the "s" to a number from 0-6. If there is no initdefault entry, init asks at the system console for the run level to use.

Inittab Example

Let's take a look at the inittab file that you got with your system. We've already explained the first line, where the initdefault action sets the initial run level. The next two actions are tied to the BOOT state. The line with the id "bl" will start executing the /etc/bcheckrc shell file when the system leaves single user mode. Since this is a "bootwait" action, no other line in inittab will be processed until /etc/bcheckrc completes.

Notice that the second field, the run level, is empty (two colons together, ::). This indicates that this line is valid for any run level, from Ø to 6, and is equivalent to having the numbers :Ø123456: in the run level field.

```
is:s:initdefault:
bl::bootwait:/etc/bcheckrc </dev/console >/dev/console 2>&1 #bootlog
bc::bootwait:/etc/brc 1>/dev/console 2>&1 #bootrun command
sl::wait:(rm -f /dev/syscon;ln /dev/systty /dev/syscon;) 1>/dev/console 2>&1
rc::bootwait:/etc/rc 1>/dev/console 2>&1 #run commands
co::respawn:/etc/getty console co_192ØØ
ttyØ::respawn:/etc/getty ttyØ co_192ØØ
tty1::respawn:/etc/getty tty1 co_192ØØ
tty2::respawn:/etc/getty tty2 co_192ØØ
tty3::off:/etc/getty tty1 co_192ØØ
tty4::off:/etc/getty tty2 co_192ØØ
tty5::off:/etc/getty tty1 co_192ØØ
tty6::off:/etc/getty tty2 co_192ØØ
```

/etc/bcheckrc handles setting the date and checking file systems. The next line, with the id bc, passes the file /etc/brc to a shell to execute. This file, also with the action bootwait, erases the old mount table (where mounted file systems are logged).

The next two lines have the action "wait" and "bootwait" and an empty run level field, meaning that these lines are valid for all numeric run levels. The line sl removes the old /dev/syscon file and links /dev/systty to /dev/syscon. Since this is a wait action, no other processes are started before this process completes. This is the line that can result in the system console being linked to a remote terminal.
commands

The next line starts the /etc/rc shell script. The rc script (run commands) creates a new mount table, mounts additional file systems and may start various daemons (pronounced like demons). Daemons are independent processes that perform tasks in the background. /etc/rc checks to see if you are entering Run Level 2 for the first time, and starts daemons if you are. The accounting programs, the cron daemon, (used to start processes according to times in the crontable), and the lp daemon (lpsched) can be started now. You need to edit /etc/rc for your system. See the section "Files that MUST be Edited".

The next four lines spawn getty-login processes for the console and three other terminals. Since no run level is specified, getty-login processes will be initiated when any run level except single user is entered. And, because these have the action respawn, any time a user logs out (killing the child of getty-login, the shell), a new getty-login process will be started.

The last four lines of this file have been turned off with the action "off". If you have a second SIO4 board allowing four more terminals, you can change "off" to "respawn" to permit logins on your last four terminals.

Do not turn on (with respawn) these lines if you do not have the capacity for eight terminals, that is, two SIO-4 boards. The message that you will see if you do this will look like

```
getty: cannot open "tty3". errno: 2
```

If you accidentally change these lines, your TRICEP will thrash about trying to open the terminals attached to the non-existent second SIO4 board, and require resetting and the editing of the /etc/inittab file while in single-user mode.

So, this is the inittab file that initially controls init. When you boot up, you will be in single user mode (because of the is:s:initdefault entry). You must signal init with

```
telinit 2
```

to go multi-user. Init then starts the bootwait actions, then the wait actions and finally the respawn actions that bring all the terminals to life.

Modified inittab Example

Now, suppose that you wanted to change inittab to agree with your system's configuration. As an example, let's imagine that you have two terminals, one used as the console and the other connected to port tty1. You also have a modem connected to tty0, and a serial printer connected to tty2. You might want to have independent control over the modem line, so that it is not always on when the other two terminals are on. And, to add a little protection to your system, you might want to come up in multi-user mode instead of single user, so that the superuser is required to enter the correct password after resetting. The edited inittab file that does all this follows.

```
is:2:initdefault:
bl::bootwait:/etc/bcheckrc </dev/console >/dev/console 2>&1 #bootlog
bc::bootwait:/etc/brc 1>/dev/console 2>&1 #bootrun command
sl::wait:(rm -f /dev/syscon;ln /dev/systty /dev/syscon;) 1>/dev/console 2>&1
rc::bootwait:/etc/rc 1>/dev/console 2>&1 #run commands
co::respawn:/etc/getty console co_192ØØ
ttyØ:34:respawn:/etc/getty ttyØ co_12ØØ
tty1::respawn:/etc/getty tty1 co_192ØØ
tty2::respawn:/etc/getty tty2 co_12ØØ
tty3::off:/etc/getty tty1 co_192ØØ
tty4::off:/etc/getty tty2 co_192ØØ
tty5::off:/etc/getty tty1 co_192ØØ
tty6::off:/etc/getty tty2 co_192ØØ
```

Notice that the initdefault run level is now set to 2. Run level 2 is the multi-user run level expected by the /etc/rc file. All of the lines with empty run level fields will match run level 2. The modem, connected to tty0, won't be allowed to login until the superuser changes the run level to 3 or 4. Tty2, with the serial printer attached, has a getty-login process started for it that sets and keeps the baud rate at 1200 baud. The last four lines are still turned off forcing init to ignore these lines.

Sending Signals to init

The superuser sends signals to init by using **telinit** (tell init). Telinit accepts one argument, the new run level, which can be a number between 0 and 6, or the letters s, a, b, c or Q. The superuser could allow a login on the modem (as in our Modified Inittab Example) by sending init a message with

```
telinit 3
```

This changes the run level to three and does not terminate the processes already running at run level 2, since their run level field is empty (matching all run levels). To turn off the modem connection, the superuser can signal init to return to run level 2 with

```
telinit 2
```

The telinit command can also be used to send other signals to init. For example, init can be forced to look at the inittab file by sending

```
telinit Q
```

This is used to get init to look at an edited version of inittab. You can terminate a login process (that was a respawn) by changing the action (respawn) to "off" and waking up init with telinit Q.

The letters a, b, and c are used as pseudo-run levels. Normally, when the run level changes, processes that don't match the run level are killed (gracefully). You can have lines in inittab with the letters a, b, or c for the run level field if you want init to start a process without affecting the current run level or killing other processes.

Processes: A Few Words

Before leaving the topic of init, it may be useful to you to have an understanding of what a process is, how it is created and what its death means. This goes a little outside the scope of this manual, but may be helpful to you.

We have already mentioned that the kernel creates the first two processes, the scheduler and init. What the kernel did to create these processes was to load their program code into memory, initialize their data areas, and set up a special data area known as the upage. The upage is to processes what the inode (file descriptor) is to files. The upage holds information including the user and group number of the process owner, the process id of the parent process, the size of the code and data areas, environmental variables (like HOME, PATH, etc.) and the stack. The upage defines the process: its memory boundaries, owner, and condition.

When init spawns a new process, for example, getty, it follows a two step procedure. First, init makes a system call to the kernel called a fork. The fork system call makes an exact duplicate of init and init's upage with one difference: the new copy of init is called the child of init in the new upage. Then new copy of init makes a second system call, exec. The exec call copies a new program, in this case getty, in the place where the child init was and modifies the upage to reflect the new code and data area. We now have the parent process, init, and a child process of init, getty.

Getty waits for someone to type in a login name. As soon as getty reads a possible login name, getty replaces itself with the login program by using the exec system call. In other words, the child process of init, getty, has been replaced by login. Login checks the /etc/passwd file and asks for a password (if one is required). If the login is successful (a match in the /etc/passwd file), the login program fixes up the upage area with the users environment, and execs the program (usually a shell) in place of itself. Now, the user's shell has replaced login, that replaced getty, that was a child of init. So, the user's shell is now a child of init.

Each time you use the shell to execute a command, the shell forks a new copy of itself, and this copy execs the command. Thus, your shell is still around waiting for the child process to finish executing the command. The shell wakes up when the command finishes (dies). If you execute a command in the background (with &), the shell process doesn't wait for the child to die, but continues independently.

When a process exits for the last time, it dies. For example, when you log out of a shell, the shell dies. Since the shell is a child (indirectly) of init, a signal, child death, is sent to init. The dead child gives up its code and data space, and its upage goes away. Then, init forks a new copy of itself, and execs a getty that waits for the next login.

gettydefs: Sets Up Baud Rates

Init begins the login procedure for terminals by creating getty processes. The getty program attempts to make the initial contact with the user through a terminal. The conditions required by the terminal, such as baud rate, parity, number of bits, are taken from the /etc/gettydefs file.

You may have noticed that getty gets two arguments passed to it in the "respawn" entries in inittab, the name of the terminal line to open and the name of a line in /etc/gettydefs. The line in gettydefs is the first in a chain of entries that specify which baud rate and what line conditions to use. If getty receives a NULL character from the terminal device driver (indicating that the baud rate is incorrect), it will try the next entry in the chain of entries. You can also force getty to change the baud rate by typing the BREAK key.

There are five fields in each line of the gettydefs file, with each field being separated by the "#" character. A blank line separates entries. The five fields are:

label	The name used as an argument by getty to find a line in gettydefs, like co_19200;
initial flags	The line conditions used when contact is first made, usually simply a baud rate, e.g., EXTA;
final flags	The line conditions given to the terminal before getty turns over control to login, (TABS, SANE);
login message	The message that the user sees on his screen when he is prompted for a login name; and
next label	The next gettydefs line to use if the connection fails at this baud rate.

The gettydefs file that comes with TRICEP systems follows.

```
co_19200# EXTA # EXTA SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_9600

co_9600# B9600 # B9600 SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_4800

co_4800# B4800 # B4800 SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_2400

co_2400# B2400 # B2400 SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_1200

co_1200# B1200 # B1200 SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_300

co_300# B300 # B300 SANE TAB3 #
    \r\n\nUniplus+ System V\r\n\nlogin: #co_19200
```

Please notice that each entry is linked to the entry that follows by the next label field. The last entry, co_300, is linked to the first entry, co_19200, forming a closed circle. Thus, getty can attempt to make contact at each of the following baud rates:

19200, 9600, 4800, 2400, 1200, 300

If you want to change this chain, you can edit the next label field. The baud rates and line conditions honored by getty are defined in the manual entry for termio(7) in the System Administrator's Manual.

The final flag SANE has a special meaning not mentioned in termio(7). SANE is a set of "normal" line conditions: 7 data bits, 1 stop bit, even parity, echo on, output processing on, process backspace and kill characters, etc. You can view the complete list with the commands

```
stty SANE
stty -a
```

which displays all the terminal settings. The EXTA flag stands for 19200 baud (which is not standard UNIX).

The \r and \n are special characters that are translated into a carriage return and line feed, respectively. You can customize your TRICEP system login prompt by changing the message in the login message field. For example,

```
co_19200# EXTA # EXTA SANE TAB3 #
    \r\n\nTRICEP System V\r\nuser name? #co_9600
```

produces the login message

```
TRICEP System V
user name?
```

You can create gettydefs entries that are unique for each login port if you like, with special login messages. Or, you can identify a particular TRICEP if you are using several by editing the login message field.

etc/passwd Controls login

The login program is executed by getty when getty has collected a possible login name. The login program reads the /etc/passwd file and searches for a match to the name passed to login by getty. Login next requests a password if a matching entry has an encrypted password. Login also asks for a password if there is no matching entry for the name in the /etc/passwd file. This is designed to foil attempted system breakers who are trying random names as possible logins. Login exits (and dies) if a matching /etc/passwd entry is not found, and init starts a new getty process to wait for the next login.

If login has successfully matched a /etc/passwd entry, it uses the entry to set up the user's environment. The environment is part of the process upage. login changes the user and group numbers in the environment (and of the controlling terminal) to match the entry's user and group numbers. The HOME variable is set to the home directory field in the entry. Finally, the program that is the last field of the entry is exec'ed, replacing login. Usually this program is a shell.

The /etc/passwd file that comes with your system has only three users that you can log into: root, rootcsh, and guest. root and rootcsh are the superuser login names. You need to add a password to both these names immediately. Type the command

passwd root

while logged in as root (or rootcsh) and add a passwd if you have not already done so. Passwords should be at least 8 characters long and have some non-alphabetic characters in them. For example,

```
fog56@end       holy/fat        .to.be.30
```

are acceptable and easy to remember passwords. Your last name, nick name, wife's or boyfriend's name, are not good passwords. The root password is very important. Don't forget it! And don't share it with everyone, but give it only to people who need to know. You should also add the same password to the rootcsh (or root) login name at this point by typing

passwd rootcsh

and adding the password.

The guest entry is the example entry in the /etc/passwd file. It allows an ordinary user to log in as guest in the /users/guest home directory. This login name also does not have a password. Passwords are necessary on UNIX systems that require any security.

Editing the /etc/passwd file is explained in detail later.

Controlling Shells: /etc/profile, /etc/cshrc

If the login program execs a shell program, more variables are added to the environment. Each shell program reads a shell script in the /etc directory that contains commands to set up the environment. Every user's shell is thus initially set up by these files. The user has no control over this, so that system administrator can give all users the same initial environment. The users can override this initialization by explicitly resetting these variables in their own set up files.

/etc/profile	Initializes the environment of all login Bourne shells (sh)
/etc/cshrc	Initializes the environment of all login C-shells (csh)
$HOME/.profile	The users own initialization script for the Bourne shell (sh)
$HOME/.cshrc	The users own initialization script for the C-shell (csh)

One of the things the a system administrator can do with /etc/profile and /etc/cshrc is to establish a system wide umask. The umask is used to mask out certain permission bits when new files are created. For example, a umask of 022 prevents write permission from being automatically granted to the Group and Others when new files are created. The user can override the umask by setting his own umask in .profile or .cshrc. Examples of user configuration files are contained in the /usr/lib directory. These files are called: .profile, .setup, .login, .logout, .cshrc and .dosrc.

Files You MUST Edit

When you get your TRICEP system, it contains an exact copy of the files on the master hard disk at Morrow. What you need to do is "personalize" your system. You need to add user names, the local time zone and a system name. Morrow has provided two menu-driven programs to help you, but you will still need to edit two files yourself: /etc/sys_id and /etc/rc. The next three sections explain exactly what to do to personalize your system.

Setting Your Time Zone in /etc/rc and /etc/profile

TRICEP systems are manufactured in California, so the Time Zone in your TRICEP is initially set for Pacific Time Zone. This is great if you live in California or Oregon, but no help if you live in Minneapolis, Annapolis, Budapest, Timbuktu or London. The **date** command looks at the environment variable TZ and corrects the display of the time and date so that it reflects local time **after** you have changed TZ to your local time zone. The **date** command also knows about: leap years, the International Date Line and Daylight Savings Time. All that you have to do is set TZ.

The TZ variable initially looks like PST8PDT. There are three items of information encoded here. The first three letters are displayed by **date** as the Time Zone identification during Standard time. The last three letters are displayed during Daylight Savings Time. If you live in Minnesota, which doesn't use Daylight Savings Time, you won't include the last three letters. The **date** command knows which Sunday to "spring forward" or "fall back", and does so if you supply the three identifying letters after the Time Zone.

The "8" in the middle stands for 8 hours earlier than the time at Greenwich, England. Unless you are a navigator or world traveler, this may seem a little strange, but the **date** command treats Greenwich Mean Time as Ø, Eastern Standard Time as 5, Central Standard as 6, Rocky Mountain as 7 and Pacific as 8. On a map of the World, each Time Zone is counted, starting with Ø, as you move West.

For countries East of Greenwich, you count from Ø backwards: all of Europe (except Portugal, Greece, Rumania, Bulgaria and Great Britain) is -1, Egypt and Greece are -2, Moscow and Baghdad are -3, Hanoi and Chunking are -8. Countries using fractional Time Zones aren't supported (and neither is Saudi Arabia, which is on Solar Time, and Mongolia, which has no legal time). Here are several example Time Zones, as you would edit into your /etc/rc file:

```
TZ=GMTØ         Great Britain
TZ=EST5EDT      Eastern Standard Time/Eastern Daylight Time
TZ=CST6CDT      Central Standard Time/Central Daylight Time
TZ=RMT7         Rocky Mountain Time/no daylight time
TZ=PST8PDT      Pacific Standard Time/Pacific Daylight Time
TZ=AST-1Ø       Australia (Sydney/Melbourne)
```

To edit your /etc/rc file, and change TZ to be EST5EDT, for example, enter the characters that appear in **bold print** below:

```
# ex /etc/rc
"/etc/rc" 41 lines, 851 characters
:/TZ/
TZ=PST8PDT
:c
TZ=EST5EDT
.
:w
"/etc/rc" 41 lines, 851 characters
:q
# []
```

Repeat the steps above, using **ex /etc/profile** as the first command.

For other Time Zones, substitute your own identifying string of three characters, the Time Zone number, and the optional Daylight Savings Time identifying string.

The /etc/rc file also starts the various system daemons. For example, the line

```
/etc/cron
```

wakes up the **cron** daemon. This daemon wakes up once a minute and looks at the file **/usr/lib/crontab**, and executes a shell for any commands that are scheduled in this file. The periodic appearance of the date on the console is done by **cron**. Lines in both /usr/lib/crontab and /etc/rc may be commented out by putting an "#" (crosshatch) as the first character of a line.

Process accounting is started by removing the crosshatch on the line

```
/bin/su - adm -c /usr/lib/sa/startup
```

This will compile information on program use, and user times. Several lines in crontab, all executing files in the **/usr/lib/acct** directory, process accounting information during the wee hours of the morning.

```
/bin/su adm -c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d` &"
```

starts system accounting. This allows you to monitor activity at the system level (kernel table usage, swap usage, read and write cache hit rates, etc.). The last four lines in crontab, all executing files in the /usr/lib/sa directory, take snapshots of the system hourly and provide a daily summary of system activity.

System Name

Your system name is initially "morrow". This name is built-into the kernel file, **/unix**, and is used by the **uucp** programs. The **uucp** programs are used for inter-system mail and possibly for getting bug-fixes or software

through phone lines. But, you need to provide identification for your system.

System names are a maximum of 8 lower case letters, and a minimum of 2 letters. No upper case letters, punctuation or numbers are permitted. The **/etc/sysname** program patches the system name directly into the kernel file and the kernel loaded into memory. An example of using **sysname** is provided below:

```
# /etc/sysname
Your system's name is "morrow".
Do you want to change it? y
New system name: hal
# []
```

Now, you system's name is "hal", or whatever. You also need to edit the file **/etc/sys_id**, that is used by the **take** command. Edit this file, following the example for editing **/etc/rc** for the Time Zone, including your new system name. There is only one line in this file, so change it.

Adding User Names

There is an entire section a little later in this manual that goes into great detail about adding User Names. This information is still valid, but there is now an easier way to add User Names. The **newuser** command creates entries in **/etc/passwd**, creates new home directories as subdirectories of **/users**, and copies prototype configuration files from **/usr/lib** to the new home directory. **newuser** also handles adjusting user and group ownership of the new HOME directory and files. For example, to add the User Name rik, follow the following example:

```
# newuser
User's real full name: Rik Farrow
New user's login name: rik
New user added.
# []
```

Easy, eh? The drawbacks to using **newuser** are: everyone is in the same group (100), everyone gets a HOME directory named **/users/"name"** and everyone gets the C-shell. This really isn't much of a problem, unless you want to do something unusual. For that, refer to the section entitled "/etc/passwd: the User Database". For now, please use **newuser** to create User Names for your possible users. You should also add passwords for these new users, as in

```
# passwd rik
```

and try logging in. Any system that requires any security at all must also have passwords for all.

One more thing about **newuser**. Using this program locks the **/etc/passwd** file until it completes. Locking the **etc/passwd** prevents anyone from logging in, or using any command that uses the file, like **find** or **ls**.

Connecting Additional Terminals

This section should be easy for you to follow, since you have successfully connected the console. The TRICEP system can have three to seven additional terminals, depending upon whether you have one or two SIO-4 boards. A quick glance at your back panel is all you need to ascertain this. Systems with one SIO-4 board have four DB-25 female connectors (like the letter D lying on its rounded side). Systems with two SIO-4 boards have 8 DB-25 female connectors.

Terminals are connected in the same way as the console was. The instructions for doing this are included in steps 5 and 6, in the Installation part of this manual. There are two differences that you need to be aware of. The first is rather obvious, that is, you will be connecting each additional terminal to a DB-25 connector labeled tty0 through tty6. Only the console is connected to the connector labeled console. (See the illustration on page 16.)

The second difference is that additional terminals may be setup to work at different baud rates. Remember that the console must be set to 19200 baud. You can use other baud rates for your additional terminals, but 19200 is recommended. If you want to use a different baud rate for a terminal, you need to edit the line in the /etc/inittab file that starts the getty-login-shell process for it. For example, to make the terminal connected to tty2 login at 9600 baud, you would change 19200 to 9600 in the line starting with tty2:

```
tty2::respawn:/etc/getty tty2 co_9600
```

Remember that co_9600 is the label of a line in gettydefs and that the BREAK key can be used to change baud rate at login.

/etc/passwd: The User Database

/etc/passwd is the user database. All the valid user names for the system are kept here. Programs that use user names, like ls, chown and find, search /etc/passwd to cross reference user ids (a number) and user names. Editing the /etc/passwd file is an essential part of system configuration. Remember that you can use the **newuser** command, described in the "Files You MUST Edit" section.

The next two sections go into the details of adding and removing users. There is more to this than just editing an entry in /etc/passwd. It is also possible to create special users by having passwd entries that execute a restricted shell (rsh or uushell) or some other program than a shell.

Adding New User Names

The superuser is responsible for adding new users to a system. The superuser decides where the new user's home directory will be and selects user and group numbers. Then, the superuser:

- Edits the /etc/passwd file and adds an entry;
- Edits the /etc/group file and adds the user name to the groups it is a member of (optional);
- Makes the home directory, changes its ownership and adds a login profile;
- Tests the new user login.

These steps are all that is necessary to create new user accounts.

/etc/passwd, The User Name File

The /etc/passwd file can be read by anyone, but it can be written to only by the superuser. This arrangement allows people to search the password files for their own and other user's entries. Certain commands, like **ls** with the l option, or **find**, use the passwd file and substitute user names for user numbers.

Editing the passwd file is restricted to the superuser because anyone who can edit the passwd file can give themselves superuser powers. The superuser can access, change or erase every file in the file system. It is necessary to have the superuser so that the file system can be maintained. It is also necessary to restrict access to the superuser to avoid accidental (or malicious) damage to the files system.

