



PRO/UNIX™

UNIX™
for

Professional™

Installation and System Manager's Guide

digital
software

PRO/VENIX™
for the Professional

**Installation and
System Manager's Guide**

Developed by:

VenturCom, Inc.
215 First Street
Cambridge, MA 02142

Digital Equipment Corporation
Maynard, MA 01754

The software described in this manual is distributed as part of Digital Equipment Corporation's Digital Classified Software (DCS) Program. This program enables software developers to submit their software products to Digital for testing according to Digital quality standards for third party software. This software product has met the DCS standard specified in the software product description (SPD) for this product. You should refer to the SPD for information about these standards, the hardware and software required to run this product, and warranties (if any warranty is available).

The software described in this manual is furnished under a license and may only be used or copied in accordance with the terms of that license. This manual is reproduced with the permission of VenturCom, Inc.

Copyright © 1983, by Western Electric. All Rights Reserved.

Portions Copyright © 1984 VenturCom, Inc. All Rights Reserved.


Except as may be stated in the SPD for this product, no responsibility is assumed by Digital or its affiliated companies for use or reliability of this software, or for errors in this manual or in the software. Additional support and/or warranty services may be available from the developer of this software product. Digital has no connection with, and assumes no responsibility or liabilities in connection with these services.

This manual is subject to change without notice and does not constitute a commitment by Digital.

VENIX is a trademark of VenturCom, Inc.

UNIX is a trademark of AT&T Technology, Inc.

The following are trademarks of Digital Equipment Corporation:

DEC	DECwriter	Professional	VAX
DECmate	DIBOL	Rainbow	VMS
DECnet	MASSBUS	RSTS	VT
DECsystem-10	PDP	RSX	Work Processor
DECSYSTEM-20	P/OS	UNIBUS	
DECUS			

The PRO/VENIX† Documentation Set

The PRO/VENIX documentation set consists of the following manuals:

PRO/VENIX Installation and System Manager's Guide

The set up and maintenance of PRO/VENIX are described in the installation sections. Other articles explain the UNIX-to-UNIX‡ communications systems. The “System Maintenance Reference Manual” contains reference pages for devices and system maintenance procedures (sections (7) and (8)).

PRO/VENIX User Guide

The *User Guide* contains tutorials for newcomers to PRO/VENIX, covering basic use of the system, the editor **vi** and use of the command language interpreters.

PRO/VENIX Document Processing Guide

The line and screen editors and **nroff**-related text formatting utilities are described in the Document Processing Guide. Topics include: line editor **ed**, and stream editor **sed**; the text formatter **nroff**; the **nroff**-preprocessors **tbl** and **neqn**.

PRO/VENIX Programming Guide

The chapters in the *Programming Guide* explicate the different programming languages for VENIX.

† VENIX is a trademark of VenturCom, Inc.

‡ UNIX is a trademark of Bell Laboratories.

PRO/VENIX Support Tools Guide

This guide includes tools for programming, such as the compiler-writing languages Yacc and Lex, the M4 Macro processor, the program development utility Make, and the desk calculator programs DC and BC.

PRO/VENIX User Reference Manual

This is a complete and concise reference for the PRO/VENIX system. This volume contains write-ups on all PRO/VENIX commands.

PRO/VENIX Programmer Reference Manual

The reference pages in this volume include system calls, library functions, file formats, miscellaneous functions and games.

Contents

INTRODUCTION

Chapter 1. SETTING UP VENIX

Chapter 2. VENIX MAINTENANCE

Chapter 3. UUCP IMPLEMENTATION DESCRIPTION

Chapter 4. A DIAL-UP NETWORK OF UNIX SYSTEMS

Chapter 5. SYSTEM MAINTENANCE REFERENCE MANUAL

section (7) Devices

section (8) System Maintenance Procedures

Introduction

The *Installation and System Manager's Guide* covers the installation and maintenance of the PRO/VENIX operating system, and implementation of the VENIX communications system. The following paragraphs contain a brief description of each chapter.

The first chapter SETTING UP VENIX explains how to install the PRO/VENIX operating system on your hardware.

The chapter VENIX MAINTENANCE is a guide to VENIX administration, maintaining file systems, managing disk space, backing up user files and recovering and diagnosing system errors.

The chapter UUCP IMPLEMENTATION DESCRIPTION explains how to install and administer your UUCP system.

The chapter A DIAL-UP NETWORK OF UNIX SYSTEMS describes a network that provides information exchange between UNIX systems over the direct distance dialing network.