Every entry in the /etc/passwd file has seven fields for information. They are:

- The user name, the name typed in during login;
- The password; when an entry is added, the password field is left blank and the **passwd** command is used to add the password;
- The user id number, a unique number between 1 and 65235 assigned to the user name;
- The group id number, a number (1-256) shared with others in the same group; members of the same group have special access permission for other members files;
- A comment field; often the user's name is put here;
- The pathname of the HOME directory; the directory assigned to the user for login;
- The pathname of a program to execute; usually, the shell is executed, and the Bourne shell is the default.

The information in password entries is separated by colons (":"). Two blank colons appearing together indicate a null field. Here is an example entry from a password file:

rik:A5hjpØvnvh9up:34:2:rik farrow:/users/rik:/bin/csh

User names can be up to eight letters long. The name must be in lower case letters, so that getty doesn't consider the terminal to be upper case only. Usually, people choose their own user names, and base them on their real names or nicknames.

Passwords appear after the user name. This field is left empty (two colons together, ::) when the entry in the passwd file is created. The **passwd** command is used to add a password in an encrypted form.

The superuser or the user can assign a password using the **passwd** command after editing the /etc/passwd file. In systems that require any security at all, passwords are mandatory except for restricted users, like who, or uucp.

The two numbers that follow the password are the user and group id. These numbers are used to check the user's file access permissions whenever files are accessed. Every one sharing the same group id number, 2 in this example, has the same group access permission. In the same way, if two user names have the same user id number, they will have the same user permissions. This is not a good way to share permissions. User numbers should be unique. The group id number is provided for sharing file access permission.

The information following the user and group numbers is the comment field, in this case, a full name. Another use for this space is for comments, such as the user's location and phone number, or the purpose for the password entry.

The next field is the pathname of the home directory. When the login process is complete, the user finds himself in this directory. This information is also assigned to the environment variable HOME during login. Anytime the **cd** command is used without arguments, the current directory is changed to the HOME directory.

The last field is the program to execute when the user logs in. The program executed is, by default, the Bourne shell. In the example, "rik" logs in with the C shell instead. In the following example of your passwd file, several more programs to execute are shown.

```
# cat /etc/passwd
root:v2xDWX6hxB5J6:Ø:Ø::/:/bin/sh
rootcsh:cHBjPcMVBi36Q:Ø:Ø::/:/bin/csh
daemon:xxxxxxxxxxxxx:1:1::/:
bin:xxxxxxxxxxxxx:2:2::/bin:
sys:xxxxxxxxxxxxx:3:3::/bin:
adm:xxxxxxxxxxxxx:4:4::/usr/adm:
uucp::5:5::/usr/spool/uucppublic:/usr/lib/uucp/uushell
check:xxxxxxxxxxxxx:6:6::/:
lp:xxxxxxxxxxxxx:7:7:/usr/spool/lp:
who::22:1:who command:/bin:/bin/who
guest::1ØØ:1ØØ:guest account:/users/guest:
```

The uucp user name is used by the UNIX to UNIX copy program. Logging in on this account executes the **uushell** program, a restricted shell. This command expects to receive input from a similar shell on a remote UNIX system. Access to the local system is controlled by forcing the remote system to log as a restricted user.

The restricted user "who" displays the output from the **who** command and then logs out. The guest account allows password free access to the system and pose potential security hazards. Guest, with the empty command field, will get the default Bourne shell.

Editing /etc/passwd

The superuser adds new users by editing /etc/passwd. It is a good idea to work on a backup copy of special files, such as /etc/passwd, instead of editing them directly. For example,

```
cp passwd passwd.bak
```

provides an immediately available copy of the passwd file for editing. The following example illustrates the addition of a user to /etc/passwd.bak:

```
$ su
Password:
# cd /etc
# cp passwd passwd.bak
# ex passwd.bak
"passwd.bak" 23 lines 567 characters
a
ralph::223:1ØØ:Ralph Minkler:/users/ralph:/bin/sh
.
:x
"passwd.bak" 24 lines, 6Ø2 characters
# []
```

When you are satisfied with the edited **passwd.bak** file, use it to replace the old copy of **passwd:**

```
# chmod 644 passwd
# mv passwd.bak passwd
```

Before replacing the old passwd file with our edited version, passwd.bak, it is necessary to change the file access permission for the passwd file to read by all and write for owner only (644). Then, passwd.bak can replace passwd. The passwd command strips away write permission from /etc/passwd every time the command is used.

At this point, a new user name, ralph, has been added to the system. Ralph still needs to have his HOME directory created before he can log in. He can also be added to the list of users in his group.

Before continuing, it is a good idea to assign a password to ralph. Entries without passwords in the /etc/passwd file pose threats to system security. Having a password protects you from anyone, whether it is your 1Ø year old whizz kid, or the employee you just fired. Only the superuser and the user his self can change (or add) a password. Here is the way the superuser adds a password:

```
# passwd ralph
Password:
Retype password:
```

Notice that the password does not appear while it is typed. This prevents someone from seeing the password on the screen. If the superuser or the user forgets the password, the superuser can always add a new password.

Also, the passwd command automatically strips write permission from the passwd file. This includes write permission even for the superuser. This means that the superuser must take the extra step of adding write permission (with chmod) before replacing the old password file.

Editing /etc/group

The /etc/group file contains entries similar to the entries in /etc/passwd. Editing this file is not essential. The purpose for this file is to allow programs like **ls** to substitute group id numbers for names.

Each item of information is separated by colons, and a password may be present. There are four types of information in /etc/group:

- The group name,
- An encrypted password,
- The group id number and
- The list of user names that can use the group id number separated by commas.

When /etc/group is edited, the superuser adds the user name to the entry for the selected groups. The user name is always added to the entry for the group id number used in the user's entry in /etc/passwd. In Ralph's entry,

ralph::223:1ØØ:Ralph Minkler:/users/ralph:/bin/sh,

the group id number is 1ØØ. This is the group id normally associated with ralph. Ralph may have access to other group id numbers if his user name is added to the list following the group id number. The **newgrp** command is used to change group id number. In the example that follows, ralph is added to his own group list, and to an additional group id number:

```
# ex group
"group" 3 lines, 169 characters
:%
users::1ØØ:admin,guest,guest1,kevin,rik:
sys::1:root,drivers,stand:
staff::2ØØ:bob,kevin,splen,len,rik,kevinb,bill:
:s/bill/bill,ralph/
staff::2ØØ:bob,kevin,splen,len,rik,kevinb,bill,ralph:
:1s/rik/rik,ralph/
users::1ØØ:admin,guest,guest1,kevin,rik,ralph:
:x
"group" 3 lines, 181 characters
# []
```

The /etc/group file allows selected users to be members of multiple groups. When ralph logs in, he has the group id number 1ØØ. Because Ralph's name is included in the "staff" entry, ralph can also have the group id 2ØØ assigned to her:

newgrp guest

The password field of /etc/group entries is usually null. If a password is present, a user whose name is not in the entry can attempt to access the group id number by using the **newgrp** command and typing the password when requested. The user attempting to change to a group id without a password or the user name on the list gets the message:

Sorry.

The /etc/group file is not required for the creation of a new user name. /etc/group is provided as a means for switching group id's. There is no easy way to add a password to the entries in /etc/group.

Creating the New HOME Directory

The superuser makes a directory for the new user with the pathname in the HOME directory field of the /etc/passwd entry. The HOME directory should be part of the same file system as the rest of the user's group.

There are three things to be done in setting up a new HOME directory:

- Making the directory,
- Copying a login profile to it and
- Changing the user and group ownership.

Here are the commands used to create a new HOME directory for ralph in the directory /users, the directory that has been set up for users" HOME directories:

```
# mkdir /users/ralph
# cp /.profile /users/ralph
# chown ralph /users/ralph /user/ralph/.profile
# chgrp users /users/ralph /user/ralph/.profile
# []
```

Changing the ownership (**chown**) of the new directory and the files within it is an essential task. Otherwise, ralph may log in to find that he does not have permission to write in his own directory, or edit any files.

The file **.profile** is used as the configuration file by the shell. It is not essential to have this file, but it does provide an example of a configuration file that can be edited by the new user. The file **.login** is used by the C shell.

Now, if everything has been done correctly, you should be able to log in as ralph. New user names are always tried by the superuser before they are given to the new user to check for mistakes.

```
# su ralph
Password:

$ ls -al
drwxr-x--- ralp root         96 Oct  3, 1984 .
drwxr-x--- bin  root        214 Aug 18, 1984 ..
-rwxr-x--- ralp root         27    Oct  3, 1984 .profile
$ []
```

Notice that the owner of "." and the .profile file is now ralph.

Removing Users

In the course of ordinary events, people move on. You may find that your list of users has changed requiring the removal of some names from /etc/passwd. We actually advise against the immediate removal of user names from /etc/passwd. What you can do instead is to give them an impossible password to prevent them from logging in.

The reason for keeping old user names around is that files are identified by their user numbers. Then, programs look in /etc/passwd and cross reference the user number to a user name. If you have removed the user name from /etc/passwd, you will be faced with a mysterious number instead of a recognizable name in long directory listings (ls -l), for example. So, here is the way to edit /etc/passwd to prevent old users from logging in:

```
# cp passwd passwd.bak
# ex passwd.bak
"passwd.bak" 23 lines 567 characters
/ralph
ralph:hd64kghnd3hgG:223:2ØØ:Ralph Minkler:/users/ralph:bin/sh
c
ralph:deleted:223:2ØØ:Ralph Minkler:/users/ralph:/bin/sh
.
:x
"passwd.bak" 24 lines, 6Ø1 characters
# []
```

The word "deleted" in the password field makes logging into the user "ralph" impossible. Programs can still identify files using ralph's name because the entry still exists. This is why the /etc/passwd file is referred to as a user database: some information about users is kept here.

You might also want to remove ralph's files from the file system. Presumably, you will want to backup the files first, in case there is something of value there. The following commands back up the files beginning in ralph's HOME directory, then delete all the files:

```
# tar cvfB /dev/djØa 798 /users/ralph
# rm -fr /users/ralph
```

The tar command requires that you have inserted a formatted floppy diskette, and have enough formatted diskettes ready to complete the backup. This was explained earlier in the section on First Time Use.

/etc/termcap: Terminal Capabilities

Some of the UNIX commands take advantage of intelligent terminal capabilities, like cursor addressing and line insertion. The editors **ex** and **vi** both use this information, as do some games. The information about terminals is kept in the file /etc/termcap, short for **term**inal **cap**abilities.

The /etc/termcap file that you have was edited so that terminals manufactured by Morrow are presented first. The reason for this is that much time is wasted by tset, vi or ex scanning through this lengthy file looking for specific entries. If your terminal is not included near the beginning of /etc/termcap, edit the file /etc/termcap, find the lines that support your terminal and move them to the beginning /etc/termcap file. The script that follows shows an editing session that "clips" out the entry for a Soroc 12Ø and moves it to the head of the termcap file.

```
ex /etc/termcap
"/etc/termcap" 1117 lines, 49225 characters
:/soroc
MI|soroc|Soroc 12Ø:\
:nu
   312 MI|soroc|Soroc 12Ø:\
:313,315nu
   313     :cd=\EY:ce=\ET:cl=2\E*:ma=^K^P^R^L^L:\
   314     :kl=^H:ku=^K:kr=^L:kd=^J:tc=adm3a:
   315 Ma|aa|annarbor|ann arbor:\
:312,314mØ
:w
"/etc/termcap" 1117 lines, 49225 characters
:q
```

Entries for terminals in /etc/termcap begin with a line of possible names. This line of names always starts in the leftmost column. The line beginning with "Ma|" marks the start of the next entry. What you need to do is find your entry, discover the line numbers for your entry (312 through 314 in this example), and move these lines to the beginning of the file. This puts them ahead of everything (including several thousand bytes of comments), and helps make your system more efficient.

Connecting Printers

The TRICEP system supports both parallel and serial printers. Since having a serial printer uses one of the possible login ports, you may wish to use a parallel printer. Other than this, there is no particular advantage to having one type of interface or the other.

The difficulty in adding either type of printer is usually in the cable. We can tell you exactly what your TRICEP system produces in the way of signals on the serial and parallel ports. You must match this up with what your printer needs and produces.

Serial Printers

Serial printers use the bit at a time method for communicating. The ends of a serial cable have DB-25 connectors on them, just like terminal cables. The required pinout on the cables AT THE TRICEP END is exactly the same as for terminals:

pin 2	Transmit data
pin 3	Received data
pin 7	Signal ground
pin 2Ø	Data terminal ready

Pins 2, 3 and 7 will be the same at the printer end. However, pin 2Ø might need to be connected to some other pin at the printer end.

The various different printer manufacturers use any of a number of pins to signal when their printer is not ready. For example, NEC Spinwriters use pin 19. If you hook up the wire from pin 2Ø at the TRICEP end to the correct one of these pins, everything will work fine. Your task is to read your printer manual and decipher which pin holds the printer ready signal. Most printers (for example, Epson, TI, DEC, etc.) use pin 2Ø.

The SIO-4 board has a built-in interlock, not controlled by software, that prevents it from sending characters unless it sees Data Terminal Ready on pin 2Ø. The Data Terminal Ready signal is logic true, meaning that when the voltage on pin 2Ø is high (between 5 and 2Ø volts), Data Terminal Ready is true, and transmission can proceed.

Your printer may use X-ON or X-OFF handshaking to turn off and on transmission. This will work fine with the UNIX software, but pin 2Ø must still be true for the SIO-4 board to work correctly.

When you have connected your printer, you can test it with a couple of simple commands. First, you need to prevent init from trying to login your printer. This was explained in the section called Modified Inittab Example. Essentially, you need to edit the /etc/inittab file, change the entry that corresponds to your printer from "respawn" to "off", and signal init that you have edited the /etc/inittab file with

```
telinit Q
```

Next, you need to set up the correct baud rate for the port. Suppose, you have connected your printer to port tty2 (and have edited this line in inittab). Let's assume that your printer is set up to work at 12ØØ baud, a nice fast baud rate. You set up the TRICEP to work at 12ØØ baud on port tty2 by typing

```
sleep 36ØØ < /dev/tty2 &
stty 12ØØ < /dev/tty2
```

The **sleep** command opens the **/dev/tty2** and keeps it open for an hour (or until you reset). This is important, because the SIO4 driver will reset the baud rate to the default, 192ØØ, when the device, /dev/tty2, is closed. Don't forget the ampersand (&) that puts the sleep in the background.

If you get back the message "Permission denied", it is because you must be logged in as superuser (the # prompt) to edit inittab, use telinit, or to change the baud rate of a port that you aren't connected to. You may also need to change the file access permission mode with chmod, as in

```
chmod 6ØØ /dev/tty2
```

With the baud rate correctly established, try this command to send a test message to your printer:

```
echo This is a test of the... > /dev/tty2
```

The line up to the greater than sign should appear at your printer. If it doesn't, you need to check the power to the printer, that the printer is ON-LINE and that it has paper ready. When all these things are true, but your test fails, you have a problem in your cable. Make certain you are using the correct port and device name. (Look at the drawing in the earlier section entitled Adding Additional Terminals). If everything else checks out okay, your problem is probably with pin 2Ø and the printer ready pin.

When you have your printer working correctly, go on to LP, the Print Spooling Facility.

Parallel Printers

TRICEP has a Centronics style printer interface for parallel printers. The connection for this printer is located just above the connection for ttyØ (see the illustration on page 16.) The cable enclosed with the TRICEP is connected here **ONLY WHILE THE TRICEP IS OFF.** Connect the cable so that the cable drops down without being folded over the connector. The red stripe will be on the left (when viewed from the back). The other end of the cable can be connected to printers with Centronics style interfaces.

The name of the parallel port is /dev/cent. Connect your printer, turn the power on to it, get its paper ready and make it ON-LINE. Then, you can test out your hookup with

echo This is a test... > /dev/cent

This should print the line up to the greater than sign on your printer. If it didn't, check your printer's setup, and your cable connections. If these things don't seem to help, compare the signals your printer requires (as outlined in your printer's manual) with the signals your TRICEP is sending. The connector pin 11, BUSY, may be the culprit.

lp: Print Spooling Facility

The lp command arranges background printing of files in the UNIX system. There is considerable flexibility allowed with lp, meaning that you must configure it so that it will work with your system. Some of the things lp can do are:

- collect files for later printing,
- add banners and other information to each job printed,
- send files to a default printer,
- send files to the first available printer of a group of printers, and
- be stopped and restarted with simple commands.

Before you can use lp, you must configure your system with the lpadmin command. The lpadmin command can only be used by the superuser, and only when the scheduler for lp, lpsched, is not running. The next section explains how to configure your system with lpadmin and add lpsched to your /etc/rc file. The section that follows outlines the basic use of lp and the enable and disable printer commands. The information presented here is based on the Section in the System Administrator's Guide labeled LP (and lots of experimentation).

LPADMIN, the lp Configuration Command

Your TRICEP system comes configured to use lp with a centronics interface printer connected to /dev/cent. This printer has been set up as the default printer, and is called mp3ØØ. You still need to read the instructions in this section to understand how lp works. You also need to add several lines to the /etc/rc file to make starting of the lp printer daemon automatic with going multi-user.

You can follow the instructions in this section if you want to change how lp works. If you want to change your configuration after lpsched has been started, you must halt lpsched with lpshut.

The lpadmin command allows you to

- create printer names and associate them with ports,
- select the interface, a program that controls the printing of characters,
- group printers into classes, so that files may be sent to any one of a group of printers, and
- establish the name of a default printer destination.

The first bit of funny business about lp is that the commands lpadmin, lpshut and lpsched are all located in the /usr/lib directory. This is because these commands will only work for the system administrator, not ordinary users. Practically speaking, these commands won't work unless you add the /usr/lib directory to your search path. This may have already been done for you, but you can check with the command

```
echo $PATH
/etc:/bin:/usr/bin:/usr/lib
```

Each directory name in the path is searched whenever you type a command to the shell. In our example, /usr/lib is the directory that is searched last. If /usr/lib does not appear in your PATH, you can add it temporarily by typing

```
PATH=/etc:/bin:/usr/bin:/usr/lib
```

and permanently by adding this line to the /.profile file. You can also prepend the directory name, /usr/lib, to all your commands, as in /usr/lib/lpadmin, but this is awkward. Or, change directory to /usr/lib and prepend ./ to all your commands.

Okay, before setting up lp with lpadmin, you need to select a name and an interface for your printer, or printers. For a name, select something short but easily remembered. For example, mp3ØØ or daisy are good printer names and spinwriter or p1 are not so good (too long and unclear).

The interface program can be a shell script or a C-program. The task of the interface is to prepare the printer, print an identification of the job being printed (if desired) and to send the characters from the files being printed to the standard output. lp handles routing the characters to their correct destination. The directory /usr/spool/lp/model has several model interface programs. These programs are shell scripts, and can be modified to fit your own requirements. There is a separate directory used for modified models, /usr/spool/lp/interface, that you can copy your modified models to.

The interface program or shell script is passed a list of arguments. These arguments can be used to create the identifying title for the print job. The filenames to be printed are included in this list of arguments and MUST be used by the interface program. The arguments are:

```
arg[Ø]    the request id given when lp receives a request,
arg[1]    the user name of the user who made the request,
arg[2]    the (optional) title of the request,
arg[3]    the number of copies of the request to print,
arg[4]    the options that can be used by the interface
          program, specified by the user, and
arg[5-n]  the full pathnames of the files to be printed.
```

If all that your interface program does is print the files without an identifying page, you could use the simple shell script:

```
shift 5
files="$*"
for file in $files
do
/usr/bin/xtab "$file" 2>&1
echo "\n\Ø14"
done
echo "\Ø14"
exit Ø
```

This script skips the first five arguments (request id, user name, title, number of copies and options) with the five shifts. Then, it prints the file (xtab "$file"), sending any error messages to the error device. Next, the echo "\n\Ø14" sends a newline and a formfeed, and the next file, if any, is printed. Then, a final formfeed is sent. This script is named /usr/spool/lp/model/mp3ØØ. The xtab program, in /usr/bin, expands tabs into the correct number of spaces to reach the next tab stop. Tab stops are every 8 spaces. The ability to expand tabs is built-into the character devices, and has the name "opost" (see termio(7) in the Administrator's Manual).

If you are using a serial printer, you need to set the correct baud rate, like

```
stty 12ØØ < /dev/tty2
```

before sending any characters to the printer. There is a catch to this, as explained earlier. The SIO4 driver resets the baud rate to 192ØØ whenever the device is "closed". Thus, setting the baud rate with **stty** appears to fail because the baud rate is immediately reset to 192ØØ when the stty command completes (and the standard input is closed). The solution is to keep the serial port from being closed by using a sleep command in the background. For example,

```
sleep 864ØØ < /dev/tty2 &; stty 12ØØ < /dev/tty2
```

keeps the baud rate for /dev/tty2 set to 12ØØ baud for 24 hours. You can add this line to your **/etc/rc** file, right after the lines that activate lpsched. Of course, if you never reset your system, or go more than 24 hours without going single user and backing up, this won't work. In this case, you can add a line to **/usr/lib/crontab** that performs a sleep once a day, as in

```
Ø 6 * * * /bin/sleep 8643Ø </dev/tty2 &; /bin/stty 12ØØ < /dev/tty2
```

This keeps /dev/tty2 open for 24 hours and 3Ø seconds (better to allow for a little slack), starting at 6:ØØ in the morning. Or, use an /etc/inittab entry to keep the device open (see Modified Inittab Example).

The model named /usr/spool/lp/model/dumb provides a much more complex example of an interface program. Dumb provides an identifying page before each request is started, complete with the user name in a banner. This model also expects a 132 character wide printer, so you may want to edit it before using it.

If you want to filter (modify) the characters sent to your printer, you need to replace the xtab command (which is a filter that replaces tabs) with the name of the appropriate filter program. For example, if your printer can expand tabs, you don't need xtab, and can use cat instead.

Okay, let's suppose you have selected the name of your printer and the interface program to be "mp3ØØ" and "mp3ØØ", respectively. The printer is connected to the parallel device, /dev/cent. The lpadmin command that we used to set this printer up is

```
lpadmin -pmp3ØØ -v/dev/cent -mmp3ØØ
```

This creates a printer destination (-p) named mp3ØØ connected to port (-v) /dev/cent, using the model (-m) interface file /usr/spool/lp/model/-mp3Ø0.

Now, if you want this printer to be the default destination, that is, the printer that lp uses unless some other printer is specified, then you would use the command

```
lpadmin -dmp3ØØ
```

This makes printer mp3ØØ the default (-d) destination of lp requests. Now, imagine that you have a second printer, and that you want this printer to be included also as a default destination. You can make this second printer, that we will name daisy, and the original printer into a class of printers, and change the default to this class. For example, let's create the new printer, daisy, a serial printer connected to tty2:

```
lpadmin -pdaisy -v/dev/tty2 -mmp3ØØ -cany
```

The -cany argument creates the class "any" with a new printer "daisy". To add the original printer to this class, you type

```
lpadmin -pmp3ØØ -cany
```

Now, both printers are a member of the class any. To make the class the default, type

```
lpadmin -dany
```

This sets up lp so that requests go to the first available printer in the class "any". If all that you have is a serial printer connected to /dev/tty2, just use these commands:

```
lpadmin -pname -v/dev/tty2 -mmp3ØØ
lpadmin -dname
```

and you have set up lp for a printer named "name", connected to /dev/tty2 and using the interface program mp3ØØ. The printer "name" is also the default printer.

Great going! You've set up lp to work with your printer (we hope). But before you can test it, you must do three things: start the scheduler, enable the printer and allow it to accept requests. This is done with the following three commands:

```
lpsched
accept mp3ØØ
enable mp3ØØ
```

If you created a class, like "any", you must also use the accept command before it can accept requests, as in

```
accept any
```

and enable each printer in the class with enable commands. If this doesn't work, you'll see something like

```
accept: destination "mp3ØØ" has disappeared!
```

We still don't know why this happens, but one solution is to remove one directory, one file and clear one file. You do this with:

```
cd /usr/spool/lp
echo > default
rmdir request/mp3ØØ
rm member/mp3ØØ
```

This undoes the setup for a printer named mp3ØØ that is connected to /dev/cent. Then, following the instructions given previously, you

```
cd /usr/lib
./lpshut
./lpadmin -pmp3ØØ -v/dev/cent -mmp3ØØ
./lpadmin -dmp3ØØ
accept mp3ØØ
enable mp3ØØ
./lpsched
```

This sets things up again for printer mp3ØØ connected to /dev/cent.

The lpsched command starts the lp scheduler. The lp scheduler initiates the printing of each file. The accept command allows lp to request printing by a particular printer. The enable command enables the printer. The accept and enable commands are only necessary when you change or modify lp with lpadmin, or after you have used disable or reject.

Is everything working? You can try the lp command and see:

```
lp /etc/passwd
```

should send the passwd file to the default printer. This will take some thrashing about, as lp creates a new file with the request to print /etc/passwd (and other information) in it, and the scheduler actually starts printing the file. If nothing happens after a thirty seconds or so, you've made a mistake somewhere. Have you enabled the printer? Will it accept requests? Is lpsched running? lpsched will refuse to run if the file /usr/spool/lp/SCHEDLOCK exists, so you can remove this file and try again.

If you want or need to modify lp, you must first halt lpsched with lpshut. Any job that was being printed will be halted, and will be restarted at the beginning when lpsched starts running again.

Making lpsched a regular part of your system is done by adding the following lines to the file /etc/rc:

```
rm -f /usr/spool/lp/SCHEDLOCK
/usr/lib/lpsched
echo "LP scheduler started"
```

These lines will automatically start the lp scheduler every time your system goes multiuser. You don't need to enable or accept requests except after modifying a printer or class with lpadmin.

Using lp

Here are four ways to use lp:

```
lp /etc/passwd
lp < /etc/passwd
cat /etc/passwd | lp
lp -c /etc/passwd
```

Only the first of these four ways does not make a copy of the file /etc/passwd. This means that if the passwd file is modified between the time of the request and the actual printing of the file, the modified passwd file will be printed. Making a copy of the file to be printed will also take lp longer than using the original.

Every user can use the lp command. Several other commands are also available to all users:

```
lpstat    reports on the status of a printer or request
enable    enables a printer
disable   disables a printer
cancel    stops a request from printing
```

The lpstat command allows you to discover what has happened to either a particular printer or to a request for printing a file. For example,

```
lpstat -pmp3ØØ
```

returns the status of the printer named mp3ØØ, and

```
lpstat mp3ØØ-ØØ2
```

reports the status of the request named mp3ØØ-ØØ2. The request names are reported when you invoke the lp command.

The disable command allows anyone to halt a printer. For example, if you notice that a printer (say, daisy) is about to run out of paper, or has jammed, you can stop it by typing

```
disable -r"paper jam" daisy
```

This not only stops output to daisy, but also adds a reason (the -r argument) that can be printed by the lpstat command. After the printer is ready to be restarted, simply type

```
enable daisy
```

Output of the last file to be printed will be restarted.

The cancel command is used to remove any file from the list of requests. Suppose, for example, some request is sending endless formfeeds to the printer. You can quickly stop this by typing

```
disable daisy
cancel daisy-ØØ5
enable daisy
```

The disable command halts the printer, the cancel command removes the request from the queue and the enable command restarts the printer. If the request belongs to a different user than the one that canceled it, the owner of the request is sent mail by cancel.

Adding Disks to Your System

You can add more disks, both hard and floppy, to your root file system with the mount command. The mount command makes a file system on an additional disk part of the root file system.

There are only three requirements for mounting additional disks:

1. There must be a file system on the disk;
2. The file system must not already be mounted; and
3. A directory in a mounted file system must be available.

If you want to add a new hard or floppy disk, you must first format it. Floppy disks are formatted by typing

```
diskformat /dev/rdjØ        5 1/4"
diskformat /dev/rdj4        8"
```

To format an additional hard disk, the **fmw** program is used. This program is interactive, and will ask you to supply the drive number and the type of the hard disk. The **fmw** program will work with Syquest, CMI 16 megabytes and Quantum 34 megabyte drvies configured as drives 1-3. For other drives or formats, see the section named "Other Disk Formats". Although the format program checks the hard disks, we recommend that the the badblock program also be run. For the second hard disk, use

```
badblk /dev/mw1hwØ
```

After formatting the new disk, a file system must be made on it with the mkfs (make file system) command. Remember please, that this is only for new disks that you are adding. Both formatting and mkfs will destroy information already on a disk.

To make a file system on a 5 1/4" floppy disk, type

```
mkfs /dev/djØ 4ØØ 2 1Ø
```

To make a file system on a hard disk, type

```
mkfs /dev/mw1c `devsize -k /dev/mw1c` 8 9      (1Ø24 byte)
mkfs1B /dev/mw1c `devsize /dev/mw1c` 5 17      (512 byte)
```

These commands assume that you have configured the disk as the second drive (drive 1), with the device name /dev/mw1c. The `devsize -k /dev/mw1c` provides the number of (1Ø24 byte) blocks available for making the file system. The interleave factor for the free list is 8 and the number of sectors per track is 9. The mkfs1B makes 512 byte per block file systems. File systems on disks the are formatted with 1K sectors are both faster and more efficient.

After making a file system, you need to check it with

fsck /dev/rmw1c	hard disks
fsck /dev/djØ	floppy disks

You should add the name of any regularly mounted hard disk into the **/etc/checklist** file. The **fsck** program uses the names in this file when it is called without a device argument. List the additional hard disks first, and the root file system, /dev/mwØa, last.

Also, use the "raw" device for additional file systems, as in /dev/**r**mw1c. The raw device can be checked quicker than the "cooked" device. The root file system is an exception to this because the **fsck** program must know when it is checking the root, and won't know if you check the "raw" root device. Now, you are ready to mount the new file system.

Mounting File Systems

The mount command requires two arguments, the device name and a directory name. The /t directory in the root directory has been set aside for mounting temporary file systems. If you intend to mount a hard disk file system regularly, you may wish to create a directory just for it, as in

```
mkdir /a
```

This makes a new directory in root for semi-permanent hard disk file systems.

Mounting file systems is done with this command:

mount /dev/mw1c /a	for hard disks
mount /dev/djØ /t	for floppies

The decision to use directory /a or /t is an arbitrary one. You can use either, or make your own directory name with mkdir.

The mount command currently produces an errant warning message:

```
WARNING!! - mounting: <> as </t>
```

This is a bug. Mount will correctly complain if there isn't a directory by that name, or if the device is already mounted. Mount won't complain if you mount a disk that doesn't have a file system. Some error message will show up when you try to use a mounted non-file system disk, like "No such directory". The list of mounted devices is kept in the human readable file /etc/mnttab. You can display this list with the command without any arguments.

```
mount
/ on /dev/mwØa  read/write on Fri Aug 24 Ø2:31:59 1984
/t on /dev/djØ  read/write on Fri Aug 24 16:44:23 1984
```

You can routinely mount hard disks by adding a line with the appropriate mount command in the file /etc/rc. If you are routinely mounting a file system, you should also add its name (for example, /dev/rmw1a) on a line by itself in the file /etc/checklist (see above):

```
/dev/rmw1c
/dev/mwØa
```

/etc/checklist is used by fsck as the default list of file systems to check.

Unmounting File Systems

Disks are unmounted using the umount command. Please observe that this is not un-mount, but umount (no first n). The umount command requires the device name as its argument, and that the file system be idle before it is umounted.

```
umount /dev/djØ
Device busy.
```

A device is busy when there is a file open in the file system or someone's current directory is in that file system. The system administrator can use the fuser (find user) command to discover the identity of a user preventing a file system from being unmounted.

Other Disk Formats

There are other disk and diskette formats available with your TRICEP system. The default diskette format for the TRICEP is the MicroDecision format: double-sided, double density, 5 1k sectors for each track. The DJ/DMA driver also provides the following formats when used with the **diskformat** command appearing on the left:

8" formats --

```
diskformat /dev/rdj4 = 1K sectors with track Ø SD. The code  auto
                       selects  for  DS,  if  drive  has  2  heads.
 diskformat  -size  128  /dev/rdj4  =  standard  single  density  (IBM)
 diskformat  -size  512  /dev/rdj4  =  DUAL  Systems  type  diskette
```

5 1/4" formats --

```
diskformat -size 512 /dev/rdjØ = DOS (2.Ø) DS 9 sectors/track
diskformat -size 512 -head Ø /dev/rdjØ = DOS SS 9 spt
diskformat -size 512 -sec 1-8 /dev/rdjØ = DOS DS 8 spt
diskformat -size 512 -sec 1-8 -head Ø /dev/rdjØ = DOS SS 8 spt
```

Sector interleaving is not supported by the DJ/DMA driver. This may affect performance if you format diskettes on your TRICEP for use on other systems. The recommended route always is to format a diskette on the target system, and then to copy the files between the TRICEP and the diskette. The DJ/DMA driver reads sector headers to determine what format it is looking at, and adjusts itself accordingly.

For hard disks drives other than the Syquest, CMI 16 or Quantum 34, you need to formulate your own options for the diskformat command. This means that you need to discover the number of heads and cylinders for your drive. We suggest using 1K sectors (-size 1Ø24) for speed and maximum data storage. Through experimentation, Morrow has discovered that a sector interleave of 4 works best for drives with 9 1K sectors per track, but the only sure way of choosing the correct combination of interleaves (soft and hard) is empirically.

You can also specify the beginning cylinder for write-precompensation by using the **-dens** flag. The **-dens** has been redefined for formatting hard disks to be the track for beginning write-precompensation. The following line shows the diskformat command used with a CMI 1Ø to create 9 1K sectors per track, for 3Ø6 cylinders, 4 heads, write-precompensation beginning with track 128 and a hard interleave of 4:

```
diskformat /dev/rmw1hwØ -size 1Ø24 -cyl Ø-3Ø5 -head Ø-3 -dens 128 -il 4
```

The hard interleave is used for numbering the sectors on a track. For example, each sector has a sector header that contains an identifying sector number. By specifying an interleave of 4, the first sector is identified as Ø, the fifth (skip 4) is 1, the ninth is 2, and so on. Then, there is a second level of interleaving created by **mkfs**, known as soft interleave. **mkfs** uses the soft interleave when it builds the free list for a new file system. **fsck** will also use the same soft interleave when it salvages the free list.

Together, the hard and soft interleave theoretically assure that the heads will be correctly positioned to read or write the next sector when the software is ready. This is an ideal case because the free list quickly gets scrambled through using the file system.

To make a file system on the CMI 1Ø from the example, you use the command:

```
mkfs /dev/mw1c `devsize -k /dev/mw1c` 8 9
```

The "mw1c" partition is for a hard disk without a swap space. Only the root file system, /dev/mwØa, requires a swap space.

Connecting Modems

Your TRICEP can be set up for remote login by connecting a modem to one of the tty ports. This way a person possessing a modem and a terminal can access your TRICEP via phone lines. But, before you can use a modem with TRICEP, certain configuration changes must be made. The cable used between the TRICEP and the modem must also meet very specific conditions.

You need to make configuration changes to two files: /etc/inittab and /etc/gettydefs. Back in the section "Modified inittab Example" we explained how to change inittab to include a modem on port ttyØ. Please follow the example in that section for including a modem on port ttyØ. (We assume that you are using port ttyØ for your modem. This is not required; you can use a different port. Just remember to make the correct substitution for the port name in your modifications.)

The /etc/gettydefs file needs to have two entries added to it for security reasons. These entries differ from the others by including the line condition HUPCL. HUPCL means Hang UP on CLose, or disconnect the phone line whenever a user logs out. When the user logs off, the TRICEP hangs up the modem from its end. The lines that you need to add to /etc/gettydefs are:

```
mo_12ØØ# B12ØØ # B12ØØ SANE TABS3 HUPCL #
     \r\n\nTRICEP System V Remote\r\n\nlogin: #mo_3ØØ

mo_3ØØ# B3ØØ # B3ØØ SANE TABS3 HUPCL #
     \r\n\nTRICEP System V Remote\r\n\nlogin: #mo_12ØØ
```

This sets up the tty line specified in inittab for proper operation with a modem. You may specify either 12ØØ or 3ØØ baud as the initial baud rate. Getty will switch to the alternate baud rate if it detects a framing error caused by non-matching baud rates or by pressing the BREAK key.

You also need to change the minor device number of the tty line that you wish to use. Bit 7 must be set to indicate to the device driver that pin 5 (CLEAR TO SEND) will be used to enable the modem. As superuser, type

```
rm /dev/ttyØ
mknod /dev/ttyØ c Ø 129
```

If you are using some other tty line than ttyØ, simply add the number of the line to the minor device number used in this example (129). For example, to change the minor device number for a modem connected to /dev/tty7, use

```
mknod /dev/tty7 c Ø 136
```

Once you have completed these configuration changes, you need to construct a cable. Also, if you want to be able to call up other computers via your modem, there are other files that must be changed. See the section on "Using cu" and "Configuring for uucp" that follow.

Modem-to-TRICEP Cable

The modem-to-TRICEP cable differs radically from a terminal to TRICEP cable. This is because more signals are necessary to properly carry out modem communication. Two additional signals are used at the TRICEP end: CLEAR TO SEND and REQUEST TO SEND. In addition, the chassis ground is included. Here is a diagram showing the way the cables is built:

TRICEP	MODEM
1 (CHASSIS GROUND)	1 (CHASSIS GROUND)
2 (RECEIVE DATA)	3 (TRANSMIT DATA)
3 (TRANSMIT DATA)	2 (RECEIVE DATA)
4 (REQUEST TO SEND)	8 (CARRIER DETECT)
5 (CLEAR TO SEND)	2Ø (DATA TERMINAL READY)
7 (SIGNAL GROUND)	7 (SIGNAL GROUND)
2Ø (DATA TERMINAL READY)	6 (DATA SET READY)

Microcomputers are normally set up to be DCE's, that is, Data Communication Equipment, so they can be used with terminals (DTE's). Of course, modems are set up the same way (as microcomputers. Mini's and mainframes are set up as DTE's). So, when modems are connected to microcomputers (or computers to computers), the transmit and receive lines must be crossed (pin2 to pin3, and pin 3 to pin 2). This is called a null-modem connection.

The modem's pin 6, DATA SET READY line, drives the TRICEP's pin 2Ø, DATA TERMINAL READY. Remember, pin 2Ø must be logic TRUE (high, +5 to +2Ø volts) before the SIO4 will allow communication, the same as for printers.

After you have activated a getty for the modem line (via inittab and telinit), getty will open the modem line, /dev/ttyØ in this example. This brings the TRICEP's pin 5 (CLEAR TO SEND) high, enabling the modem to receive calls by activating pin 2Ø (DATA TERMINAL READY) on the modem.

The modem brings its CARRIER DETECT line (pin 8) high when it answers a call from another modem. This is connected to the TRICEP's pin 4, (REQUEST TO SEND), waking up getty so it can collect a login name. From here on, the process is identical to an ordinary user login.

TRICEP drops its CLEAR TO SEND line, pin 5, when the user logs out or exits. This forces the modem to drop its carrier (disconnecting the remote modem) and hang up. This is done by the HUPCL flag in the gettydefs line for the modem.

If the carrier is lost inadvertently, TRICEP will kill all process associated with the modem line and hang up. Once the all the processes are killed, TRICEP prepares for the next caller.

NOTE: Additional voltages are provided at limited amperages on pins 8, 9 and 1Ø of each RS-232 connector on the TRICEP. These are intended for use with short haul modems and the like.

pin 8	+12V	(Limited to 4 ma)
pin 9	+12V	(Limited to 2Ø ma)
pin 1Ø	-12V	(Limited to 4Ø ma)

Using cu

The **cu** command, used to **c**all **u**p remote modems, needs to have two files changed before you can use it. The first change is that you must create a "special" device name for **cu** to use. This special device name has the same major and minor device numbers as the original, non-modem, device. For a modem connected to /dev/ttyØ, use the command

```
mknod /dev/cuØ c Ø 1
```

to create this device file. If you try to call out using the modem configured file, /dev/ttyØ (see above), you won't succeed. The modem bit prevents the device from opening until a carrier is detected. The device file /dev/cuØ doesn't have this bit (128) set, so you can call out.

The second thing about **cu** is that its baud rate is configured by the file **/usr/lib/uucp/L-devices.** This means that trying to set the baud rate with the command

```
cu -s 12ØØ -l/dev/cuØ
```

doesn't work because it is overridden by the L-devices file. This is intended to insulate users from needing to know the correct baud rate. Add the line

```
DIR cuØ cuØ 12ØØ
```

to the **/usr/lib/uucp/L-devices** file to establish the baud rate for this line at 12ØØ baud. Of course, substitute 3ØØ for 12ØØ if you need a lower baud rate.

The DIR in this line stands for DIRect connection to a modem. The first "cuØ" stands for which **/dev** file to use, and the second stands for which "automatic dialer" to use. Modems, like the D.C. Hayes, accept ascii characters as commands to dial out, unlike older modems that had separate autodialers. To dial Morrow using a Hayes modem connected to port ttyØ (through /dev/cuØ, of course), you would type:

```
cu -l/dev/cuØ
Connected  (from the cu program)
(Enter "ATDT4156321 4ØØ8<RETURN>" but you  won't see an echo)
CONNECT    (from Hayes Modem when it gets answer from remote)
morrow login: ...

(text of session)

% exit
~.

Disconnected  (also from cu)
```

If you are using a handset (telephone) to manually dial, type the "cu -l/dev/cuØ" command, wait for the "Connected" response, dial the number and switch your modem to "Originate" when your hear the remote computer "answer" with a high pitched tone. If the baud rate appears incorrect (no "login:" message, or garbage), keep pressing the RETURN key, or hit the BREAK key for a second and wait.

Configuring for uucp

The uucp programs, Unix to Unix copy, are used for inter-system mail and file transfer. There are three files in the **/usr/lib/uucp** directory that you need to edit before you can use **uucp**. The first file is the L-devices file, which was explained in the previous section. The other two files are the L.sys and USERFILE. Also, if you want to be an active node, you need to have an autodial modem, as mentioned above, or a direct line (RS-232 null-modem or short haul modem) connecting the two systems.

A passive node is a system that can be logged into by other systems using uucp. There is already an /etc/passwd entry for uucp that uses the uushell. But, even if you are a passive system, you may still want to edit the USERFILE. You may also want to add a password to this entry, or create other entries with passwords (see section named "/etc/passwd: The User Name File").

The USERFILE determines what directories may be accessed by users through the **uucp** programs. This is a security precaution. The existence of **uucp** logins is a terrible breech in a system's security. An intruder can use **uucp** to copy any file that a user has failed to protect, unless you have edited USERFILE. The USERFILE supplied with your TRICEP allows unlimited access to the file system. It contains the line

```
,  /
```

that means that any user on any machine may access any file. This is limited only by the standard file permission modes. The **uucp** programs have the same file access permissions as set for "others", that is, non-owner and non-group.

==========

Although this is out-of-place in this section, here is a short explanation of file permissions:

FILES

read	file may be copied or viewed
write	file may be modified or cleared (made empty)
execute	file may be executed (read permission not necessary except for shell scripts)

DIRECTORIES

read	names in directory may be listed
write	names in directory may be changed or removed
execute	files or subdirectories may be accessed (searched)

To completely explain file permissions takes about 1Ø pages of examples. But, hopefully, this succinct summary will guide you.

Of special interest is the way file permissions work with directories. If a directory is read-only, the only thing that can be done with it is to list the names in it. But, a directory that is execute-only allows persons who know the name of a file to access a file, but hides the names of files in this directory from others. Also, removing execute permission (also known as "search permission") from a directory protects all the subdirectories branching off from it. For example,

chmod 7ØØ Private

makes the directory named Private, and any file or subdirectory of Private, inaccessible to anyone except the owner of Private (and the superuser).

==========

Each line in the USERFILE file has the format

name,system [c] pathname [pathname...]

where name is the login name, system is the system name (as set by **/etc/sysname**). The "c" is the optional call-back required flag. When this flag is present, **uushell** will terminate a conversation with the remote system and call it back, using the entry for the system in the L.sys file. This assures you that the system that has called is authentic, not someone using stolen login names and passwords.

The pathnames are the names of directories that **uucp** can access. For example, if your USERFILE contains the line

```
uucpt,thoth    /usr/spool
```

a remote **uucp** program, logging in as "uucpt", from a system named "thoth", can access files that are in the /usr/spool directory, or subdirectories of /usr/spool.

```
uutoms,tommy  c  /
```

allows the user named "uutoms", calling from system "tommy", to access any file (depending on the file's permission), but only after the "tommy" system has been called back.

```
uucp,     /usr/spool/uucp/public
```

permits any user named "uucp" from any system (notice that there's no system name) to access files in the /usr/spool/uucp/public directory.