The SYSTEM MAINTENANCE REFERENCE MANUAL contains reference pages for devices and system maintenance procedures (sections (7) (8)).

A few words on notation: throughout this manual, frequent citations are made to pages in the *User Reference Manual* or *Programmer Reference Manual*. These are often in the form of a name followed by a section number, e.g. **fsck(1)**, referring to the **fsck** command description in section one of the *User Reference Manual*.

Commands to be typed in literally by the user are given in **bold**; generic arguments are in *italics*.

Contents

SETTING UP VENIX.....	1-1
1.1 HOW TO USE THIS DOCUMENT	1-1
1.2 CREATING THE SYSTEM IMAGE	1-1
1.3 DISTRIBUTION MEDIA.....	1-2
1.4 LOADING AND BOOTING VENIX	1-2
1.5 LOADING THE TRANSFER DISKETTE.....	1-2
1.6 PART I — ENTERING THE ACCESS NUMBER.....	1-3
1.7 PART II — USER AREA	1-4
1.8 PART III — SYSTEM AREA.....	1-7
1.9 PART IV — SETTING THE TIME ZONE	1-9
1.10 DIFFICULTIES IN BOOTING	1-11
1.11 ERROR MESSAGES.....	1-11
1.12 SOME INITIALIZATION EXERCISES	1-11
1.13 THE /etc/rc FILE	1-13
1.14 CONFIGURING YOUR PRINTER.....	1-16
1.15 CREATING NEW USERS	1-17
1.16 MULTI-USER SET UP	1-22
1.17 BACKING UP YOUR PRO/VENIX DISTRIBUTION	1-25
1.18 BACKING UP FILES AND FILE SYSTEMS	1-25
1.19 ABOUT DISK PARTITIONS	1-26
1.20 LIST OF PRE-CONFIGURED DEVICES	1-27
1.21 MISCELLANEA	1-27

Chapter 1

SETTING UP VENIX

1.1 HOW TO USE THIS DOCUMENT

This document contains step-by-step instructions for installing your PRO/VENIX system. If you don't have previous UNIX/VENIX experience, a look through **VENIX FOR BEGINNERS** in the *User Guide* will be helpful. The instructions for the installation will make more sense if you have an overview of the VENIX operating system.

A few words on notation: throughout this manual, frequent citations are made to pages in the *User Reference Manual*. These are often in the form of a name followed by a section number, e.g. **fsck**(1), referring to the **fsck** command description in section one of the *User Reference Manual*.

Examples of commands to be typed in by the user are always given in **bold**; responses from VENIX are in plain text. Commands are almost always in lower-case letters.

After you copy your system onto the hard disk, you may need to edit tables and files to customize the system to your particular installation. The **ed** editor is a line-oriented editor which will work on any terminal, and you may find this the easiest to use at first. You could also use the **ice**, **vi**, or optionally **fw** (The FinalWord) screen editors to make these changes. (See **AN INTRODUCTION TO DISPLAY EDITING WITH VI** in the *User Guide*). After your system is up and running, you will probably find the screen editors preferable for day-to-day use.

1.2 CREATING THE SYSTEM IMAGE

1.2.1 Minimum Hardware Required

VENIX will run on the Professional 350 without any special memory or hardware requirements.

SETTING UP VENIX

Floating point hardware is required for the Fortran 77 and Pascal compilers, and for the **lint** C verifier program. If the floating point chip is in place, the C compiler and other programs will use it automatically. Otherwise VENIX has a simulator that executes floating point code.

1.3 DISTRIBUTION MEDIA

PRO/VENIX is delivered on ten diskettes: one bootable transfer (XFER) floppy, five USER AREA floppies, and four SYSTEM AREA floppies.

1.4 LOADING AND BOOTING VENIX

If you have already been using your Professional with P/OS, be sure to use P/OS to backup all of your existing data on the hard disk before installing VENIX. The VENIX transfer process will write over any files and data that are currently in place. Save your backup files, following the instructions on pages 78–80 in the *Professional 300 Series User Guide for Hard Disk System* (AA-N603A-TH).

The VENIX loading process is controlled step-by-step via a control dialogue, as you will see. The entire process takes about 40 minutes. If you encounter problems at any point in the loading process and are unable to proceed any further, you can start over by rebooting (power off and power on).

The loading process consists of four parts: entering the access number (Part I) restoring the user area (Part II), restoring the system area (Part III) and setting the time zone (Part IV).