A good strategy for creating your USERFILE is to make entries for the user names and systems that you know and trust, and to have a restrictive catchall entry last. A "catchall" entry has the name and system fields empty (a comma by itself). For example,

```
,    /usr/spool/uucp/public
```

limits any user whose name did not appear previously in the USERFILE to files in the /usr/spool/uucp/public directory. In the following example, only the users uucpt and uutoms can access files outside of the /usr/spool/uucp/public directory.

```
uucpt,thoth    /usr/spool
uutoms,tommy  c  /
,              /usr/spool/uucp/public
```

The L.sys file (in /usr/lib/uucp) must be edited if you intend to be an active system, that is, a system that can call other systems. This assumes that you have an intelligent modem that can interpret ascii strings as calling commands. Each entry in the L.sys file has the name of a system that may be called and some other information, in the form

```
system times device speed device login_stream
```

"system" is the system name built-into the remote system. "times" specifies when the system may be called. Please refer to the Administrator's Guide, page 11-5, for a good explanation of what may go in "times". "device", which appears twice, is the name of the line used for calling the system, for example, cuØ. "speed" is the baud rate, like 12ØØ. Finally, comes the login_stream. This seems the most difficult part of an L.sys file entry to understand. Let's look at an example entry:

```
tommy Any cuØ 12ØØ cuØ "" ATDT5551212 CONNECT "" "" @
  ogin--ogin-EOT-ogin-BREAK-ogin log-name ssword our-password
```

"tommy" is the system name, which can be called "Any" time, using device "cuØ" at 12ØØ baud. The rest of this entry (that is ONE line long, no newlines) contains the "login_stream".

The login_stream is a series of expect-send strings. In other words, your system "expects" a string, and "sends" the following string after receiving the expected string. Just to relate this to your everyday experience, when you sit down at a terminal connected to a Unix system, you expect to see the string "login". Your response is to send your login name. Next, you expect to see the string "Password", and you send your password.

And, indeed, the last part of the example resembles this. The string "ogin" is expected, then the "log-name" is sent, then the string "ssword" is expected, and the password is sent.

Now, for the rest of the entry. Two double quotes together, "", represent a null string (that is, nothing). This is the first "expect" string, expect nothing. Then, you want to send your intelligent modem, in this case a Hayes, the string that causes it to dial a number, ATDT5551212. The Hayes will respond with CONNECT when it has succeeded in contacting the remote system, so the next thing to expect is "CONNECT". Then, you send an "@" sign, the default line-erase (kill) character. This should clear out any garbage in the remote system's buffer.

The next "expect" string is a fancy one. It contains secondary expect-send strings, and means that it will attempt to receive the expected string, "ogin" (short for login), and will send other strings when it receives an **un**expected string. Here's our fancy expect string:

```
ogin--ogin-EOT-ogin-BREAK-ogin
```

Interpreted, it means: expect "ogin", if you get anything else, send a RETURN (--), expect "ogin", if you get anything else send EOT (End of Transmission), expect "ogin", if you get anything else, send BREAK, expect "ogin". This sequence is a venerable hand-me-down that seems to work. The idea is that sending the @, RETURN, EOT or BREAK may convince the remote system to send you a string containing "ogin", for example, "morrow login:".

To create your own L.sys file, make one line entries for each system that you want to be able to call. The system name must be the same as the remote system's name, and will be used by **mail** and **uucp**, as in **mail morrow!rik**". You should use the hand-me-down expect-send string, adding in the correct phone number, your own login-name and password (if any).

MORROW ENHANCEMENTS

Your TRICEP system includes software that is not provided by other System V Unix ports. This software is part of the bridge between your TRICEP and the world of single-user microcomputers. This bridge includes file transfer programs for CP/M and MS/DOS, and the ability to run MS/DOS programs on TRICEP SP-80188 slaves.

Other programs have been added to your TRICEP system. The ddt command (disk debugging tool) allows the editing of non-ascii files. The cptree command copies subdirectories from one part of a file system to another. Documentation on these and other programs (as they are added) is found in the back of this manual (also in /usr/man/local/man1 directory).

Diskette Specialists: far and dar

The two most popular operating systems for microcomputers are CP/M and MS-DOS. Each of these operating systems uses its own specific disk organization, making disks containing files from one operating system unusable by the other. These disks are also unusable to almost everything else but a particular brand of microcomputer and its operating system.

There is a way of making such disks usable, by combining hardware that can read different formats with software that understands different operating systems. The TRICEP programs, far and dar, understand CP/M and MS-DOS disk organization. And, the TRICEP floppy disk controller (and device drivers) can read and write disks formatted by:

- Micro Decisions using CP/M 2.2 or 3.0
- IBM PC's and other compatibles using MS-DOS 1.0, 2.0, 2.1
- Other systems using CP/M 2.x and 8" diskettes

The driver software attempts to decide whether the disk is double or single sided, so that this is not a concern of the user.

Both far and dar use a simple command line interface to replace, extract, show a directory and delete files. The arguments for both programs are similar, making them easier to learn and use. The similar arguments will be explained first, followed by the differences in each program.

MORROW ENHANCEMENTS

Your TRICEP system includes software that is not provided by other System V Unix ports. This software is part of the bridge between your TRICEP and the world of single-user microcomputers. This bridge includes file transfer programs for CP/M and MS/DOS, and the ability to run MS/DOS programs on TRICEP SP-188 slaves.

Other programs have been added to your TRICEP system. The ddt command (disk debugging tool) allows the editing of non-ascii files. The cptree command copies subdirectories from one part of a file system to another. Documentation on these and other programs (as they are added) is found in the back of this manual (also in /usr/man/local/man1 directory).

Diskette Specialists: far and dar

The two most popular operating systems for microcomputers are CP/M and MS-DOS. Each of these operating systems uses its own specific disk organization, making disks containing files from one operating system unusable by the other. These disks are also unusable to almost everything else but a particular brand of microcomputer and its operating system.

There is a way of making such disks usable, by combining hardware that can read different formats with software that understands different operating systems. The TRICEP programs, far and dar, understand CP/M and MS-DOS disk organization. And, the TRICEP floppy disk controller (and device drivers) can read and write disks formatted by:

- Micro Decisions using CP/M 2.2 or 3.0
- IBM PC's and other compatibles using MS-DOS 1.0, 2.0, 2.1
- Other systems using CP/M 2.x and 8" diskettes

The driver software attempts to decide whether the disk is double or single sided, so that this is not a concern of the user.

Both far and dar use a simple command line interface to replace, extract, show a directory and delete files. The arguments for both programs are similar, making them easier to learn and use. The similar arguments will be explained first, followed by the differences in each program.

far and dar: Command Line Syntax

As mentioned, both far and dar share most of their syntax. The format for these commands is

```
far device_name {arguments} {files}
dar device_name {arguments} {files}
```

The device_name is the name of a special file in the device directory, usually /dev/dj0. /dev/dj0 is the name of the first 5 1/4" floppy disk drive. /dev/dj4 is the name of the first 8" drive connected to the system.

You do not have to include the /dev directory name when using far and dar. These commands prepend the directory name /dev to device_names if necessary. For example,

```
far /dev/dj0
far dj0
```

are equivalent.

The default argument when no other arguments are present is **t**, for show directory (**t**able of contents). Thus, the previous far examples would display the directory of a CP/M diskette in the first 5 1/4" drive (/dev/dj0).

There are four other common arguments to far and dar: **c**reate, **d**elete, e**x**tract and **r**eplace. Only one of r, d, or x arguments (or t) may be used at a time. That is, you cannot extract and replace a file with the same command. The actions of these four arguments follows:

c create an empty directory on the diskette; this deletes all the files listed in the diskette directory; the c option works together with the d and r options to create a new CP/M or MS-DOS directory where (possibly) no directory existed before. This should be used with caution because it ignores the current diskette format. The c option used without d or r has no effect.

d delete the named files.

x extract the named files <u>from</u> the diskette.

r replace the named files <u>to</u> the diskette.

The verbose argument can be used in combination with any of the other five arguments. Verbose means just what it implies: be verbose (talky) while performing the action. The verbose option also provides a summary of the space remaining on the diskette.

The {files} argument means all the files on the diskette (or in the directory) when no filenames are present. In other words,

```
far dj0 x
```

means copy all the files from a CP/M diskette to the current directory, and

```
dar dj0 r
```

means copy all the files from the current directory to an MS-DOS diskette. Otherwise, far and dar operate only on the filenames given, as in

```
dar dj0 d doc.bak
```

that removes the file doc.bak on an MS-DOS diskette.

Expansion metacharacters, * and ?, must be used with caution, because the shell will attempt to expand them before passing them to the far or dar commands. If you want far to expand a filename, for example, *.com, you must protect it from the shell by quoting it. dar does not perform metacharacter expansion, but far will.

```
far dj0 x "*.com"
```

copies all the .com files from a CP/M diskette, and

```
far dj0 x *.com
```

copies all the .com files that already exist in the current directory AND on the floppy to the current directory.

far Differences

Since far deals with CP/M diskettes, it works somewhat differently than dar. The major difference is that CP/M has 16 user areas for subdividing diskettes.

The user option sets up the far command so that it works in a particular user area. For example,

```
far dj0 xu3
```

extracts all the files in user area 3 of a CP/M diskette. The default user area is 0.

CP/M filename conventions specify that filenames can be up to 8 characters long, with a three character extension. Small characters are converted to capital letters, so the filenames

```
CHAP31.BAK
chap31.BAK
```

are identical to CP/M. Besides capitol letters and digits, CP/M allows the following punctuation characters:

! @ # $ % ^ & ( ) _ - + ~ ` ' " | { } \ /

far will automatically convert lower case letters to upper case, truncate filenames longer than 8 characters and extensions longer than three, and remove directory names. For example, /user/rik/new.text would become NEW.TEX in a CP/M directory.

dar Differences

MS-DOS uses a different strategy for organizing files on a disk. MS-DOS uses a system of subdirectories very much like the UNIX file system. The dar command does not provide a special argument for creating subdirectories, but will create them as needed.

As an example, if we copy a file to a DOS diskette with the command

dar dj0 r /users/rik/new.text

dar creates two directories on the diskette: \USERS and \USERS\RIK. Notice that MS-DOS uses a backslash (\) to separate directories. dar handles the conversion between a slash and a backslash, so you will always use a slash.

Then, the file NEW.TEX is created in the \USERS\RIK directory. MS-DOS has the same limits on filenames as CP/M (8 characters for the name, three for the extension), and allows the use of the following characters in filenames:

A-Z	0-9	$	&	#	%	'	(	)	-
@	^	{	}	~	`	!			

udos: UNIX DOS

The **udos** program provides the capability to run MS/DOS software on TRICEPs equipped with SP-188 boards. The MS/DOS software must use the DOS system call interface and not make calls to proprietary ROM's. This means that you can run SuperCalc II for the 8086 or the MS/DOS macro-assembler on your TRICEP, but you can't enjoy the flight-simulator programs.

udos includes a terminal emulation capability that uses the Berkeley curses library. This allows you to use one installed version of a program, like SuperCalc, to run on a variety of terminals. Terminal emulation works well with programs that with low to moderate character output. Wordprocessing programs have high character output, spreadsheet programs have low character output.

The 80188 processor on the SP-188 board runs at one and a half times the speed of the processor in an IBM-PC. The higher clock speed and character output handling can result in MS/DOS programs running at 1.75 times the speed of the same program running on an IBM-PC XT. This makes the TRICEP system an ideal software development environment for MS/DOS programmers.

The **dar** program is used to copy DOS formatted diskettes to directories in the Unix file system. The documentation for **udos** appears in the back of this manual, and is intended to be a complete explanation for the use of **udos**.

ddt: Disk Debugging Tool

Although this program takes its name from an old CP/M program, its use is somewhat different. ddt allows the viewing and editing of non-text files. This is useful when you need to change several bytes in an already existing file that cannot be handled by a text editor. **adb** is the UNIX version of the "dynamic debugging tool" and is more suitable for debugging programs, whereas ddt is better for editing files.

TRICEP SYSTEM-SPECIFIC VALUES

Anyone who gets a System V UNIX port gets a standardized kernel with one exception. The exception is the device drivers, the software that controls and communicates with devices: disks, terminals, memory and printers. The implementation of device drivers has much to do with the performance of a UNIX port. Poorly written device drivers slow down UNIX, because the device drivers are among the most often used parts of the kernel.

The device drivers for the TRICEP were rewritten by Morrow to increase speed and add flexibility. Generally speaking, device drivers are based on a generic device driver that is edited to fit the hardware. The TRICEP device drivers were written specifically to take advantage of the built-in intelligence of the TRICEP devices.

The hard disk controller (HDC-DMA), the floppy disk controller (DJ-DMA) and the terminal controller (SIO4-DMA) all have separate processors that unburden the main CPU. Each peripheral CPU handles device I/O in parallel with the main CPU, uses DMA to transfer information to/from main memory, and signals completion to the main CPU by interrupts.

The connection between the rest of the kernel and the device drivers is made through the device switches. There are two device switches, one for block devices (disks) and one for character devices (terminals, printers, etc.). The device switch structure definitions are contained in the file /usr/src/sys/conf.c, for those of you who have purchased reconfiguration licenses.

Major and Minor Device Numbers

The UNIX system accesses the devices through the switch table by means of special files in the /dev directory. These files are block or character special files. Block or character special files are indicated in long directory listings by having a b (block) or c (character) at the beginning of the permissions:

```
ls -l /dev/mw0a /dev/console
crw------- 2 root     0,   0 Oct 3 13:45  /dev/console
brw------- 1 root     0,   0 Oct 3 13:47  /dev/mw0a
```

The two numbers preceding the date are the major and minor device numbers. Special files do not have any data blocks. Instead, they are used to hold (in the inode) the major and minor device numbers. The major device number selects one particular device driver from the block or the character device switch. The minor device number is passed to the device driver and contains information used by the driver to select, for example, which of several drives or terminals to access, or which disk partition to use.

Special files are created with the **mknod** command. Your system comes with most of the special files that you will need in the /dev directory. The next several sections define the major and minor device numbers for the different devices.

If you ever lose a special file, or need to add one, the mknod command is used to create special device files. The syntax for this command is

```
mknod device_name b-c major_device_number minor_device_number
```

where b stands for block special and c for character special. If you needed to create a new block device, for example, the fourth hard disk used as a file system without a swap space, you would use the command

```
mknod /dev/mw3c b 0 26
```

The definition and meanings of the major and minor device numbers are explained in the next several sections. The table at the end of this chapter provides most of the information you are likely to need for creating special files.

TTY Devices (SIO4-DMA)

The tty devices are character devices and have major device number 0. The minor device number has two purposes: to select one of eight possible ports and to enable/disable pin 5 (Request to Send). The least significant three bits select the port. Bit seven enables Request to Send when it is a 1, and disables it when it is zero. (See Adding Modems). The command

```
mknod /dev/console c 0 0
```

was used to create the console device, and

```
mknod /dev/modem c 0 129
```

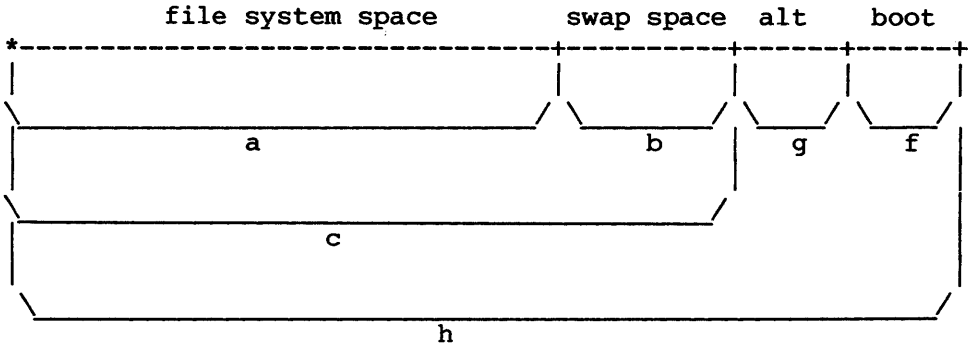
creates a modem device using port tty0 and Request to Send enabled (129 = 1000 0001). The console is the first port (number 0), tty0 the second port (number 1), and tty6 the seventh port (number 7).

Mini Winchester Devices (HDC-DMA)

The hard disks (mini winchesters) have a block major device number of zero. This does not conflict with the tty device, since the tty major device number refers to the character device switch. The minor device number for hard disks has two purposes: to select one of four possible hard disks and select a partition on the disk.

Hard disks are traditionally partitioned on cylinder boundaries. A partition is a logical (non-physical) division of a disk. A cylinder is the collection of tracks on a disk that may be accessed by one of the two to eight read/write heads without stepping.

There are six partitions built-into the hard disk device driver. They are described in the following diagram:



The designation of partition "a" as a file system is purely arbitrary: there is nothing special in the device driver requiring this partition to be a file system. Likewise, the "swap", "alt" (alternate blocks) and "boot" (boot loader) are just names. This is the way your root hard disk is currently set up. The following table is based on the illustration of partitions, and includes the number that is built into the minor device number for each partition.

a	0	File system with swap space reserved
b	1	Swap space (at least two megabytes)
c	2	File system without a reserved swap space
d	3	reserved for future use
e	4	reserved for future use
f	5	Boot loader space (at least 50 blocks)
g	6	Alternate blocks space (at least 50 blocks)
h	7	Whole drive - used for formatting, copying

The location of the partitions is written in the boot block, block 0, when a disk is formatted. This is not typical: most UNIX systems compile the partitions into the kernel. Having the partitions written in the boot block allows the device driver to determine the partitions dependent upon the size of the hard disk. Otherwise, a special kernel (or minor device number) would be needed for each different hard disk size.

The structure that defines the boot block is found in the file /usr/include/sys/diskparam.h.

The minor device number for the hard disk drivers is built as follows:

BIT	7	6	5	4	3	2	1	0
	Write-Block 0	X	X	Drive number		Disk partition		

Most mw devices do not allow writing to block 0 (the boot and configuration block). Setting bit 7 to 1, as in the device /dev/mw0hw0, allows writing in block 0. This protects block 0 from accidental overwriting. The disk formatting programs (diskformat and fmw) have private access to block 0.

Bits 3 and 4 determine which of four possible drives to access. Bits 0 through 2 select one of the 5 possible partitions to use. Bits 5 and 6 are not used.

There is also a raw hard disk device, major character device number 4. The raw, as opposed to cooked, device transfers data directly between the disk and user memory, bypassing the block buffers, and locking the calling process into memory until the process finishes. This is useful for system programs that perform block reads and writes (such as dcopy and dd). The same minor device numbers apply to the raw device. Use of the raw device locks a process into memory during disk transfers, and should only be used for copying disks, or running fsck on other than the root file system.

Floppy Disks (DJ-DMA)

The block major device number for floppies is 1. The minor device number has two purposes: to select one of eight possible drives and select one of three partitions. The least significant three bits select one of eight possible drives: 0 to 3, the 5 1/4" drives, and 4 to 7, the 8" drives.

The dj device has two special partitions. One is indicated by setting bit 7 to 1, and is used specifically with the tar program. The tar program was designed for tape archiving and doesn't know that it shouldn't overwrite the boot block of 5 1/4" floppies. The boot block is special because it contains information used to determine single or double sidedness for 5 1/4 inch CP/M and MS/DOS format floppies. The special tar dj device, indicated by the letter "a", (e.g., /dev/dj0a), is ONLY for use with tar and 5 1/4" floppies. The regular partition, including the entire diskette, is used for all other applications.

The other dj device partition is selected by setting bit 6 to a 1. The specifies the part of a 5 1/4" stand alone floppy disk that contains the boot loader. This partition is only used on stand alone diskettes, used for repairing or rebuilding the root file system. It is named /dev/dj0f or /dev/djboot for the first drive.

There is also a character special floppy device with major device number 5. The raw floppy device is used mainly for formatting (diskformat). The minor device numbers work the same as for the cooked (block) device.

mw3g	b	0	30	fourth hard disk, alternate blocks
mw3h	b	0	31	whole fourth hard disk
mw3hw0	b	0	159	like mw3h & enable write boot block
null	c	2	2	null device, throw things away here
rdj0	c	5	0	raw first 5 1/4" floppy (format)
rdj4	c	5	4	raw first 8" floppy (for format)
rmw0a	c	4	0	first h.d. file system (copy f.s.)
rmw0h	c	4	7	whole first hd (copy disk)
rmw0hw0	c	4	135	whole first hd & enable boot write
rmw1a	c	4	8	second hd file system (copy f.s.)
rmw1h	c	4	15	second whole hd (copy disk)
rmw1hw0	c	4	143	whole second hd & enable boot wrt
rmw2a	c	4	16	third hd file system (copy f.s.)
rmw2h	c	4	23	whole third hd (copy disk)
rmw2hw0	c	4	151	whole third hd & enable boot wrt
rmw3a	c	4	24	fourth hd file system (copy f.s.)
rmw3h	c	4	31	whole fourth hd (disk copy)
rmw3hw0	c	4	159	whole fourth hd & enable boot wrt
swap	b	0	1	swap device (only use one)
syscon	c	0	0	linked to console
cu0	c	0	1	second serial port for call out
tty0	c	0	129	second serial port as modem
tty1	c	0	2	third serial port
tty2	c	0	3	fourth serial port
tty3	c	0	4	fifth serial port
tty4	c	0	5	sixth serial port
tty6	c	0	6	seventh serial port
tty7	c	0	7	eighth serial port

Some of these devices do not currently exist in your device directory. And, you don't necessarily need them. If you add hard or floppy disks, for example, you will probably need to add a new device name. Simply find the name (or description) that matches your new device, change directory to /dev (cd /dev), and use the first four columns of this table for arguments to the mknod command. For example, to add a third hard disk device (that doesn't need a swap space), you would type

```
cd /dev
mknod mw2c b 0 18
```

to create the new device name.

All file system devices (device names beginning with "mw" or "rmw") must be owned by check for security reasons. The file access permissions for these devices must be read and write by owner only. To make these changes to our example, use these commands:

```
chown check mw2c
chmod 600 mw2c
```

These commands make the file system on mw2c accessible through the operating system only. Otherwise, a user could accidentally (or maliciously) access the file system directly.

Master Device Table

There are several other device drivers used in the TRICEP. These are for accessing memory, kernel memory, and the parallel port. There is also the null device, a bit bucket used for throwing away unwanted data. The table that follows defines these devices, along with the devices already mentioned.