1.5 LOADING THE TRANSFER DISKETTE

To begin, insert the “Bootable XFER” floppy into floppy drive zero. (This is the top drive if your Professional 350 is flat on a table, or the left-hand drive if the unit is in a floor stand.) THE XFER DISKETTE MUST REMAIN IN THIS DRIVE FOR THE ENTIRE INSTALLATION PROCESS. Be sure that the orange arrows on the floppy drive and the floppy diskette are aligned properly. Turn the machine on to boot the transfer diskette.

The first word to appear on the screen should be the “Digital” logo. After this, the floppy light should turn on and the drive will click. In approximately 30 seconds, the XFER program will begin running. If this does not happen, you may have a hardware problem; see “Difficulties in Booting” later in this document.

The words “PRO/VENIX Rev 1.*n*” will materialize on the screen, followed by total memory and available memory in kilobytes.

The first question to appear on the screen is:

Do you wish to install PRO/VENIX (y or n)?

Type in “y”.

If you type in “n”, no, the shell will be executed, and you will be able to run VENIX commands. (This may be useful in the future if you are experienced with VENIX and wish to run maintenance programs not possible through XFER. To restart the transfer process, type CTRL-D to log out.)

1.6 PART I — ENTERING THE ACCESS NUMBER

After you have typed “y”, the screen will display:

Part I — Entering the Access Number

This program installs your PRO/VENIX system.

First you will need to obtain your access number.

The VENIX access number is a code which permits you to install and run VENIX. (Anyone attempting to make a bootleg copy without the VENIX access number will be unable to run VENIX.) This number is issued by your Digital Support Center, which will also provide you with a customer registration number to be used in future calls.

The next instruction on the screen is:

To receive your access number, contact:

Digital Support Center

1-800-DEC-8000

You will need to supply the following information:

Software ID # : SXXXX

Machine ID #: MXXXX

Your software ID and machine ID numbers will appear automatically on the screen. When you call the Digital Support Center, you will be asked for these numbers. You may also be asked for your computer serial number, which is stamped on the back of the unit; do not confuse this number with the Software and Machine ID numbers displayed on the screen.

SETTING UP VENIX

After providing this information, you will be given your VENIX access number. We strongly recommend that you write this number down; you may need it in the future if you have to reinstall PRO/VENIX.

Type in your access number, as indicated on the screen:

Please contact the Digital Support Center now, and when you have done so, enter your access number followed by a 'return'.
Number?

If you mistype the number, you will receive the message: "That's not the correct access number for this machine." You will be given an opportunity to retype it correctly.

You are now ready for Part II. If you are installing VENIX for the first time, you will normally answer "y" to all questions. If you have already installed VENIX, and are re-installing VENIX after a system crash, you will selectively answer "n" to questions concerning different partitions.

1.7 PART II — USER AREA

After you've typed "y", the next direction on the screen will be:

Part II — Restoring the User Area

Do you wish to restore the user area? (y or n)

Type "y". (If you type "n" for no, VENIX will skip directly to Part III, installing the system area.)

At this point, XFER checks to see if there are any VENIX files in the user area of the disk, left over from a previous installation. A warning will appear automatically. It will say "WARNING: There is a VENIX file system already on the user area. Do you wish to continue (y or n)?" If you type "n", VENIX will skip to Part III, leaving the user area intact. If you type "y", the user area will be erased, and the transfer process will continue.

The next message to appear on the screen is:

Formatting user area: please stand by ...

After approximately 30 to 60 seconds (depending on the size of your disk) you will see:

Formatting complete.

Do you wish to check for bad blocks on the user area (y or n)?

Type in “y”. (You should only type “n” if you have previously installed VENIX on this disk, and are sure that there are no bad blocks.)

A file system will now be constructed on the user area; bad block checking is done at the same time if it was requested. The command shown will be something like

```
/etc/mkfs -b /dev/w0.usr 14976
```

This will take about three to five minutes if bad block checking was requested, or about fifteen seconds if the checking was not chosen. (In the latter case, **-b** flag will not be given in the above command). The number 14976 shown is the number of blocks in the user area, and will vary depending on the size of your winchester.

If a bad block is found on the winchester, you will receive an error message of the form “Error on the winchester ...” followed by the message “Bad block number *n*”. VENIX can tolerate up to eight bad blocks in the file system it creates. In the unlikely event that more than eight bad blocks are found, the message “Too many bad blocks” will be given. If this message is given, or the message “Bad block in i-node region” appears, then the winchester will be unusable with VENIX. The only recourse if this happens is to get a new winchester and retry the installation procedure.