Device	Type	Major	Minor	Description
cent	c	6	0	centronics parallel port
console	c	0	0	system console
dj0	b	1	0	first 5 1/4" floppy
dj0a	b	1	128	first 5 1/4" floppy for tar only (linked to mt1)
dj0f	b	1	64	boot partition of stand alone
dj0boot	b	1	64	linked to dj0f
dj1	b	1	1	second 5 1/4" floppy
dj1a	b	1	129	second 5 1/4" floppy for tar only
dj2	b	1	2	third 5 1/4" floppy
dj2a	b	1	130	third 5 1/4" floppy for tar only
dj3	b	1	3	fourth 5 1/4" floppy
dj3a	b	1	131	fourth 5 1/4" floppy for tar only
dj4	b	1	4	first 8" floppy
dj5	b	1	5	second 8" floppy
dj6	b	1	6	third 8" floppy
dj7	b	1	7	fourth 8" floppy
error	c	3	0	used by kernel
kmem	c	2	1	kernel memory
mem	c	2	0	memory
mt1	b	1	128	tar virtual tape device - linked to dj0a
mw0a	b	0	0	root file system
mw0f	b	0	5	boot cylinder of root file system
mw0boot	b	0	5	linked to mw0f
mw0h	b	0	7	whole first hard disk
mw0hw0	b	0	135	like mw0h & enable write boot block
mw1a	b	0	8	second hard disk file system
mw1b	b	0	9	second hard disk swap space
mw1c	b	0	10	second hard disk, f.s. w/o swap
mw1f	b	0	13	second hard disk, boot cylinder
mw1g	b	0	14	second hard disk, alternate blocks
mw1h	b	0	15	whole second hard disk
mw1hw0	b	0	143	like mw1h & enable write boot block
mw2a	b	0	16	third hard disk file system
mw2b	b	0	17	third hard disk swap space
mw2c	b	0	18	third hard disk, f.s. w/o swap
mw2f	b	0	21	third hard disk, boot cylinder
mw2g	b	0	22	third hard disk, alternate blocks
mw2h	b	0	23	whole third hard disk
mw2hw0	b	0	151	like mw2h & enable write boot block
mw3a	b	0	24	fourth hard disk file system
mw3b	b	0	25	fourth hard disk swap space
mw3c	b	0	26	fourth hard disk, f.s. w/o swap
mw3f	b	0	29	fourth hard disk, boot cylinder

INTERPRETING CONSOLE ERROR MESSAGES

In a perfectly administered, perfectly functioning system, you will never see an error message. Of course, perfection is not possible in this world. Instead, file systems will run out of space, disk systems will develop some errors and system usage will occasionally overload your system.

This section explains the error messages that are generated by the kernel. The Unix commands and programs can produce their own error messages; these are not covered in this section. They are supposed to be self-explanatory (and sometimes are). The kernel, with its responsibility for handling the hardware and the file systems, has built-in messages that announce what has gone wrong. Unfortunately, these messages are not consistent and not self-explanatory.

And, even worse, most of the kernel error messages were picked out of the kernel file with the command

strings /unix.

Unisoft does not provide documentation for the error messages produced by their kernel. This chapter will not be "perfect" until UniSoft provides us with support by comparing our interpretations with their kernel source code.

Error messages from the device drivers added by Morrow are fully documented because source code is available. These include the hard and floppy disk devices. A breakdown of the error messages produced by the kernel follows:

From the UniSoft-provided kernel:

- Kernel table overflows
- File system problems
- Swap space problems
- Trap and interrupt errors
- Memory management errors

From the Morrow device drivers:

- Floppy disk errors (dj)
- Hard disk errors (mw)

Please bear in mind that the next five sections discuss "interpretations" of error messages. Hopefully, UniSoft can be persuaded to amend this situation before this section of the manual is typeset.

"Panic..."

If you see a message on the console beginning with the word "panic", your system has crashed. Gracefully. The panic message means that the kernel has detected an error that it considers so deadly that it has halted the system before more damage can be done. The panic routine may attempt to write out block buffers before halting the system (performing a sync). If this fails, or a second drastic error occurs before the system is halted, you may get a Double panic!

Panic messages are, as you may imagine, not a good thing. Memory management errors or a missing root file system can cause a panic message. You may be able to reset your system and continue, after, of course, running **fsck** on your file system(s). The panic message may be caused by something random (like a full moon or a power surge) and never happen again. Or, you may have serious hardware problems. Our experience at Morrow has been that you can often reset, **fsck** and keep going. After all, the kernel did detect the problem and halt the system. If the panics keep happening, you most likely do have serious hardware problems. Write down the error message and contact your service representative.

Kernel Table Overflows

The Unix kernel is the operating system. The kernel is alway present in memory, and provides the interface with user commands and programs and the file system and devices. Part of this interface consists of tables where information about currently executing programs (processes), open files and devices are kept. The size of these tables is fixed when the kernel is generated.

You can monitor how well the tables in your system meet your needs by using the System Activity Package, with the programs sadc and sar. The tables were configured fairly large by Morrow, but your environment may overflow these tables. The solution in most cases is to either distribute your systems usage (to reduce usage), or re-generate the kernel with larger tables.

MESSAGE	INTERPRETATION
no file	Open File Table has overflowed; no more files may be opened or programs started
Inode Table Overflow	Files are defined by their inodes; no more files may be opened or programs started

Timeout table overflow	The timeout table is used by device drivers and programs that want to set an alarm; may affect device drivers
out of text	The text table has overflowed; no more programs can be loaded as pure text
no procs	The process table is full; no new process can be started
proc on q	Process already exists; a diagnostic rather than a table overflow

File System Problems

The Unix file system is a rigorously defined data structure. Different parts of the file system should correspond or something has gone awry. The **fsck** program uses these correspodences to repair file systems. But, the kernel must be able to detect some problems with the file system that occur while the file system is in use. These problems may be hardware related errors, or caused by running out of a resource, for example, free blocks.

In the case of hardware related errors, Hard or Floppy Disk error messages will be intermingled with the File System error messages. A flood or dramatic increase in these error messages presages hardware failure. The solution is to stop using the hardware before total failure. If the problem is with a mounted (floppy or hard) disk, you can unmount the offending disk. File system problems with the root file system requires that the system be shutdown immediately.

You may elect to run fsck, but use the -n option. The fsck program can finish ruining a file system already damaged by faulty hardware. The -n option checks the file system but doesn't write anything. A flood of error messages while using fsck is further evidence of hardware (or floppy disk media) failure.

If you run out of space or inodes, you need to backup and remove unused files. You may need an additional hard disk if this is a frequent problem.

MESSAGES	INTERPRETATION
Bad free count	The count of free blocks in the superblock differs from the number of free blocks that exist in the file system; usually, the kernel repairs this on its own
no space	No more free blocks are available in this file system

bad block	The number given as a block address is out of range, that is, it is in the inode area or past the last block in the file system; different than the badblock list and alternate blocks; unmount and run fsck
Out of inodes	No more inodes are available in this file system
bad count	The count of the inodes in the superblock disagrees with the number actually found; usually, the kernel fixes this
no fs	Panic: superblock has gone away
bflush: bad free list	The list of free blocks has a bad block in it; it must be rebuilt by unmounting the file system or going single-user and running fsck
iaddress > 2^24	bad inode number, run fsck

Swapping Errors

The swap space is an extension of memory. Processes that cannot be loaded because memory is already full are loaded into the swap space. They wait in swap space until another process is swapped out to make room for them.

The swap space is not organized as a file system. The swap space is also not treated as a block device: processes are stored as one piece, and the character (or raw) device driver copies between the swap device (usually the root hard disk) and memory without using the block buffers. Thus, file system error messages are irrelevant for the swap space.

Most of these messages have to do with running out of swap space. The solution to this is to reduce usage of the swap space or to increase the size of the swap space. Reducing usage can be done by using pure text versions of the most used programs, or by redistributing usage. Increasing the size of the swap device is not easy. It requires modifying the hard disk (mw) device driver and possibly backing up and rebuilding the root file system.

One error message has to do with disk errors in the swap space. The badblock program should be used to locate the bad block(s) and the mwbad program used to store the bad block number and provide an alternate block.

MESSAGE	INTERPRETATION
No swap space for exec args	Ran out of swap space while trying to exec (load) a program
Not enough swap space to fork	Ran out of swap space while trying to fork
out of swap space WARNING: swap space running out needed: %d blocks DANGER: out of swap space needed: %d blocks	} Ran out of swap space while trying to swap
IO error in swap	Run badblock and mwbad to fix this

Trap and Interrupt Errors

A trap is a special instruction to the 68ØØØ processor. The trap instruction forces the processor to save its current state (registers and status information) and look in a particular memory location for the address of a routine to execute. One trap instruction, for example, is used to make system calls. Generally speaking, you should never see a trap error message as these are mainly for debugging the kernel. A trap error message means that either a program uses an illegal trap instruction or a memory error has occurred that simulated a trap instruction.

Interrupts are like a hardware generated trap instruction. The difference being that trap instructions occur when they are encountered in a program and interrupts happen when a device wants to signal the processor. Usually, a device causes an interrupt by putting a signal on the S-1ØØ bus vectored interrupt lines (/VIØ-7). The interrupt signals completion of an I/O transfer and that the device needs to be serviced.

Most interrupt error messages are produced by the device drivers. These interrupt error messages refer to interrupt lines that are not handled by the device drivers. Stray interrupts are caused by hardware failure.

MESSAGE	INTERPRETATION
trap trap type %d 1111 emulator trap unexpected kernel trap	An unexpected trap instruction has been executed; either a program has the offending instruction or memory has failed
Random interrupt ignored stray interrupt at %x	An interrupt has occurred on a line that does not have an interrupt service routine address installed; possibly incorrectly installed or functioning hardware

Memory Management Errors

Memory management for the TRICEP is performed by the Motorola 68451. The 68451 has two functions: two protect memory from unauthorized or accidental access, and to translate virtual addresses into physical addresses. Memory management routines are hardware dependent, that is, the routines are NOT part of the Unix kernel purchased from AT&T, but are added by the porting company (UniSoft).

As with trap and interrupt error messages, memory management errors should not occur. A memory management error indicates hardware failure, either in memory or the processor board.

MESSAGE	INTERPRETATION
kernel memory management error out of mmu registers unsuccessful mmu load	Hardware failure of processor board or memory

Floppy Disk Error Messages

Floppy disk error messages are produced by the **dj** (Disk Jockey DMA) device driver. These messages report on the condition of: the drive, the diskette inserted in the drive, sectors on the diskette and the controller itself.

The most common error messages will be either messages about sectors where errors occurred (data CRC error) or errors caused by accessing a drive while the door is open or no diskette is inserted. Diskette with bad blocks (sectors) should be replaced. There is no badblock mechanism for floppy diskettes.

Other error messages refer to time-outs. A time-out occurs when the controller has been issued a command but failed to respond in a reasonable amount of time. This usually indicates that the controller cannot access the requested drive because the drive is off or the cables to the drive have been disconnected. Occasionally, the controller may time-out because of a bad command or a hardware fault in the controller itself.

The floppy disk error messages can be divided into three classes: errors that occurred while opening, errors occurring while formatting and errors occurring during reading or writing.

MESSAGE	EXPLANATION
dj%d: drive timeout	No response from drive; drive off or cable misconnected; possible controller problem
djintr() timeout	The controller was given a command that wasn't responded to; probably controller problem

timed out waiting for header	Sector identifying header not found; probably diskette not formatted (or hard sectored)
dj%d: error MESSAGE	The controller has reported the error given by MESSAGE;

MESSAGES follow:

illegal command	The dj controller received an illegal command; software or memory fault
illegal disk number	Attempt to access a drive number greater than 8; software or memory fault
drive not ready	Controller detected that a drive is not ready; open drive door, no diskette, no power to drive or mis-connected cable
disk unreadable	Unformatted diskette
header sync byte not found	Search for a particular sector header failed
sector header CRC error	Sector header contains an error
seek error	Information written in sector header disagrees with the controller's idea of the track number
data CRC error	Error in sector data, data inaccessible through block device
illegal sector number	Controller received request for a sector that doesn't exist; software or memory
diskette write protected	5 1/4" diskette has write-protect notch covered, or 8" diskette has write-enable notch uncovered, during an attempt to write; cover or uncover notch on diskette
data (DMA) overrun	Controller was unable to transfer part of a sector because of a DMA conflict on the bus; dj driver will retry
lost command	Controller error; controller "forgot" what it was doing

Hard Disk Error Messages

Although the hard disk and controller for the TRICEP were chosen to be the most reliable possible, the hard disk is also one of the most common sources of error messages in Unix systems. In the TRICEP, both the root file system and the swap space are on the built-in hard disk (standard configuration). This makes the hard disk especially important, the weakest link in the chain of hardware: you lose your hard disk and the system won't run. And, you had better have been making backups.

MESSAGE	EXPLANATION
mw%d: drive timeout	Drive did not respond during open; possibly drive not on, cables disconnected, controller problem
mwgo: illegal command	The mwstart routine has detected an illegal command; software or memory fault
mw%d: drive timeout	Drive did not respond during open; possibly drive not on, cables disconnected, controller problem, may be temporary
mw%d: error bn=%D, sr=Øx%x	Soft error occur; see full explanation below
mw%d: hard error bn%D status=Øx%x	Hard error occurred; see full explanation below
Not entire drive partition Don't know how many cylinders mw drive %d, block %d bad mwspecial: unknown command	Bad ioctl() calls; software or memory fault

Read and write error messages for the hard disk (mw) device driver resemble those for the floppy driver: drive designation, and a block number followed by some status. There are some differences, however. First, all the returned status codes are different. Second, you can add blocks reported as bad to the badblock list with **mwbad**. And, third, there are both hard and soft errors. Hard errors indicate that the data is unrecoverable after ten retries; soft errors means that retries were necessary because an error occurred.

The block number represents the logical block number within the particular disk partition specified by the minor device number. A clustering of bad blocks in a small range may indicate a damaged track. But, you cannot easily map logical blocks to physical sectors because of skewing and partitions. All you need to do is use mwbad to add an unreadable block to the badblock list. Replace any damaged file from backups. Remember that the mw driver uses 1k blocks and **mwbad** uses 512 byte blocks. Double the block number reported, add 1, then use **mwbad** to replace both of these blocks.

The returned status codes provide additional information about the error. There are only nine error codes to deal with.

STATUS	DESCRIPTION
ØØ	Controller busy - software or spurious interrupt
Ø1	Drive not ready - drive
Ø4	Sector header not found - drive (not formatted)
Ø5	Data not found (no data preamble) - drive
Ø6	Data overrun, DMA arbitration failed - software or CPU
Ø7	Data CRC error, data incorrect - drive media
Ø8	Write fault, read after write failed - drive or media
Ø9	Sector header CRC error - drive media
AØ	Illegal command - software or memory fault

USING THE STANDALONE FLOPPY

The Standalone Floppy included with your TRICEP is designed for extreme measures. Like the MX missile, we hope you never have to use it. Then again, it's not anywhere near as dangerous, and it is easy to use.

The Standalone Floppy has a skeletal UNIX system on it. It has a UNIX kernel that has been modified to use the floppy disk as the root and swap device. There are ten commands and three shell scripts. And, the exact same boot loader as the one written on the boot partition of your hard disk. With these bare bones, and your **tar** backups, you can recreate your UNIX system on a blank hard disk.

Why would you want to do this? Well, two reasons. Early TRICEPs were formatted with 512 byte sectors. The mw device drivers were revised to handle 1Ø24 byte sectors in January, 1985. There is a 5.88% storage increase in capacity and a 1Ø% increase in speed gained by using 1Ø24 byte sectors. This is one reason.

The other reason is that something terrible has happened to your root hard disk. Usually, this means that you can no longer boot your TRICEP. This may be as "minor" as having lost part of the boot loader, or as major as losing important parts of the file system, making it inaccessible. The next section explains how to replace your boot loader from the floppy, and a later section explains recreating your system from scratch (and those **tar** backups we asked you to make).

The Standalone Floppy can also be used for other emergencies. For example, if you have configured your system to come up multi-user (with is:2:initdefault) and forgotten your root password, you can twiddle with the **/etc/inittab** file (see Lost Root Password). Then, you can come up single user and add a new password.

Or, if you are really a UNIX guru, you might try rebuilding the superblock with **adb** after the file system has been trashed. You can also run **fsck** from the Standalone Floppy on the root hard disk, or use the Standalone Memory Test if your system has been crashing.

Can't Boot the Hard Disk

There may be hardware reasons for the hard disk not booting, but we have found that the problem might be a "damaged" boot partition. Since the Standalone Floppy has a copy of the boot loader on it, you can possibly replace the boot loader on the hard disk. The first step is to boot from the Standalone Floppy. Insert the Standalone Floppy and reset the TRICEP. After the Memory test, you will be queried:

```
Start from the floppy?y
```

Your response is the letter "y". This loads the boot loader from the floppy disk. You will then be presented with the regular prompt

```
Press RETURN to continue

 Standalone boot

:[]
```

At this point you should press return and (hopefully) your system will come up. If it fails, your problem is more serious than a trashed boot partition: GO TO the next section. Otherwise, you can type the command

cp /dev/djØboot /dev/mwØboot

to copy the boot loader from the floppy to the hard disk. Then, type "sync" and reset the TRICEP. This time your system should be able to boot from the hard disk. Answer "n" to the query "Start from the floppy?". If you still can't boot, you may have bad blocks in your boot partition.

If any mw error messages appeared while coping the boot loader, use **mwbad** to get an alternate block assigned for them and try the copy again. Remember that the mw driver reports block numbers in 1k size, and **mwbad** believes in 512 byte blocks. Multiply the bad block number by two, and use **mwbad** to add this doubled block number, and the one following it, to the bad block map. For example, if you see

```
mwØ: hard error bn=3267 sr=Øx7
```

use **mwbad** to add block numbers 6534 and 6535 to the bad block map. Then, try the boot loader copy again.

mw(Ø,Ø)unix Not Found

Okay, you have failed when the boot loader tried to find the kernel file (unix) on the root device (mw(Ø,Ø)). The next step is to boot up the Standalone Floppy (see previous section). Once you have the : prompt (after resetting the system with the floppy), type

dj(Ø,Ø)unix

This causes the boot loader to look for the file "unix" on the first floppy drive (dj(Ø,Ø)). This "unix" has been subtly modified to use the Standalone Floppy as the root and swap device. Your TRICEP will slowly load unix, then quickly display the usual Copyright and memory messages, then a $ prompt. You may also see:

```
Insufficient swap space for available memory
Largest runnable process is ####
```

This tells you that the swap space available on the standalone floppy is small compared to the available memory. This is okay since you should limit yourself to small diagnostic and repair tasks that are run one at a time in the foreground. Setting the diskette up this way prevents swap space overwrites to critical areas of the floppy.

The dollar sign prompt ($) means that you have the "half" shell running. The half shell is a very stripped down version of the ordinary shell. It has the **mount**, **umount**, and **sync** commands built-into it. It does not have metacharacter expansion, or PATH, or anything fancy included with it. It's small to save space on the floppy. Generally speaking, all you have to remember is to use full pathnames when you type commands.

Now, you have the half shell prompt, and your hard disk has problems. The next step is to use **fsck** to check the hard disk. Type

/etc/fsck /dev/rmwØa

to run **fsck** on your root device. The usual rule about not using the raw device with the root hard disk can be ignored because the root file system is on the floppy now.

Any luck? If **fsck** can't access the superblock or the root inode, you will have to rebuild your file system from ground zero. This is probably easier than hiring a guru to attempt to fix up your file system with **adb**, but if your problem is minor and you don't have any backups, you MIGHT consider it. Otherwise, try the next section.

If the only thing wrong is that the file "unix" is missing, you can mount the hard disk, copy it across from the floppy, and fix it using the patch program. The commands that follow outline this procedure, assuming that you have booted the Standalone Floppy:

```
mount /dev/mwØa /a
cp unix /a/unix
patch -w /a/unix _rootdev Ø
patch -w /a/unix _pipedev Ø
patch -w /a/unix _dumpdev 1
patch -w /a/unix _swapdev 1
patch -w /a/unix _swplo Ø
umount /dev/mwØa
sync
```

This procedure mounts the hard disk, copies **unix** to it, changes the root and swap devices to the hard disk and unmounts the hard disk. You may now reset and boot UNIX.

Rebuilding Your System From Scratch

Actually, you'll need your **tar** backup diskettes that you made if you followed our directions in the First Time Use section. You will also need any other backups you have made and the Standalone Floppy. Make certain that you have backed up your configuration files in **/etc**: passwd, inittab, gettydefs and rc.

Boot up the Standalone Floppy, following the instructions in the section named "Can't Boot Hard Disk". Once you have the "$" half shell prompt on the screen, you are ready to format your hard disk. This is the point of no return. Once your hard disk has been formatted, everything previously stored there is gone (forever!). I hope you've been making backups.

The Standalone has two scripts for rebuilding either CMI 16 megabyte or Quantum 34 megabyte hard disks. Merely type,

sh /etc/cmi16save.sh
sh /etc/quantsave.sh

and press RETURN to continue when asked to do so. This process takes about 5Ø minutes to complete, and only takes the single RETURN response after it starts. Your hard disk will be reformatted with 1K sectors, checked for bad blocks, given the optimal interleave and have the files on the Standalone copied to it. New devices are made with the MAKEDEV script and **unix** is patched so that the root and swap devices are the hard disk. When the script completes you will see the half-shell prompt ($). Type

sync

and reset your TRICEP. This time your TRICEP should be able to boot from the hard disk in the normal fashion. If it doesn't, you've got hardware problems. Otherwise, the half-shell prompt will appear on the screen, and you will have a skeletal UNIX system to work with. This is where your **tar** backups come in. Insert the first diskette, and type

tar xvf /dev/djØa

and watch while **tar** begins replacing your files. You may notice that **init** cannot be created. This is important, and we'll fix it later. When the first diskette is finished, insert the next diskette, type the same line,

tar xvf /dev/djØa

and continue. The C-shell is on the fifth diskette, so if you are tired of retyping the same line, you can type

csh

to replace the half-shell, type the tar line once more, then simply type !! to repeat the last command:

tar xvf /dev/djØa
!!

It will take about two to three hours to finish extracting your files from the diskettes. But, don't go to sleep yet. You are using something called "tiny init" instead of the real thing. When you have finished with the last diskette, re-insert the first **tar** diskette (it should have **/etc/init** on it). Then type

mv init init.old
tar xvf /dev/djØa /etc/init
sync
ln /etc/init /etc/telinit
sync

and reset your TRICEP. This time you will notice the "INIT: SINGLE USER MODE" message. Then type

rm init.old
rmdir lost+found
mklost+found
creating slots...

The **lost+found** directory must have a large empty directory. The **mklost+found** script creates dummy files and removes them leaving lots of space. **fsck** needs these spaces for reconnecting orphaned files. With this directory in place, you can now run **fsck**. Please do so.

/etc/fsck

Next, you should run **vchk**. This will check to see that all the files, directories and correct permission modes are in place. **tar** creates directories with all permissions for everyone. This is a security disaster. **vchk** will create a script that will fix file ownership and permission modes. Type

vchk -c > results

This will take about 1Ø minutes to complete. When **vchk** completes, you need to extract the lines specifying missing file from the **results** file with an editor. First, collect these lines into a file with

grep take results > missing.files

Then, fix the file permissions and ownerships with the line

grep -v take results | sh

grep will select the lines without the word take and pass them to a shell to execute, to the best of its ability. The shell will fail to change the mode or ownership of missing files, for example. You will need to recover these missing files from somewhere else. Your dealer, perhaps?