1.7.1 Loading the User Area

The user area holds some system commands and library files. In this next section, these files will be loaded onto the hard disk.

There are typically five USER AREA floppies provided. These are labelled 1 of 5, 2 of 5, etc.

The next direction on the screen will be:

Ready to transfer files from USER AREA diskettes.

DO NOT remove the XFER diskette from the upper or left-hand drive.

During this procedure, the transfer (XFER) diskette will remain in drive zero—the upper or left-hand drive. The USER AREA floppies will be inserted in drive one, the lower or right-hand drive. VENIX has a gentle reminder concerning the insertion procedure for floppies:

SETTING UP VENIX

When inserting diskettes, please remember to align the matching orange stripes on the diskette and drive.

This is followed by:

Insert USER AREA diskette #1 and press 'return' ...

The first USER AREA diskette is labelled 1 of 5. After you have placed the USER AREA diskette in the lower or right-hand drive, press the 'return' key. The **tar** command will then be run to transfer the user files from the diskette to the winchester. You will see:

```
tar xf /dev/rf1 /usr
```

Starting ...

The in-use light on the floppy drive will blink for about a minute, as the files are copied. After the loading has finished (the in-use light on the floppy drive has gone off), VENIX will print on the screen:

Insert USER AREA floppy #2 and press 'return'.

Remove USER AREA diskette #1 from the lower or right-hand drive, and insert diskette #2. Press CR. The in-use lights will go on, and you will see the **tar** command being executed again and the word "Starting ..." appear on the screen.

When diskette #2 is finished, you will be asked to insert diskette #3, and so on, for all five diskettes. In each instance you will be told when to take out the diskette and insert the next one in sequence.

When all the USER AREA diskettes have been transferred, Part II is completed. The message

User area completed.

is given. Withdraw the last USER AREA diskette from the lower or right-hand drive.

1.7.2 Recoverable Errors

If, during this loading procedure, you do any of the following things:

- Insert a diskette in the drive upside down;
- Press CR when no diskette is in the drive;
- Remove the XFER diskette from drive 0;

you will receive HARD or SOFT error messages.

1. The error message "HARD ERROR in reading this diskette" means that the diskette is inserted improperly, or has a defective surface.
2. The error message "SOFT ERROR in reading this diskette" means that the wrong diskette was inserted.

In both cases, after the error message, you will be asked:

```

Would you like to
  Abort the transfer
  Retry reading the diskette, or
  Ignore the error?
Type 'a', 'r', or 'i'.

```

Typing 'a' causes the transfer to abort. Power off and start again. If you have mistakenly removed the XFER diskette from drive 0, you should always make this choice.

Typing 'r' allows you to retry reading the user area diskette. You will be instructed to:

```

Check diskette and press 'return' when ready.

```

You can remove the diskette from the lower or right-hand drive and check to make sure it is the right one. If it is the wrong diskette, reinsert the right diskette, and press CR.

Typing 'i' causes the error to be ignored. The system will be installed, but one or more files may be missing or corrupted.

If you repeatedly get a HARD ERROR, the diskette may be bad. Contact your VENIX distributor.

XFER will let you insert the USER AREA diskettes in the wrong order, provided that you do not attempt to use the same diskette twice in one session. But since inserting the diskettes out of order may lead to minor permission problems on the winchester user area, it is strongly recommended that you stick to the order given by the diskette labels.

1.8 PART III — SYSTEM AREA

You have just completed loading the user area onto the winchester hard disk. In this section, Part III, the system area will be transferred to the hard disk. The procedure is nearly identical to that used for the user area.

SETTING UP VENIX

Now, the screen will say:

Part III — Restoring the System Area

Do you wish to restore the system area (y or n)?

Type ‘y’. (If you type ‘n’, VENIX will go to the end of the loading process. This should only be done if you have previously installed a system area on the disk).

If a VENIX file system was found on the system area — left over from a previous installation — a warning will appear (“WARNING: There is a VENIX file system already on the system area. Do you wish to continue (y or n)?”). If you type ‘n’, VENIX will preserve the files; if you type ‘y’, the files in the system area will be obliterated.

The warning mentioned above will not occur if you are loading VENIX for the first time. Instead, the following words will appear on the screen:

Formatting system, temp, swap, and boot areas: please stand by ...

Formatting is done here for the system area itself, as well as several miscellaneous areas of the winchester. After a short interval you will see:

Do you wish to check for bad blocks on the system area (y or n)?