Finally, you can add your backed up files and directories back to the system. Remember that the **/etc/passwd, /etc/inittab, /etc/rc** and **/etc/gettydefs** files are back in their preconfigured state. Hopefully you have backups of everything.

Lost Root Password

If you haven't configured your TRICEP with **/etc/inittab** to come up multi-user, you won't have a problem with this. You will be super-user after resetting and can change the password. But, if you have added á line like

```
is:2:initdefault
```

to the **/etc/inittab** file, you will always come up in multi-user mode, and need the superuser password to shutdown, backup or change the password. If you you reconfigured and forgotten your password, you can follow this simple procedure to recover it. Of course, so can anyone who has a Stand-alone Floppy, so keep it locked away.

Follow the directions in the section "Can't Boot Unix" and boot up the Standalone Floppy. Once you have the half-shell's $ prompt on the screen, type these commands:

```
mount /dev/mwØa /a
cp /a/etc/inittab /a/etc/temp
cat > /a/etc/inittab
<control-D>
umount /dev/mwØa
sync
```

These commands mount the hard disk, make a copy of the inittab file named temp, clear the original inittab file (the control-D is the End of File character) and unmount the hard disk.

Now you can sync and reset the TRICEP. When it comes up, the **/etc/inittab** file will be empty and **init** will query the console for a run level:

```
ENTER RUN LEVEL (Ø-6, s or S): s
```

Enter "s" and RETURN. This will bring your system up in single user mode and you will be superuser. Type

```
passwd
```

to change the root password to something you can remember. Then, enter

```
cp /etc/temp /etc/inittab
rm /etc/temp
```

to restore your old **/etc/inittab** file. Good job, doctor.

Running memtest

The memory testing program on the Standalone Floppy is called **memtest.** This is a stand alone program, meaning that you can NOT use it while UNIX is running. To start **memtest**, reset your TRICEP, and with the Standalone Floppy inserted, type

```
dj(Ø,Ø)memtest
```

This loads the primitive but thorough memory testing program. **memtest** will ask for a starting and ending location.

```
Standalone memory test

Starting location: Ø
End location: ffff
ØØ 55 aa ff checker interact5 ...
```

This region, Ø - FFFF, is below where the **memtest** program is loaded in memory. You can't test the memory where the program is loaded because **memtest** writes test patterns throughout memory. To test the rest of memory, reset and restart **memtest** for

```
DRAM 256k 1          1AFD2 - 3FFFE
DRAM 256k 2          4ØØØØ - 7FFFE
DRAM 256k 3          8ØØØØ - BFFFE
DRAM 256k 4          CØØØØ - FFFFE
```

You can only stop **memtest** by resetting, and it continues forever. Notice that memory locations start and end on EVEN boundaries (it crashes if you use odd boundaries). The **memtest** program will ring the "bell" (beep your terminal) each time it successfully completes a pass. Allow at least two passes on each memory board before resetting and trying another.

If an error is discovered, **memtest** displays the location, the data written and the data read (in error). By testing boards individually you can localize the problem to one board. The first pass on a 256k board takes about 5 minutes and the second pass takes about 15 minutes. The second pass involves a more convoluted test pattern for interact11.

EVERYDAY PROCEDURES

This section is intended to guide you through every day activities, like

- booting
- going multi-user, running fsck and entering the date
- checking for free disk space, policing the disk
- going single-user and backing up
- system shutdown

Each section that follows covers one of these topics. Your daily activities may vary from this; this is intended to cover a wide variety of installations.

Some systems may be running 24 hours a day; others may be shut down every evening and restarted in the morning. The things that everyone will have in common is a need to: check for free disk space and a regular backup routine.

Booting

Booting was covered in considerable detail in Step 7 of the Installation section. The only difference you will notice is that the system automatically resets itself when you turn the power on. To refresh your memory, you

1. turn on the console, and any external hard disks
2. turn on the TRICEP (automatically resetting it)
3. press RETURN to continue Standalone boot
4. press RETURN to begin execution

If you are rebooting the TRICEP after a system crash, you will not be turning on the power, but turning the keyswitch to RESET. And, you MUST run fsck every time the system is reset after a crash. Your system may need to be reset because of a hardware failure. The consequences will probably be minor as long as you remember to use fsck.

Going Multi-User

TRICEP systems are delivered so that they are in single-user mode after booting. This can be changed by modifying the /etc/inittab file (see Modified Inittab Example).

You change from single-user mode to multi-user mode by typing

telinit 2

This sends a message to the init process to change to run level 2. Now, in the unmodified /etc/inittab file, this run level matches every entry. After you have modified inittab, this run level will probably match the multi-user mode, although this is merely custom, and not a default. If telinit 2 doesn't wake up the other terminals (with a sign-on prompt from gettydefs), look at your /etc/inittab file and read the section named Inittab Controls Init. Run level 2 starts the daemons in /etc/rc.

Setting the Date

Going multi-user starts the /etc/bcheckrc script. This script displays the date (or what the TRICEP believes to be the date) and allows you to agree or disagree. Please check this date carefully, as it is used in backups and other important system functions.

The date is displayed as the day of the week, the first three letters of the month and the date, and the time in 24 hour notation. In 24 hour notation, you subtract 12 from the hour to convert to "ordinary" time. For example, in

date
Fri Aug 24 16:44:23 1984

16:44:23 is 4:44 and 23 seconds, P.M. Subtract 12 from 16 to get 4 P.M.

Entering a date can be a little confusing too. The date command likes the new date in one particular format, a series of pairs of digits. Each pair represents one part of the date, as in

```
Aug 24 16:44
 \  |  /  /
 Ø8241644
```

The month comes first (January is Ø1, February is Ø2, and December is 12), followed by the date, the hour (in 24 hour notation) and the minutes. You must represent each part of the date with a pair of digits, and you must have all four pairs. Optionally, you can include the last two digits of the year after the other four pairs.

You will need to reset the date every time the power is turned back on. It is considered bad practice to reset the date while in multi-user mode, so the /etc/bcheckrc script is the perfect time to do it.

Running FSCK

The /etc/bcheckrc script also asks you if you wish to check your file systems. The answer is YES, YES, YES! This is so important, a little exaggeration is permissible.

fsck can detect minor deficiencies in the file system before they grow into major ones. There are two reasons for running fsck before going multi-user: first, the file system that you are checking must be idle, and second, because damage to the file system occurs most often from incorrect shutdown procedures.

If you run fsck on an active file system, files that are modified while fsck is running will make fsck think that things are amiss. This can have unpleasant results, as fsck fixes things that aren't broken. fsck also may request that you halt the system and reset without sync (BOOT UNIX - NO SYNC!). This means exactly that. Turn the keyswitch to RESET and reboot. Otherwise you will undo the work that fsck has just done. You can see that this could easily upset people if you were in multi-user mode and needed to reboot.

Damage to the file system occurs from two causes: improper shutdown and hardware failures. Improper shutdown means that the human in charge was lazy or forgetful, and shut the system off without using /etc/shutdown while in multi-user, or sync; sync if in single-user mode. Hardware failure either happens or it doesn't, but proper system shutdown can be controlled.

If fsck asks you for a yes or no (y or n) response, always respond y, unless you KNOW better. The worst that fsck will do to you is to remove two files when only one is incorrect (when both files share a block, that is, a duplicate block). The file that was last updated probably is correct, but fsck can't be sure. The best thing to do is replace both files from backups.

Is there ever a time when you shouldn't run fsck? Yes. If you suspect that your system has hardware problems with the hard disk (you keep getting disk error messages on the console), and fsck discovers lots of problems, stop fsck and have your hard disk checked. fsck can make things worse if the problem is in the hardware and not in the file system, by trying to fix non-existent problems.

Checking for Free Disk Space

One of the most common problems with any computer system is running out of disk space. UNIX has lots of ways to create files. The accounting programs, if activated, create lots of files automatically. Users, if unchecked, will quickly overrun the system with things like a.out, test, tempfile, and a host of other files with names that look much more important. How's a poor system administrator to keep up?

Well, there are several tools at your disposal. Since prevention is the best medicine, you can keep track of the amount of free space available on the hard disk with the df (disk free) command. This simply reports the number of free disk blocks remaining.

```
df /dev/mw0a
2844
```

2844 blocks translates into 1.422 megabytes remaining (df uses 512 byte blocks). Bell Labs recommends that the root device have at least 2 megabytes free for temporary files, and that the file system with user directories have at least 1 megabyte per user free. This is pretty unreasonable, especially if you have a single 16 megabyte drive with both the root and user directories resident. You can strive for at least .5 megabytes free space (1000 blocks) for each user, with a minimum of 1 megabyte total, and you will remain trouble free. If you have less space than this, check for free space often (hourly).

The second set of tools are du and find. du stands for disk usage, and reports on the number of blocks in files or directories. Keep a list of user home directories in a file, for example, /etc/homes. You can use this list with du to see how much space each user has consumed:

```
du -s `cat /etc/homes`
445 /user/bob
223 /user/rik
1022 /user/len
345 /user/kevin
701 /user/norm
122 /user/gayle
```

The file /etc/homes has the list of home directories in it, one per line. The du -s command produces a summary of blocks used by each user. You can talk to users who have taken up more than their share and ask them which files they want you to back up and remove. If they are truculent (and unyielding), you can be obnoxious, too. You, as system administrator, can search out and remove (after backing up, of course) old files.

The find command is ideally suited for uncovering old and unused files. For example,

```
find /user/joe -atime +30 -print > joes.oldfiles
```

creates a file containing the names of files in the /user/joe subtree that haven't been accessed in the last month. You can be nice and point this out to Joe, and you can be tyrannical, and backup and remove the files (using tar and rm) if nothing else works.

The find example in the User's Manual is a classic. This clever command searches the entire file system for files named either a.out (created by cc) or core (created by program crashes) that haven't been accessed in a week. The names a.out and core have certainly lost their meaning after a week, so they are true space wasters. Use

The second find command selects files modified since the last time you used the find command to select files. This obviously won't work the first time you do an incremental backup, but is better for subsequent backups. The -newer option goes back to exactly the time of last backup, whereas -mtime only goes back exactly a day at a time.

Once you have your list of files, you need to copy them to your backup medium. For floppies, use

tar cvfB /dev/djØa 78Ø `cat tar.8.24`

This creates tar archive floppies. Date these diskettes, please.

After you have been backing up your system for a while with tar, you need to consolidate your backups. You do this by using find to collect the names of all files modified in a longer period, say a week, two weeks or a month. For example,

find / -newer tar.8.23 -print > bigtar.9.23

collects all the files modified since tar.8.23 was created. This consolidates all of the backed up tar diskettes by including the latest version of all backed up files in one tar sequence. Now you can start reusing your daily tar backup diskettes. Remember to label diskettes with the date the backup was made, the format (tar) and the volume number.

The original backup, your monthly backups and your daily backups comprise your entire file system in most recently backed up form. If you need to restore your system from scratch, you start with the original tar diskettes (all 4Ø), then use the latest months backup, and finally, the daily backups since the last monthly backup. This will restore your file system to the state it was in at the point of the last daily backup.

cd /
tar xvf /dev/djØa

restores files to their position in the file system.

find / \(-name a.out -o -name core \) -atime +7 -exec rm {}\;

to find and remove them.

Find can also be used to find large files (with -size), files owned by particular user (-user) and files with dangerous permission bits set (like set user-id to root).

Going Single-User and Backing Up

Backing up is the second most important duty of the system administrator. (Running fsck is the first). Running any computer without a complete set of backups is asking for trouble. Even if the hardware never fouls up, some human will accidentally erase a file and come crying for help. Most computer centers discourage this tendency (humans making mistakes) by punishing such behavior with stiff fees. But, whether you are running a business or a school, backups are a must.

Before you can make good backups, you must make the TRICEP go into Single-User mode. Active file systems are changing file systems. If the system is multi-user when you backup, you can easily be backing up a file while it is changing. A backup of a file that changed is useless. At some time convenient to almost everyone (probably inconvenient to you), make the system single user with the shutdown command (used from the root directory while logged in as root):

shutdown 60

This takes about a minute to complete. The init program tries to be polite and gives everyone 20 seconds to finish their business. If you consider this extreme, use a wall (write all) to warn folks amount the imminent shutdown:

wall System going down for backup in 5 minutes!!<control-D>
System going down for backup in 5 minutes!!

The <control-D> terminates the message.

When you see the message SINGLE USER MODE on the console, you are ready to begin backing up. This is a two stage process: selecting the files to backup and copying the selected files.

The find command selects the files on the basis of the last time that the file was modified (written to). There are at least two ways to do this.

```
find / -mtime -1 -print > tar.8.24
find / -newer tar.8.23 > tar.8.24
```

The first find command selects any file that has been modified within the last 24 hours. The -mtime -1 means modified in less than 1 day. If you want to increase the margin, substitute larger values for the 1.

Turning the System Off

There are two ways to go about this, depending on whether you are in multi-user or single-user mode. In single-user mode, you can simply type

sync;sync

wait about ten seconds (the hard disk should be quiet), and turn off the power with the keyswitch, turn off other peripherals (hard disks, printers, terminals, etc.) after turning off the TRICEP.

When you are in multi-user mode, you need to gracefully shutdown the system before you can type your syncs. The /etc/shutdown script will do this for you. Simply type

shutdown

while logged in as root and in the root directory, answer "y" when requested and wait for the SINGLE USER MODE message. Then you can either backup the system (as in the previous section) or type

sync;sync

and turn the power off.

USING THE REST OF THE MANUALS

We don't really have to tell you that you now have access to the somewhat overwhelming resources of UNIX. You already know that. What you probably would like to know is how to learn about what you have. The TRICEP portion of the manuals provides only a brief glimpse of what is available.

This section of the TRICEP manual is a guide to the rest of the documentation. It is intended to get you started on the road to becoming a UNIX guru (one who really understands UNIX). But, you must be warned first that the material presented in the UniSoft documentation is not sufficient for learning UNIX.

As mentioned previously, the UniSoft documentation was written mostly by people at Bell Labs and UC Berkeley. This documentation was written by members of programming staffs for other programmers familiar with the UNIX system. Thus, there is a type of "catch 22" involved: this documentation can help you learn about UNIX if you already know about UNIX.

The documentation takes two forms: manual entries and papers that expand on manual entries. There is not a wealth of examples. There are instead concise definitions of commands, files and system calls. Once you have a grasp of how things work, this becomes useful. At first, and even later, the manual entries can be headache provoking.

We are not apologizing for this. If the documentation were written in a more didactic style, it would expand to encyclopedic size. Instead of 6 inches of dense documentation, you would have 6 feet of readable documentation piled up on your shelf. This wouldn't help you very much. It's still too much to read.

The TRICEP manual and the UniSoft User Guide are both written in a terse but generally easy to understand form. We recommend that you start by reading both of these manuals, and trying the examples provided.

The TRICEP manual tries to follow a "cookbook" style approach: we give you the exact commands needed to perform a specific task, backup, for example. There is also some tutorial material designed to quickly give you a feel for the UNIX system.

The User Guide explains many of the basic commands used in UNIX. But only the section on ed provides you with hands-on working examples. If you don't find working with the User Guide enough help, buy one of the many books written about the UNIX system. Some of these books, those listed later in this section, provide many examples of how to do things. This hands-on approach makes UNIX easier to learn.

Later, you will find that the manual entries and the supporting papers sufficient for your use.

Divisions in the Documentation

There are eight divisions in the documentation:

- TRICEP Installation and Maintenance Guide
- User Guide
- User's Manual
- Administrator's Guide
- Administrator's Manual
- Programming Guide
- Support Tools Guide
- Document Processing Guide

The Guides are papers written about various programs or features of the UNIX system. They are generally more comprehensive than the manual entries and often provide some examples. The Manuals (User and Administration) are a printed form of the information produced by the man command. They are generally terse, but sometimes provide interesting and useful examples. The manual pages are especially helpful for discovering the correct options and syntax for the commands.

The material that follows explains what is defined in each division of the documentation.

TRICEP Installation and Maintenance Guide

This division of the manuals explains how to install your TRICEP. Included in the installation section is information on how to diagnose and correct hardware problems. The rest of this manual provides you with specific information on how to manage your TRICEP system. Much of this material is in the form of tutorials intended to give you a basic understanding of your system. The rest is in the form of commands that can be followed exactly, or with some file name substitution, so that the system can be managed without needing to understand it.

UniPlus User Guide and Manuals

This is your next stop. The User Guide contains moderately simple descriptions of the basics of UNIX: the two main editors (ed and vi) and the two shells (sh and csh). Reading these papers gives you the information that you need to learn more. Remember, though, that reading alone won't teach you anything. It is best to read these sections while sitting in front of your terminal, logged in. Try everything that is suggested, even if an example is not provided. This may be a little frustrating, but produces the fastest results.

The User's Manual consists of the manual pages for all of the User commands, system calls and subroutines, file formats and games. The first section on commands is the one that you will use the most. A permuted index is provided at the beginning of the commands section that can help you locate commands. Look in the middle column of the index for whatever you need. The last column gives you the name of the manual entry. The first column is actually a continuation of the middle column.

Entries in the Manual section often have a number appended after them. This number refers to the subsection of the manual that the entry belongs to. There are six sections to the User Manual:

1. Commands - programs generally accessible and used as tools, with definitions of the options and arguments required, and the type of results produced;

2. System calls - these are the subroutine calls for accessing and controlling the kernel, used by the commands and other application programs;

3. Subroutines - these are the subroutine calls available to the 'C' language, similar to system calls except they do not provide access to the kernel but provide other functions (string manipulation, math functions, i/o processing);

4. File formats - here are the definitions of the file formats used with some of the commands, such as the structure for the file system, gettydefs, and executable files;

5. Miscellaneous files - some information on networking, ascii characters, and, most important, a description of the termcaps file (terminal capabilities); and

6. Games - this is exactly what it sounds like, a listing of games that may be available on your system (unless some sourpuss removed them).

Following this scheme, you would know to look in section 4 (File formats) when you see something like inode(4) or a.out(4). Commands, like cp, ls or tar, are in the Commands Section, so they have a 1 appended to them (cp(1), ls(1) and tar(1)). Commands found in the Administrators Manual have 1M attached, as in fsck(1M) or mkfs(1M).

Document Processing Guide

Here you will find directions for using nroff and troff, the text formatting programs. These programs are intended to be used with particular printers and terminals for typesetting. In addition to the sections on nroff and troff, three special purpose macros are discussed: tbl, for tables, eqn, for equations and mm, for memorandum formatting.

Programming Guide

This division expounds mainly on the 'C' language. The 'C' language and its interface for the 68000 are defined here. The libraries, defined in the Subroutines section of the User Manual, are mentioned again. Lint, a clever program that checks 'C' syntax, is described. EFL, a preprocessor for FORTRAN, is defined also. (This must be used with a FORTRAN compiler that is not provided).

The information on the UNIX operating system in the back of this division will give you a quick overview of how UNIX works (as viewed from 'C' programs).

Support Tools Guide

There are nine tools described in this manual: two calculator programs, four supplements to 'C' (or RATFOR) programs, two program development support tools and the communication program. Of these, I would rate dc (desk calculator), make (for compiling large programs) and awk (pattern scanning) as the most useful. The make program, for example, is used to define interrelationships between files and how to link modules together.

Administration Guide and Manual

The Administrator's Guide provides generalized information on administering to your system. This information is in addition to the more specific information in the TRICEP manual. However, the sections on fsck, lp, uucp and accounting go into much more detail than does the TRICEP manual. These four sections suffer from the typically dense writing style, but are important enough to read.

The Administrator's Manual is similar to the User Manual in that it contains man style entries. The entries in this section were chosen because they are NOT generally available to users. These programs, special files and procedures are limited to the superuser and other special users (like the lp administrator).

An M in a reference to a command, such as fsck(1M), means that the command's entry is in the Administrator's Manual.

APPENDIX: CHECKING FOR HIDDEN DAMAGE

By the time you have removed the TRICEP cabinet from the packing materials, you should have discovered any gross damage suffered during shipping. In spite of being well packed, it has been our experience that shippers sometimes drop boxes hard enough to dislodge even securely fastened PC boards. This means that you may have loose, rather than broken, parts that you can reattach yourself. And by "hidden", we mean inside the TRICEP cabinet. Opening the cabinet is simple and not dangerous, if due caution is exercised.

You do NOT need to check inside your TRICEP unless you want to or are directed to do so by the troubleshooting section in Step 7.

If you have already plugged in your TRICEP, please unplug it. The cover to the metal cabinet is attached by four screws, two along the bottom of each side, that you need to remove before you can slide off the cover. The cover slides off toward the front.

When you have removed the cover, you will have revealed the printed circuit boards. Each board looks like a green or blue rectangle, about 5 by 10 inches (13 x 25 cm) in size, covered with black integrated circuits on one side. Two brass rails secure the boards at the top. The boards should be neatly standing in a row, with white plastic guides holding the boards upright, and a black plastic connector fastening them along their lower edge.

By looking down between the pc boards (there is room for several more boards right in the middle) you will see another, much larger, green printed circuit board that extends from the front to the rear of the cabinet. This board is called the WUNDERBUSS I/O, and holds 14 black connectors, each about 7 inches (18 cm) long. These connectors work identically to one another, that is, they provide the same data and signals to each printed circuit board that is plugged into them. To work properly, the lower edges of the smaller pc boards must be firmly attached to the black connectors. This connection, between the smaller printed circuit boards and the WUNDERBUSS I/O, is sometimes a problem when shipping computers.

Checkpoint:

We are going to check for two things: that the boards are held between the white plastic guides, and that all the boards are firmly connected.

1. To see whether each pc board is still held between the two white plastic guides, look along the left and right ends of the boards. The white guides begin 3/4" (2 cm.) below the top of the boards, and it's easy to see whether each board is resting in the guides, unless you keep your computer under the table or in a dark room.

2. It's pretty hard to see the connection between the boards and the WUNDERBUSS, but you can do two things. You can see that the top edge of each board is about 3/8" (.9 cm.) above the metal sides, and that the tops of all the boards appear to be about the same height. And you can press down on the tops of the boards to reseat them. To reseat boards, press gently with your thumbs on both ends of the top of a board. You can use up to about 10 lbs. (4 kilos) of force pressing down, and can alternate the force from your left thumb to your right to gently rock the board into place. Even if all the boards look seated, try this with at least one board.