Type ‘y’. If you have already installed VENIX and verified that the system area is free of bad blocks, you can type ‘n’ to bypass the bad block check and save some time.

After you have typed ‘y’, take it easy for a few minutes while VENIX makes a file system and checks for bad blocks on the hard disk. The command executed will be:

```
/etc/mkfs -b /dev/w0.sys 2944
```

If bad block checking was not requested, the **-b** is left out of the command shown above. The same rules apply to bad block checking on the system area as given before for the user area: up to eight bad blocks can be handled successfully, but if more than eight bad blocks are found, or a bad block is listed in the “i-node region,” then the winchester will have to be replaced for successful installation.

1.8.1 Restoring the System Area

The next step is loading the primary system area (“root” area) onto the hard disk. VENIX first tells you that the system will now be transferred onto the winchester, and then prompts you to insert the first system area diskette.

Ready to transfer files from SYSTEM AREA diskettes.

DO NOT remove the XFER diskette from the upper or left-hand drive.

Insert SYSTEM AREA diskette A and press ‘return’.

Insert SYSTEM AREA diskette A in the lower or right-hand drive. (There are four SYSTEM AREA diskettes, labelled A,B,C,D.) When ready, press CR. The **tar** command will then be run to transfer the system area files from diskette to winchester:

```
tar xf /dev/rf1 .
```

Starting ...

When the loading is finished, you will be instructed to:

Insert SYSTEM AREA floppy B and press ‘return’ ...

Remove SYSTEM diskette A from the lower or right-hand drive, and insert diskette B. Press CR. The in-use lights will go on, and you will see **tar** being executed and the word “Starting ...” appear again.

When diskette B is finished, you will be asked to insert diskette C, and so on, for all four diskettes. In each instance, you will be told when to take out the diskette and insert the next one in sequence.

If you get a HARD or SOFT ERROR message during the loading procedure, see section “Recoverable Errors.”

When all the SYSTEM diskettes have been transferred, you are finished with the transfer of the system area. After a pause while the system area is checked, XFER will continue on to part IV.

1.9 PART IV — SETTING THE TIME ZONE

Part IV is setting the time zone. VENIX will readjust the clock to reflect your local time zone, standard or daylight saving time.

On the screen you will see the menu:

SETTING UP VENIX

Part IV — Time Zone Selection

1. Eastern time zone (EST)
2. Pacific time zone (PST)
3. Mountain time zone (MST)
4. Central time zone (CST)
5. Atlantic time zone (AST)
6. Japan time zone (JST)
7. Greenwich mean time zone (GMT)
8. Other

Enter a number to select your time zone (1 – 8)?

Select a number, 1–8, to indicate your time zone. After you have typed in the number, your choice will be displayed on the screen and you will be asked to confirm it:

You have selected the xxx zone. Are you sure?

Type “y” if the correct time zone is indicated on the screen. If you entered the wrong number, type “n” and the menu will come up again.

If you selected “other” (8), further information will be requested to calculate your time zone:

Enter minutes west of GMT (minus for east)?

The time will be calculated and set for your local time zone.

The next question to appear on the screen is:

Does daylight saving time apply here (y or n)?

Answer yes (“y”) if daylight saving time ever applies in your time zone (regardless of whether it happens to be in effect at the time you are installing VENIX). In this case, changes in daylight saving time will be taken into consideration. The system date and file access times will always accurately indicate the correct time for your time zone, standard and daylight saving time.

After Part IV is completed, the last instruction will appear:

Installation complete. Remove diskettes from drives
and turn power off and on to reboot the hard disk.

Take the XFER diskette out of the upper drive, and the last SYSTEM AREA diskette out of the lower drive. After you reboot the system, you’re ready to fly with VENIX. If the installation has proceeded without problem, you may skip to section 5.

1.10 DIFFICULTIES IN BOOTING

1.10.1 General

There are several general types of problems which will prevent you from transferring the floppies or booting your winchester.

If no prompt appears, or the screen is dark, this may be due to a problem in the hardware. The Professional 350 contains a built-in diagnostic program which checks for many hardware problems every time the unit is powered on. If it locates such a problem, it may draw a picture of the unit with the failing board highlighted. Alternatively, it may leave one or more red lamps lit on the back of the unit. See page 5 – 33 in the *Professional 300 Series Technical Manual* (EK-PC350-TM-001) for meaning of these lamps.

1.11 ERROR MESSAGES

An error message in the form “Error ...” indicates an error in reading or writing to either the floppy itself or the hard disk. This could be due to incorrectly configured hardware, a misaligned floppy drive, or write protection on the XFER floppy (it must be write-enabled).

A message beginning with “PANIC ...” indicates an unrecoverable error, and will be followed by silence (i.e. the screen is blank, return key is inoperative, etc.). (See **VENIX MAINTENANCE** in this manual for a list of “PANIC” messages.) While PANIC messages are unlikely to occur during the loading process, they generally indicate that the hardware is bad. If problems persist after several attempts to reload, consult your distributor for further advice.

1.12 SOME INITIALIZATION EXERCISES

1.12.1 Booting

At this point, it is assumed that you have successfully installed a complete VENIX system on your hard disk. Remove the floppy diskettes from both drives, and reboot by turning the power on and off. After a few moments, VENIX will print on the screen:

PRO/VENIX Rev 1.*n*

(*n* is the current release level) as well as total and available memory size, in kilobytes.

SETTING UP VENIX

1.12.2 Memory Size

VENIX automatically determines the amount of memory present in the system. The “total” memory is the amount of memory actually present in your Professional 350 (typically 512 kb). The “available” memory is the portion of this total which is available to user programs. This is equal to the total memory minus the size of the VENIX kernel (which is, including buffers, approximately 48 kb).

You may wish to check the total memory size given by VENIX, to be sure that it jibes with what you expect to be installed. If it doesn't, you likely have a hardware configuration error or perhaps a bad memory board.

1.12.3 Aborting the Boot Sequence

The next message to appear is:

Checking file systems on system and user areas ...

Type CTRL-C to abort checking.

If a CTRL-C is typed immediately after this message, VENIX will stop its automatic boot-up procedure. As we'll explain in the paragraphs below, this only need be done under unusual circumstances. If you have just installed VENIX, skip on to the next section.

When the automatic boot-up is aborted with CTRL-C, VENIX logs you in as the super user “root” without prompting you for a password. This deliberate security breach is provided in VENIX as a fallback, in the event that the super user password is forgotten or the password file becomes corrupted. This is the only opportunity in VENIX for a user to assume “super user” status without first providing a password.

A CTRL-C might also be typed at this point if the “user” portion of the disk is damaged, and the user wishes to examine and repair it before it is automatically mounted. In any case, after the user has finished correcting the password, or fixing the disk, a CTRL-D can be used to log out. At this point, VENIX resumes its automatic boot-up procedure and executes the commands from `/etc/rc` (see below), skipping the automatic `fsck`.

1.12.4 Automatic fsck

A consistency check (`fsck`) of the file system is now executed automatically, to ensure that the file system structure is valid on the system and user areas. If the

fsck doesn't appear at this point, your transfer may have been done incorrectly and you will have to reload the floppies onto the winchester.

The **fsck** output will be displayed on the screen, and normally looks something like this:

```

/dev/rw0.sys (System Area)
Phase 1 - Check Blocks
Phase 2 - Not needed
Phase 3 - Check Pathnames
Phase 4 - Check Reference Counts
Phase 5 - Check Free List
    170 files 2752 blocks 110 free
/dev/rw0.usr (User Area)
Phase 1 - Check Blocks
Phase 2 - Not needed
Phase 3 - Check Pathnames
Phase 4 - Check Reference Counts
Phase 5 - Check Free List
    204 files 3020 blocks 14435 free

```

The statistics provided for each file system (number of files, number of disk blocks used, number of disk blocks free) will probably be somewhat different from those shown above. In day-to-day use, as more files are created by users, these numbers will of course change.

If a CTRL-C is typed after **fsck** has started, the check will stop and VENIX will skip on to the next step, reading the initialization file **/etc/rc**. For further information about **fsck** see the appropriate section in the **VENIX MAINTENANCE** chapter.

1.13 THE **/etc/rc** FILE

After the **fsck** has been done, VENIX reads a series of commands from the file **/etc/rc**. The message

```

System entering user login mode.
Reading command file /etc/rc ...

```

will appear. Messages will be printed giving the current date, and the baud rate assigned to the printer port. (Both of these items may be changed later by the super user; they are set and reported now just for convenience.)

The file **/etc/rc**, incidentally, is simply a series of standard VENIX commands. It is available for examination and editing by the super user. This might be done,

SETTING UP VENIX

for example, to change the baud rate used with your printer; see “Configuring Your Printer.”

After all the commands in `/etc/rc` have been read, VENIX is ready for use.