If you find a board that is completely unseated, well, reseat it. You will probably need to loosen the retainer rails that are held in place by two phillips screws. If you find a loose board or cable, it should be readily apparent where it connects. There are no fixed slots for the boards; all slots are interchangeable. However, boards should be evenly spaced for best ventilation. The drawings on the following pages show the proper location for cables.

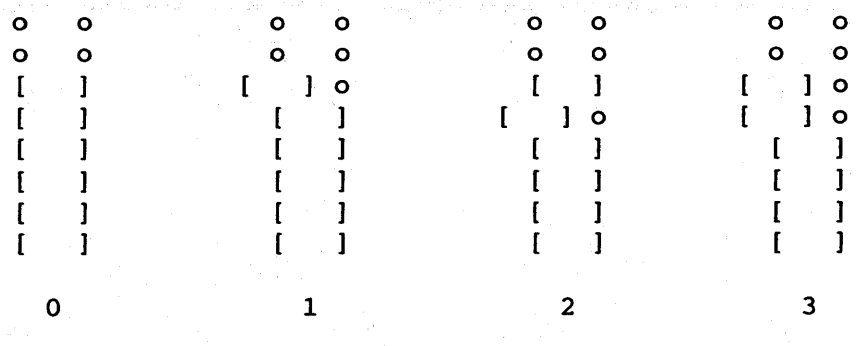
If you can't figure out where to connect a cable, get on the horn to your dealer. These situations are very unlikely, but we want you to be as self-reliant as possible in case they occur.

Memory Board Addressing

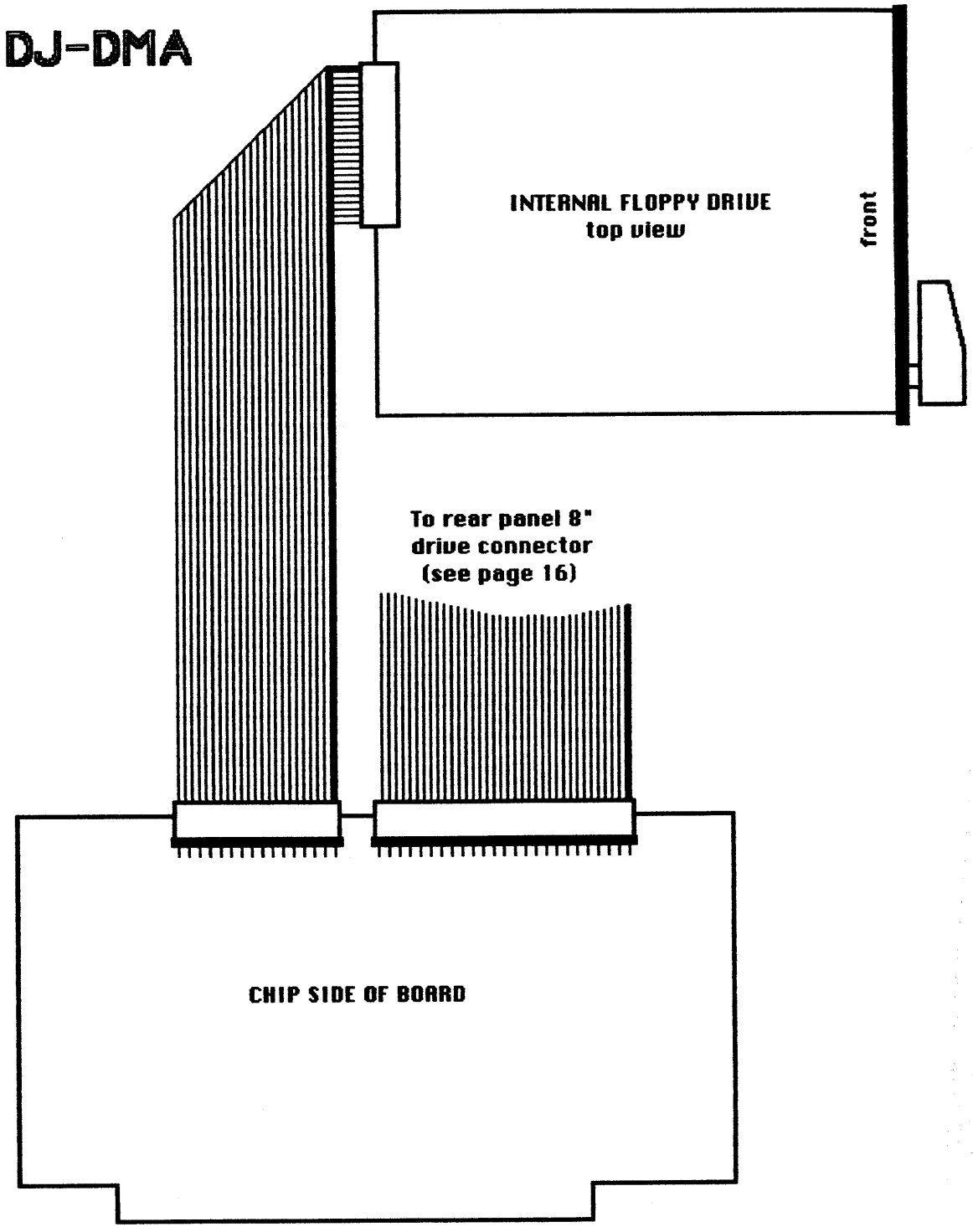
If TRICEP fails to boot, issuing an error message about memory instead (see page 14), first check that all memory boards are properly seated. If the problem persists, you may have one board or more incorrectly addresses. Memory addressing is configured with jumpers in the lower left area of each board.

To examine these jumpers, you must first remove the retainer rails and slide out the memory boards. Memory boards are customarily located in the frontmost slots of the Wunderbuss motherboard. Memory boards can be distinguished by their neat rows of chips and lack of cable connectors.

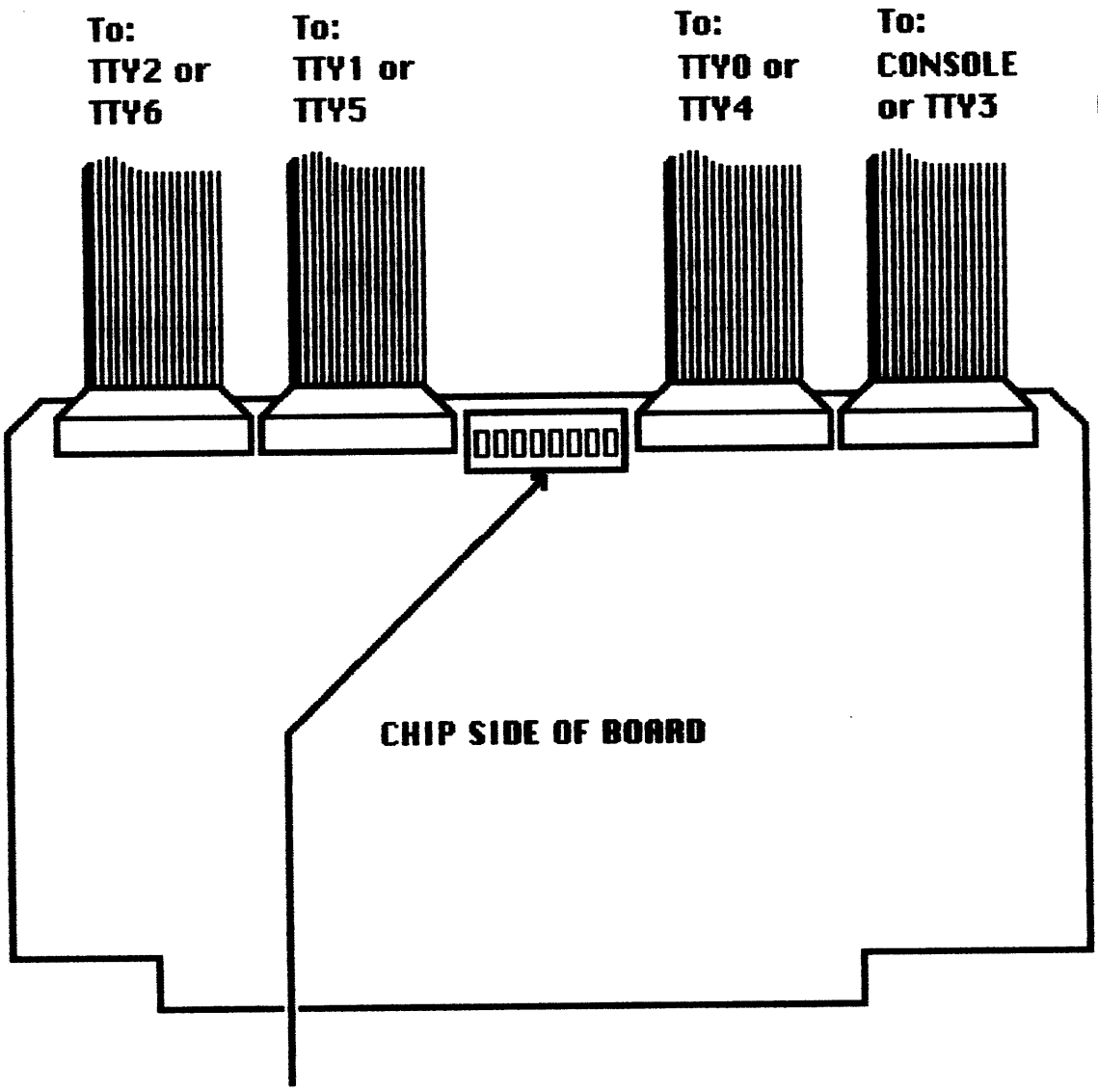
On Morrow 256K DRAM memory boards the jumper settings for the first four boards look like:



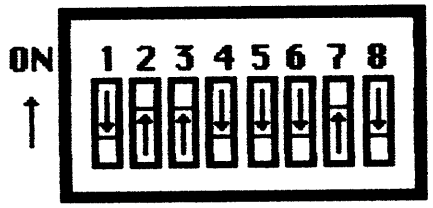
To have correct memory addresses requires that the addresses start at 0, and continue contiguously. If you still fail the memory test after checking the memory boards, please contact your dealer for assistance.


DJ-DMA
INTERNAL FLOPPY DRIVE
top view
front
To rear panel 8"
drive connector
(see page 16)
CHIP SIDE OF BOARD

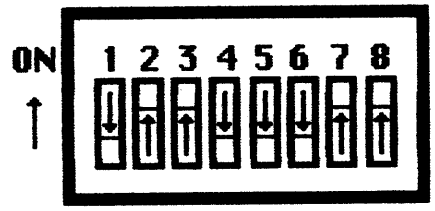
SIO-4



See page 16 for back panel connectors.

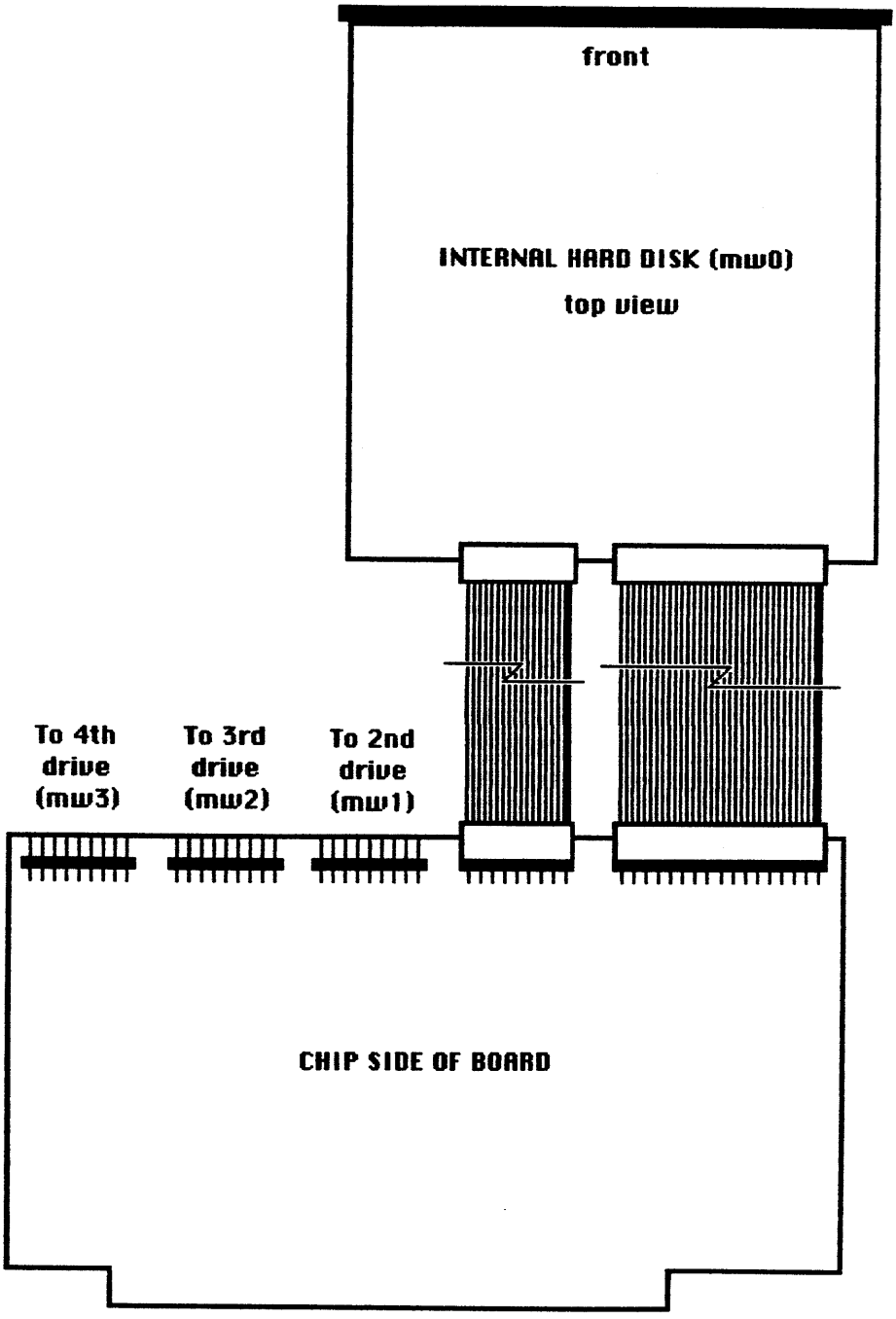


1st board
(CONSOLE - TTY2)



2nd board
(TTY3 - TTY6)

HD-DMA


front
INTERNAL HARD DISK (mw0)
top view
To 4th drive (mw3)
To 3rd drive (mw2)
To 2nd drive (mw1)
CHIP SIDE OF BOARD

ADDENDA TO THE UNIPLUS+ USER'S MANUAL

The following pages contain information on utility programs unique to the Morrow implementation of Unisoft UNIX. The programs are:

CLEAN
CPTREE
DAR
DDT
FAR
HSH
NEWUSER
PATCH
UDOS

NAME

clean - clean CP/M or MS/DOS format text files

SYNOPSIS

clean [-u] file ...

DESCRIPTION

Clean removes return characters from each of the named files and truncates the file upon encountering a CP/M end-of-file character (^Z).

The "cleaning" is done in place. To clean a file for general Unix use, enter the clean file command.

The -u option reverses the process. It stands for "unclean". Use

```
clean -u file
```

to make a file suitable for CP/M programs to look at.

EXAMPLE

To import a CP/M text file, enter

```
far /dev/dj0 -x file; clean file
```

SEE ALSO

far (1) dar (1)

NAME

cptree - copy UNIX file hierarchies

SYNOPSIS

cptree [-v] [source] [destination]

DESCRIPTION

The contents of the source directory and all its descendents are copied into the destination directory. The destination directory must exist, but **cptree** will create any needed sub-directories. If either the source or the destination names are not given, **cptree** will ask for them. If **-v** is in the command line, **cptree** will print each operation in the tree copy as it is performed.

cptree will link together files in the destination that are linked in the source directory. It will also preserve file ownership and permission modes. This makes **cptree** useful for creating exact duplicates of file systems. It does, however, not preserve the previous modify times.

It is possible to copy a tree into a sub-tree of itself. **cptree** will avoid getting into a wormhole.

EXAMPLE

To copy a user's directory tree to a floppy mounted on /f, type

```
cptree /users/mike /f
```

The assumption implicit here is that a **du -s** has been done on /users/mike so that it is known that the files will indeed fit on the floppy.

SEE ALSO

mv(1), cp(1), ln(1)

NAME

dar - access MSDOS format floppy diskettes

SYNOPSIS

dar device [flags] [files]

DESCRIPTION

Dar provides a simple interface to MSDOS floppy diskettes. It may be used for backing up files on diskettes, and recovering them as necessary. "Device" is assumed to be the name of a device special file, such as, dj0 or dj4, corresponding to a floppy drive that contains a MSDOS diskette. Note that dj0 stands for mini floppy 0. Dar can copy files in either direction, and can also be used to read the directory of a MSDOS floppy. When the v flag is used, it reports the amount of space still available on the diskette.

Names such as /dev/dj0 may be abbreviated as dj0. You can check the names of the storage devices by entering

ls /dev/d* (return)

The flags are:

c Create. Start a new dos diskette from scratch. Remove all files, clean out the file allocation table. This option must be used to copy files to a freshly formatted diskette. It may be used in conjunction with any of the other arguments, and if so, it is performed first.

d Delete. Delete the named files from the floppy diskette. If no file names are given, no action is taken.

p Print. Each named file is sent to the standard output (most likely your terminal).

r

Replace. Each named file is copied to the floppy. If no file names are given, all the files in the current directory are copied. If a file of the same name already exists on the floppy, it is silently overwritten.

t Table. Print a table of contents of the floppy diskette. If any files are named, only those named and present are listed.

x Extract. Each named file is copied from the diskette to the current directory. If no file names are given, the entire contents of the diskette are extracted.

v Verbose. Files deleted are preceded by a "d", those replaced are preceded by an "r", those extracted are preceded by "x". The percentage of the diskette that is occupied, along with the number of bytes still available, is reported when dar is finished.

At most, one of the flags - d, t, r, x - may be present. If none are present, -t is assumed. -v may be used in combination with any of these as in "-xv".

The file list may contain wild card matching patterns to be applied to the set of file names in the UNIX directory. Refer to the section "Argument Expansion" in the sh program writeup in this manual.

EXAMPLES

You receive a MSDOS diskette and want to copy the files off. To copy its entire contents to your TRICEP, insert the diskette into floppy drive 0 and type:

```
dar dj0 -xv
```

To copy a file from a diskette, changing its name in the process:

```
dar dj0 -p filename > newname
```

To copy a file from diskette into a file that is elsewhere than your current directory:

```
dar dj1 -p name > /dir/...
```

(path from root to desired location; you could also specify a new name)

The following discussion is intended primarily for users that are having trouble getting dar to do what they want it to.

One aspect of dar command syntax somewhat complicates the issue of using it, namely, the use of metacharacters. An example should help clear this up. Let's say your current directory contains two files, a.com and a.bak. Your MSDOS floppy contains a file a.pil that you wish to copy into your current directory. Since this is the only file on the floppy that begins with "a", you lazily enter dar dj0 -xv a*. Next you are wondering what's wrong when you are faced with the message "a.com not found on diskette / a.bak not

found on diskette". Of course they weren't found. You didn't even want dar to be looking for them.

What has happened is this: Your command interpreter (shell) thought you meant to search the current directory for a match to the wildcard argument before going out to the floppy for the copy. It indeed found two matches (a.com & a.bak), so dar went gleefully out to the floppy to copy them. In vain, unfortunately.

The solution here is to remember that **dar** does not perform metacharacter expansion (* or ?).

Now, just to set the record straight, there is a further complication that may help you to understand how the shell operates. Suppose the file on your floppy is called b.pil instead of a.pil. If you enter dar dj0 -xv b.* (no quotes), the file will in fact be copied, contrary to the previous discussion. Furthermore, everything on the floppy diskette will be copied, much to your chagrin. This is because your shell took b.* to your current directory, looked to see if a file of that name was really there, and finding no such file, it passed the command to dar as " dar dj0 -xv ". As noted above in the description of the x flag, if there is no file specified, the whole diskette gets copied.

SEE ALSO

far - to access CP/M diskettes.
udos - to run MS/DOS programs.

NAME

ddt - a program for examination and manipulation of non-text files

SYNOPSIS

ddt [file]

DESCRIPTION

ddt can be a handy tool for programmers and others who need to manipulate arbitrary (i.e. non-ASCII text) files. It is similar to CP/M's DDT, but the Digital Research version works with memory, while this one works on files. Changes made to a file under **ddt** are immediately effective. There is no intermediate buffer as with some editors.

Addresses (mentioned below) are given in hexadecimal, (up to six digits long). If the first character of an address is an ' (apostrophe), the value of that number will be that of the ASCII character following.

The following are valid addresses:

```
0    6    b    B    'y
c3b0      78ba4  'A      FFFF '/
```

Address list may be comma-separated or space-separated.

The following are valid address lists:

```
0,ffff    0,5,7     0,3000,'a 'a,'b,'c
```

No command takes more than three addresses. Extra addresses are ignored.

ddt always remembers the last file address accessed. In most cases, if you give no address on the next command, this old remembered address is used.

COMMANDS

d - display

Takes 0, 1, or 2 arguments. Displays the contents of the file in hex bytes on the left and ASCII characters on the right. If no address follows **d**, the next 128 bytes, starting at the given location, are displayed.

Finally, if two addresses are given, that part of the file which lies between the two addresses is displayed.

Examples:

```
d 0, ffff

d

d 55

d a000
```

f - fill (requires three arguments)

```
f A, B, C
```

The portion of the file which lies between addresses A and B (inclusive) is overwritten with a flood of the character C.

If C is large, then the low order byte of C is used.

Example: f 0, 3000, e5 writes E5s from 0 to 3000 hex.

r filename (Read)

The file on which ddt is operating is changed to filename if possible.

Examples

```
r new
r /usrsrc/cmd/pbx.c
r /etc/dtab
r a/b
```

s (address) Substitute

Takes 0 or 1 arguments. if no argument is given, the currently remembered file address is used. The substitute command allows you to examine and optionallly change the file on a byte-by-byte basis. After entering the substitute command you will be in "substitute" mode until you exit it by entering a single period on a line. If while in the substitute mode you press RETURN, the byte you were looking at will remain unchanged and you will be stepped to the next byte. If you enter a byte value and then push RETURN, the byte will be changed to the byte entered.

/ (address)
// (search)

Ddt has the ability to search for patterns in a file. When the "/" command is given, you will be asked to enter a series of bytes to search for. If an address follows the

"/", the search commences at the given address in the file. Without an address, the search commences at the current file address.

The "//" command is a shorthand to search again for the previously given pattern.