1.13.1 Login

At this time, a most welcome word will materialize on the screen:

```
login:
```

VENIX is waiting for you to enter a user login name. We’ve installed a few login names which you might wish to use. Next to the login prompt, type:

```
demo
```

for a little demonstration which we’ve prepared. Left alone, the demonstration will repeat itself forever. When you’ve had enough, hit CTRL-C to return to “login”.

Next, try logging in as **guest**. If you list (with the `ls` command) the files in the **guest** directory, you will see one—a small C program—called **hello.c**. If compiled with the command

```
cc hello.c
```

and then executed by typing

```
a.out
```

it will print a little greeting.

1.13.2 Logging Off

To log off the system, type CTRL-D. VENIX will respond with another “login:” prompt, and wait for a new log in.

1.13.3 Shutting Down

Your Professional 350 should not be powered off while programs are running. If this happens, file system damage may occur. (Any such damage will be reported by **fsck** when the computer is next turned on.) However, if none of your own programs are running, and no other users are on the system, you can always turn the computer off safely. If you have been executing programs in the background, and are unsure whether they are still running, the **ps(1)** command can be used to ascertain their status.

1.13.4 Super User

The super user has powers far beyond those of mortal men and women ... namely, the ability to access any and all files, as well as to execute many privileged commands associated with system growth and maintenance.

To login as the super user, type:

root

When the password prompt appears on the screen, type:

gnomes

When you type **gnomes** on the keyboard, it will not be visible on the screen. The super user password can be changed from “**gnomes**” to something else if a more secure system is desired (see the section “Editing the Password File.”) After you have typed in the password, you will see the words:

Welcome to VENIX

Beware: you are the super user!

And underneath, a prompt “SUPER>” will appear. The “Beware ...” message is simply a warning that you are the super user. The “SUPER>” prompt is a repeated reminder of your status.

1.13.5 Color Monitor

If you have a color monitor (VR241) on your Professional console, you will find that your screen intensity is somewhat low. This is because PRO/VENIX normally uses a lower intensity, appropriate for monochrome monitors. To raise the intensity, use the **setscreen(1)** command. Type

setscreen -c

This higher intensity will overdrive a monochrome monitor, and lead to screen distortion; it is strongly recommended that it only be used with color monitors.

To lower the intensity to the default setting, as appropriate for monochrome monitors, type:

setscreen -m

The **setscreen** command “remembers” in battery backed-up memory the intensity you selected. The command

setscreen

with no flag simply sets the screen intensity to the one last selected. This is done

SETTING UP VENIX

for you at boot-up in the command file `/etc/rc`, so once you have selected the higher intensity you need not type the command ever again (unless of course you switch to a monochrome monitor).

1.13.6 Setting the Date

The super user can set the current date, using the **date** command, which is:

```
date yymmddhhmm
```

yy is the current year (e.g. 84), *mm* is the month (01 to 12), *dd* is the day (01 to 31), *hh* is the hour (00 to 24), and *mm* is the minute (00 to 60). For example:

```
date 8406100946
```

sets the date to 9:46 in the morning, June 10, 1984.

If the date is not set, then the system uses the date and time it last remembers.

If the year, month or day is omitted, the last remembered value is taken by default. Now if you realize that you were off by two minutes, simply type:

```
date 0948
```

to set the date to 9:48 of the same day.

The Professional 350 has a battery backed-up perpetual clock. You only need to set the date once. After that, the date should be correct, even if you power the machine off and on. When the system is booted, the command **date -l** is automatically run to load the date from the perpetual clock. If you change the date, that date is remembered in the perpetual clock.

1.14 CONFIGURING YOUR PRINTER

To hook up a serial printer, plug it into the printer port. To test out your printer, type:

```
date > /dev/lp
```

The date should be printed out. If nothing happens, check that the printer is securely connected.

If your printer still doesn't respond, the baud rate may need to be changed also. The default baud rate (as set in boot-up file `/etc/rc`) is 4800. If your printer is set for a different baud rate, you can either reset the switches on your printer for 4800, or use the **stty** command. **stty** resets the baud rate option for the com1 port. To use **stty**, type:

stty (baud rate for your printer) > /dev/lp

For example, if the baud rate for your printer is 1200, you would type:

stty 1200 > /dev/lp

To permanently reset the baud rate, you will also need to modify the **stty** command in the **/etc/rc** file. To edit **/etc/rc** you must login as the super user. **/etc/rc** should be edited to contain the same **stty** command used above.

1.15 CREATING NEW USERS

If there is only one person using your Professional, you could continue using the system under the “guest” login account. But if there are several people making use of the machine, and particularly if users are logging in via modem, then separate accounts should be created.