NAME

far - floppy archiver

SYNOPSIS

far device [flags] [files]

DESCRIPTION

Far provides a simple interface to CP/M floppy diskettes. It may be used for backing up files on diskettes, and recovering them as necessary. "Device" is assumed to be the name of a special file (such as /dev/dj0 or /dev/dj4) corresponding to a floppy drive that contains a CP/M diskette. Far can copy files in either direction, and can also be used to read the directory of a CP/M floppy. When the -v flag is used, it reports the amount of space still available on the diskette.

Names such as /dev/dj4 may be abbreviated as dj4. You can check the names of the storage devices by entering

```
ls /dev/d*
```

The flags are:

c Create. Causes the previous contents of the diskette to be thrown away. The entire directory is cleared prior to other operations. This won't have any effect on the actual diskette unless the -r or -d flag is specified. This flag also has the effect of disabling the check for a CP/M diskette. Normally, far refuses to write on a diskette which doesn't look like a CP/M diskette. The -c flag makes any diskette look like a CP/M diskette.

d Delete. Delete the named files from the floppy diskette. If no file names are given, no action is taken.

p Print. Each named file is sent to the standard output (most likely your terminal).

r Replace. Each named file is copied to the floppy. If no file names are given, no action is taken. If a file of the same name already exists on the floppy, it is silently overwritten.

t Table. Print a table of contents of the floppy diskette. If any files are named, only those named and

present are listed. The contents of the diskette are not changed.

uN User number. The default user number is zero. The letter u followed immediately by a decimal number will cause any of far's actions to happen with respect to this user number. With -t, only files with the same user number show up. With -r, files are created on the diskette with the specified user number. With -x, only files with the same user number are extracted. Most files on most diskettes are "user zero" and you don't usually have to worry about the -u option flag. The -u flag may be embedded in the flag argument as in "-xu14v".

x Extract. Each named file is copied from the diskette to the current directory. If no file names are given, the entire contents of the diskette are extracted.

v Verbose. Be verbose about it. Each operation is printed along with the affected file name. Files deleted are preceded by a "d", those replaced are preceded by an "r", those extracted are preceded by "x". The percentage of the diskette that is occupied, along with the number of bytes still available, is reported when far is finished.

At most, one of the flags - d, t, r, x - may be present. If none are present, -t is assumed. -v may be used in combination with any of these.

The file list may contain wild card matching patterns to be applied to the set of file names in the CP/M diskette's directory. Refer to the section "Argument Expansion" in the sh program writeup in this manual.

EXAMPLES

You receive a diskette in CP/M format and want to copy it to your TRICEP system. To copy its entire contents to Unix, insert the diskette into floppy drive 0 and type:

```
far dj0 -xv
```

To copy a file from a diskette, changing its name in the process:

```
far dj0 -p filename > newname
```

To copy only the ".doc" files over from a CP/M mini-floppy:

```
far dj0 -xv "*.doc"
```

To view the files of user 3 on a CP/M mini-floppy:

```
far dj0 -u3
```

To copy them off:

```
far dj0 -u3x
```

To copy a file from diskette into a file that is elsewhere than your current directory:

```
far dj4 -x name > /dir/...
```

(path from root to desired location; you could also specify a new name)

The following discussion is intended primarily for users that are having trouble getting far to do what they want it to.

One aspect of far command syntax somewhat complicates the issue of using it, namely, the use of quotation marks. An example should help clear this up. Let's say your current Unix directory contains two files, a.com and a.bak. Your CP/M floppy contains a file a.pil that you wish to copy into your current directory. Since this is the only file on the floppy that begins with "a", you lazily enter far dj0 -xv a*. Next you are wondering what's wrong when you are faced with the message "a.com not found on diskette / a.bak not found on diskette". Of course they weren't found. You didn't even want Unix to be looking for them.

What has happened is this: Your command interpreter (shell) thought you meant to search the current directory for a match to the wildcard argument before going out to the floppy for the copy. It indeed found two matches (a.com & a.bak), so Unix went gleefully out to the floppy to copy them. In vain, unfortunately.

The solution here is to use quotation marks around your filename(s). Had you entered far dj0 -xv "a*" , the shell would have left the command alone, and far would have gone directly out to the floppy to find anything that began with an "a". Success.

Now, just to set the record straight, there is a further complication that may help you to understand how the shell operates. Suppose the file on your floppy is called b.pil instead of a.pil. If you enter far dj0 -xv b.pil (no quotes), the file will in fact be copied, contrary to the previous discussion. Furthermore, everything on the floppy diskette will be copied, much to your chagrin. This is

because your shell took b.pil to your current Micronix directory, looked to see if a file of that name was really there, and finding no such file, it passed the command to Unix as " far dj0 -xv ". As noted above in the description of the x flag, if there is no file specified, the whole diskette gets copied.

In summary, you should enclose the argument in quotes whenever you want to copy specific files from the floppy to Unix. The only exception to this is a situation where you have certain files already established on Unix and you want to extract

SEE ALSO

dar(1), sh(1), csh(1)

NAME

hsh - compact shell designed for use with Standalone Floppy

SYNOPSIS

sh (linked to hsh) invoked automatically by init when booting from Standalone Floppy

DESCRIPTION

hsh is a compact version of the normal sh shell designed expressly for use with the Standalone Floppy. It differs from sh in these ways:

1. allows no wildcard expansion.
2. has three internal commands: mount, umount, and sync
3. search path is limited to /bin and /usr/bin
4. only error message returned is "?"
5. prompt is "$"

SEE ALSO

sh (1)

NAME

newuser - create a new user account

SYNOPSIS

newuser

DESCRIPTION

Newuser automates the process of creating new user accounts. The superuser (root), the only user allowed to run **newuser**, has only to enter the login and full name of the new user, and the program does the rest. The "rest" includes adding an /etc/passwd entry, creating the user's HOME directory, copying intialization files, like .profile and .cshrc, from /usr/lib, and changing the ownership of the HOME directory and files to the new user. It is a good practice to add a password to the new account with **passwd**, and to try out the new account by logging into it after creating it.

As mentioned, files in the /usr/lib directory that begin with a dot (.) and that are listed below (in Files) will be copied into the new user's HOME directory. These files are intended to form a model for the new user to edit. They provide for the user's local environment, like the PATH, PROMPT, and TERM variables. Other programs or shell scripts can be added to these files as desired. More information is available about these files in the section labeled "Controlling Shells" of the TRICEP manual.

EXAMPLES

To add a new user, John VanderWood, to your TRICEP system, you would type the command and answer the prompts:

```
# newuser
User's real full name: John VanderWood
New user's login name: john
New user added.
```

You can add more than the full name, for example, a telephone number, if you like. Everything after the full name prompt gets added to the comments field in the /etc/passwd entry. User names must be short and all lower case.

FILES

/etc/passwd, /usr/lib/.profile, /usr/lib/.setup, /usr/lib/.login, /usr/lib/.cshrc, /usr/lib/.dosrc, /usr/lib/.logout

SEE ALSO

passwd(1), sh(1), csh(1), udos(1), passwd(4), group(4)

WARNINGS

THE **newuser** COMMAND LOCKS THE /ETC/PASSWD FILE WHILE IN USE. THIS MEANS THAT NOONE CAN LOGIN OR ACCESS THE PASSWORD FILE UNTIL THE SUPERUSER COMPLETES THE COMMAND.

NAME

patch - limited non-interactive version of adb for TRICEP Standalone Floppy

SYNOPSIS

patch [-bwl] a.out _symbol value

DESCRIPTION

Patch is a debugging tool designed expressly for use in the limited space of the Standalone Floppy. Its primary purpose is patching the floppy's version of the file unix for the appropriate root, swap, etc. devices. This is part of repairing a missing or damaged unix on the hard disk.

-b byte patch of symbol
-w word patch (2 bytes) of symbol
-l long word patch (4 bytes) of symbol

a.out object file with symbol table

_symbol underscore-prepended value in symbol table

value hex value to patch _symbol

EXAMPLE

The following command puts the 16-bit (-w) value of 0 Hex into the symbol rootdev in the file /a/unix:

```
patch -w /a/unix _rootdev 0
```

SEE ALSO

adb (1)

NAME

udos - UNIX-DOS, MS/DOS emulator

SYNOPSIS

udos [options] [file] [arguments]

DESCRIPTION

udos provides a system call interface consistent with Microsoft's MSDOS operating system. **udos** also allows the use of some of the built-in commands that are a part of the DOS COMMAND.COM command processor. The intent of **udos** is to enable users to run software that is written for the popular MS/DOS operating system, while still enjoying the power and flexibility of the UNIX operating system.

Any program that uses the DOS system call interface will work with **udos.** Programs that are designed to work specifically with the IBM PC, that is, that make references to the ROM or the video-mapped memory for graphics, will **NOT** work with **udos.**

There are three options that are recognized by **udos.** Any other options (a single character preceded by a dash (-)) will be passed to the DOS program being invoked by **udos.**

t This is the most important option. The "t" stands for terminal emulation. Including this option in the **udos** command line causes terminal control characters to be translated from those required by ADM31 terminals to the terminal specified in the TERM or TERMCAP environment variables.

c This is a debugging flag, intended for creating a summary of DOS system calls processed by the simulator. A report is sent to the standard output when **udos** is terminating that lists the number of the system call (in hex) and the number of times the system call was made.

v Another debugging flag, this one produces a display similar to the DOS debugger register display. One "-v" produces a display every system call, two -v's or three -v's result in more frequent displays. This may be useful to program developers, but is really a remnant from the development of **udos.**

If you wish to pass an option to a DOS program that uses the letters t, c or v, merely capitalize the letter and **udos** will ignore the option flag and pass the option to the DOS program as a command line option. For example,

udos conf -T

passes the option -T to the DOS program conf.com without activating terminal emulation. If "T" hadn't been capitalized, terminal emulation would be turned on, even though the option follows the DOS program name.

TERMINAL EMULATION

The terminal emulation ability of **udos** works most acceptably with programs that produce a relatively low volume of character output. Spreadsheet programs, like SuperCalc II, are low volume users of character I/O. Wordprocessing programs, like NewWord or WordStar, are examples of high volume users of character I/O. The terminal emulation ability built-into **udos** uses the **curses** library routines. These routines can result in a faster character output in the low volume I/O programs, and faster execution. In high volume I/O programs, the overhead in using terminal emulation will outstrip the performance gained by using the **curses** library routines. If you are using **udos** at a low baud rate (via a modem, for example), terminal emulation will probably result in higher performance even for wordprocessing programs.

There are two preconditions for using terminal emulation. The first is that the DOS program must be installed for an ADM31 terminal. This is a terminal type with most of the intelligent capabilities provided by newer terminals. The second requirement is that the TERM or TERMCAP environment variable be present. Morrow TRICEP systems are factory configured (in .setup or .cshrc files) to invoke the **tset** command and set the environment variables TERM and TERMCAP. If you have edited or removed these setup files from your home directory, you can restore them by copying the prototype files from /usr/lib.

DRIVE NAME ASSIGNMENTS

The DOS operating system provides for a hierarchical directory structure similar to UNIX. Things are complicated, however, because DOS is designed for use with floppy disk drives, and uses a combination of drive letters (like A:) and pathname specifications.

udos handles drive name assignments by mapping the drive letter to the pathname of a UNIX directory. For example, typing

```
udos
```

assigns drive A: to be your current directory. When you enter a drive name while using **udos** the real pathname transparently replaces the DOS drive name.

You can assign drive names to UNIX directories explicitly

with the assignment operator (=). For example,

```
A>c:=\users\len\sc
```

assigns drive C: as the directory /users/len/sc. Notice that backslashes (also called "slants") were used in between directory names. This is an MS/DOS convention, but ordinary slashes will also work with **udos**.

udos will also set up drive letters A:, and possibly B:, automatically during invocation. If you invoke **udos** without specifing a program to run (interactive mode), drive A: will be set to you current directory. However, if you invoke **udos** with the name of a MS/DOS program that is not in your current directory, the directory containing the MS/DOS program will be assigned to drive A:, and your current directory will be assigned to drive B:. For example, typing

```
udos /usr/dos/sc/sc2
```

from your home directory will cause drive A: to be /usr/dos/sc, and drive B: to be your home directory. You will be logged onto drive B:. This allows program files (and overlays) that have the SYS attribute to be accessed while logged onto another drive, a feature of many MS/DOS programs.

The drive assignments are displayed automatically when you enter **udos** without a DOS program name (interactively). Current drive assigments can be displayed by typing an equal sign (=) while in interactive mode. Drive assignments may also be contained in dosrc files (see below).

BUILT-IN COMMANDS

udos implements some of the MS/DOS built-in commands. These commands are:

date	displays the MS/DOS date
time	displays the MS/DOS time
type	send a file to the standard output
del	remove a file
dir	display directory
cd	change to a subdirectory of a drive
ren	rename a file

There are several possible causes for confusion in this list. Date and time are displayed in MS/DOS format. But, you cannot set either the date or the time from **udos**. The "cd" command sets both the current directory AND the sub-directory of a drive name to be used. Thus, the following sequence

```
A>cd b:\other
A>
```

establishes the subdirectory the B: drive to be the current directory for THAT drive. Any references to drive B: then refer to the subdirectory (not the root directory).

The wild card metacharacters "*" and "?" are not implemented, so that the command

```
dir *.c
```

will not produce the expected result (a listing of the files in the current directory that end in .c).

The copy command is missing, but can implemented using an alias (see below). All of the commands associated with the processing of batch files are also missing. Users wishing to emulate MS/DOS batch files are advised to use the more powerful shells and utilities provided by UNIX. Redirection of output is also lacking from **udos**, but, once again, can be accomplished by invoking **udos** from a shell. For example,

```
udos program > file
```

redirects the output of "program" to "file".

ALIASES

An alias feature, similar to C-shell's, has been included with **udos.** Aliasing essentially works by replacing one string with another. In practice, this allows users to define short, mnemonic, strings to be replaced with longer and more complex ones. Argument substitution, a feature of aliasing under the C-shell, is not supported. The following list contains several examples of useful aliases (the exclamation point cuases a temporary escape to a Bourne shell):

```
alias ls !ls7 -F
alias copy !cp
alias ren !mv
alias mkdir !mkdir
alias rmdir !rmdir
alias n nw myfile
```

The first alias, ls, is replaced with the string "!ls7 -F" anytime "ls" is typed after the DOS prompt. The alias "copy" causes the UNIX command "cp" to be executed in a sub-shell. The next three examples show how you can replace DOS built-ins with UNIX commands. A warning is necessary here: when you invoke a shell (with a an exclamation point), you can't use drive letters like a: as part of a

filename.

The last example allows the execution of a 8 character command with two keystrokes. Aliases may be included in dosrc files.

DOSRC FILES

The **udos** program works in a manner similar to the other UNIX shells, and shares a feature common to both shells: runtime files. There are two runtime files that are used with **udos:** /usr/lib/dosrc and HOME/.dosrc. dosrc files may contain anything that can be typed after the command prompt (A>). The dosrc files may contain aliases, drive assignments and names of programs to execute. The commands in the file /usr/lib/dosrc are executed first. The commands in the user's own .dosrc file in his home directory are executed last, possibly overiding drive assignments or aliases made by the system-wide /usr/lib/dosrc file. These files are optional; there is no penalty if they don't exist.

COM, EXE and BAT EXTENSIONS

MS/DOS recognizes four filename extensions (three characters following a dot in a filename) as special. **udos** will recognize three of these four. The fourth extension, SYS, refers to special files that modify the operating system and is not relevant to **udos.**

When you issue a command to **udos,** you don't include the file extension. **udos** will automatically try appending the extensions .COM, .EXE and .BAT to the name of the command, in that order. If you specify a complete pathname (beginning with a /), only the specified directory will be searched. If a relative pathname is used, (one not beginning with a /), **udos** will search all the directories contained in the DOSPATH environment variable. Format of the DOSPATH environment variable is just like the PATH envirnment variable, a series of UNIX directory path names separated by colons.

INSTALLING SOFTWARE

Tricep systems built after January of 1985 include the /usr/dos directory. This is intended as the repository for directories containing MS/DOS software. The /usr/dos/sc subdirectory, for example, contains the programs and files used by the MS/DOS version of SuperCalc II. It is suggested that other software packages are also installed in subdirectories of /usr/dos, although this is not a requirement. The following script outlines the installation of NewWord.

```
cd /usr/dos
mkdir nw
```

```
cd nw
dar dj0 x
udos nwinstal
(install NewWord by selecting a terminal and printer)...
chmod 4444 *; chown bin *; chgrp bin *
chown bin . ; chgrp bin . ; chmod 555 .
```

The suggested procedure is to create a new subdirectory (/usr/dos/nw), change directory to it, extract the files from the MS/DOS diskette containing the programs (with dar), run the installation program (under **udos**) and change the mode and ownership of the files in this new directory.

The mode "4444" has a special significance to **udos.** Permission mode 4444 means "Read permission for all, and set the user-id". **udos** interprets the set user-id bit as designating a file as a "SYSTEM" file, which allows some programs to access files in the A: drive while logged onto another drive.

If re-installation of a program is necessary, you will need to change the permission mode to "Read and Write" (4644) before **udos** will allow alteration of the file. Although **udos** is owned by "root" and has the set user-id bit, it checks file permissions and does not allow writing to files that normally are off limits to the "real" user. In a file set to be Read-only for all, this even includes the superuser, root.

As noted in the section on Terminal Emulation, you must install software that will be used with the emulator for the ADM31 terminal. This allows one copy of a program to be used with several different terminals. If you are not using terminal emulation, a copy of the program must be installed for each type of terminal connected to the system.

Printer interfaces should be installed as the LST: device. If you are allowed graphics as an option, disable the graphics. There is a special case where you have connected an intelligent graphics device to your Tricep, and the program drives this particular device as if it were a printer. In this special case, graphics will work.

PRINTERS AND UDOS

MS/DOS programs that wish to direct output to printers should use the LST: device. The LST: device currently defaults as the /dev/cent device under UNIX. This is acceptable if you are the only user of a system that has a printer connected to the /dev/cent port. Otherwise, there will be trouble.

When multiple users direct output to /dev/cent, the output

will be combined in an unpredictable fashion (garbage, in other words). Or, if you are using a serial printer, you will never see any printed output. The solution to this is to use the DOSPRN and DOSPRNBAUD environment variables.

The DOSPRN variable is set to the name of the file that you wish to redirect your LST: (printer output) to. This may be the name of a serial port with a printer attached to it. In this case, you must also have set the environment variable DOSPRNBAUD to the correct baud rate for your printer. For example,

```
setenv DOSPRN /dev/tty2
setenv DOSPRNBAUD 1200
```

establishes /dev/tty2 as the file to redirect the LST: output to, and attempts to set the baud rate of this port to 1200 baud for C-shell users. Bourne shell users need to use

```
DOSPRN=/dev/tty2
DOSPRNBAUD=1200
export DOSPRN DOSPRNBAUD
```

to accomplish the same effect. The C-shell commands can be added to the user's .login file, and the Bourne shell commands to the user's .profile file in their home directory.

If you do not want to send output directly to a printer, substitute a file name instead of a device name for DOSPRN. You can also add an alias to your .dosrc file to send the file to lpr and clear the file for new input. For example, if you have set DOSPRN to be dosprint in your home directory, the alias

```
alias print !(lp -c $HOME/dosprint; echo > $HOME/dosprint)
```

copies dosprint to the spool directory and then readies the file for more LST: output. Notice that this is not used for printing any file while under **udos**, but only for directing that the output sent to the LST: device be sent to the lp daemon for queuing and printing.

FILE NAMES

MS/DOS is a single case system. **udos** compensates for this by changing uppercase letters in pathnames to lowercase. This can result in unexpected difficulties if you normally mix upper and lower case letters in your file and directory names. For example, you wish to assign your directory /users/rik/TRI to be drive C: with the command

```
A>c:=/users/rik/TRI
```

Select a directory for Drive C:

udos translated the letters "TRI" to "tri", so the assignment failed.

MS/DOS places other restrictions on file names. Only the first eight characters (up to the first .) are used, then the first three characters (if any) after the dot. Longer filenames are truncated. As an example, suppose you wanted to access the file enhancements.text while using **udos.** The name gets truncated (twice!) and becomes enhancem.tex, which won't correspond to the original file name.

The set of characters permitted in MS/DOS filenames is:

```
A-Z     0-9     $       &       #       !       (
-       @       ^       {       }       ~       '       `
```

Since some of these characters have a special meaning to the UNIX shells, care must be taken in creating filenames from **udos.** If you stick with names using only letters and numbers, you won't have any difficulties.

EMERGENCY ESCAPE

The interrupt character (default is DELETE) that is used under UNIX will not work under **udos.** MS/DOS programs use a variety of characters for special functions other than interrupting a runaway program. However, there is a special mechanism built into **udos** that will allow you to regain control of your terminal if a program has crashed.

Four tildes (~~~~) will terminate **udos** after a program crash. The four tildes may also terminate a running program if you can type enough of them fast enough. The catch is that **udos** waits until there are four tildes in its input buffer before terminating. If the program is still alive and calling for input, it may be difficult to get the four tildes into the buffer before the program reads (and clears) the buffer. Sometimes holding the ~ has the desired effect.

EXAMPLES

We will assume that you have correctly installed the MS/DOS program Super Calc II for an ADM31 terminal. Then, if you type the following from your home directory,

```
udos -t /usr/dos/sc/sc2
```

terminal emulation will be on, drive A: will be assigned to /usr/dos/sc, drive B: to your home directory, and you will be logged onto drive B:. Being "logged onto drive B:" in this case is equivalent to saying that your home directory is also your current directory. In practice, it is better

to use a C-shell alias to shorten this command line, as in

```
alias sc2 "udos -t /usr/dos/sc/sc2"
```

FILES

/bin/udos	
/usr/lib/udos.sp	(gets loaded into the SP188)
/usr/lib/dosrc	(system wide DOS rc file)
HOME/.dosrc	(user's own DOS rc file)

SEE ALSO

dar(1), csh(1), sh(1), chmod(1), mkdir(1), lp(1)

DIAGNOSTICS

If you don't have an SP188 slave and try to use **udos** you will get the diagnostic

```
No MSDOS cells available
```

If there are more instances of **udos** than there are sp-188 boards, then the same message is displayed. Accidently (or deliberately) running a non-MSDOS program will result in the message

```
Uninitialized interrupt
```

A program with bugs may also cause this. Use ~~~~ to escape. **udos** traps on signals, and will reset the terminal to its previous state before exiting.

BUGS

Perhaps calling these "bugs" is going too far. Wildcards and command line editing are missing from this version of udos. It is suggested that the user make use of the power of UNIX shells instead of trying to get a poor overworked slave to try to duplicate it.