There are three steps to creating new user accounts under PRO/VENIX: editing the password file, creating the home directory, and creating the user’s login initialization file, **.profile** or **.login**.

1.15.1 Editing the Password File

The system coordinator (as super user) can install new users by editing the password file **/etc/passwd**.

Each line in the file corresponds to a separate user, and contains an entry of the form:

name:password:UID:GID:empty:directory:shell

where

- name* is the user’s name (lower-case letters only).
- password* is the user’s encrypted password. After the new user logs in, he will put in his own password, using the **passwd** command. Do not enter text into this field yourself. Since this field is encrypted, when set it appears as nonsense letters.
- UID* is the user’s numerical ID number, which should be unique among all user ID’s. Numbers 1 – 128 may be used; user ID 0 is reserved for the super user “root”.
- GID* is the user’s numerical group ID number, indicating the group he or she belongs to. Until you set up different user groups, you can use group ID number 10 for everyone. If you wish to

SETTING UP VENIX

divide your users into different groups, then the file `/etc/group` can be edited to create new groups, and users can be given group ID numbers assigning them to one or another group. Numbers 1–128 may be used; group ID 0 is reserved for the super group (see the **VENIX MAINTENANCE** chapter for details).

empty is an unused field.

directory is the user's "home" directory, where he is placed when first logging in.

shell is the user's shell interpreter, which is run when he logs in. If this field is left blank, the standard "Bourne" shell is used (see **sh(1)**). Users who wish to use the C-shell **cs(1)** should put the pathname `/bin/csh` in this field; note that this field is *not* terminated with a colon. (While the Bourne and C-shells are in many respects identical, the latter offers several unique features such as a "history" mechanism for easy reissuing of past commands. See the two shell tutorials in the *User Guide* for details.)

The following commands could be used to enter a new user "holmes" in the password file. The line editor **ed(1)** is used to edit the password file:

```
SUPER> ed /etc/passwd
188
$a
holmes::10:10::/usr/holmes:
.
w
217
q
```

This appends an entry for "holmes" to the end of the password file, with user ID 10, group ID 10, and login directory `/usr/holmes`.

1.15.2 Making Home Directories for New Users

The home directory given for the newly installed user must now be made, and the owner of the directory be changed to that user. Login as each user to make sure the password file is correctly edited. For example, for user "holmes" these commands would be given:

```

SUPER> mkdir /usr/holmes
SUPER> chown holmes /usr/holmes
SUPER> chgrp other /usr/holmes
SUPER> login holmes
$ l -a
total 2
drwxrwxr-x 2 holmes      64 Sep 23 12:07 .
drwxrwxrwx 7 root       208 Sep 23 12:04 ..
$ login root
Password: gnomes
SUPER>

```

In the example above, **/usr** refers to the top directory on a file system in the user area; the command **mkdir /usr/holmes** makes a directory called **holmes** there. The command **chown holmes /usr/holmes** changes the ownership of directory **/usr/holmes** to **holmes**, and **chgrp other /usr/holmes** changes the group ownership to “other” (otherwise the group owner would be the root’s group, “system”).

The new user entry is tested by logging in under the new name (“login holmes”) and listing all the files (**l -a**) (that’s the letter “l” not the number one). The only entries in this new directory should be “.” and “..”, referring to the current directory (**/usr/holmes**) and its parent directory (**/usr**). Finally, a “login root” and “gnomes” will get the system coordinator back to super user status.

The following problems may arise when the new user tries to login:

1. The user receives the message “No directory”. This means that the shell could not find the login directory, as specified in the password file. Either that directory is not present, or the wrong number of fields are in the password file, causing the system to be confused about what field holds the directory name.
2. The user receives a “Password:” prompt, although the user has not designated a password. The user may have typed his or her name incorrectly and should try logging in again. Alternatively, the password file entry for that user might be jumbled.
3. The user can not access his or her files. Either the ownership of the files was not correctly preset for the user, or the password file entry for that user has bad user or group ID numbers.

In any of these cases, refer back to the **passwd** command file to correct any mistakes in the file entry.

SETTING UP VENIX

1.15.3 Setting Up `.profile` and `.login` Files

Now that the user account and directory is set up, a login initialization file should be made, either by the system coordinator (as super user) or by the new user. This file will be called `.profile` if the user is running the standard Bourne shell, or `.login` if the user is running the C-shell. (The choice of shells was made when the `/etc/passwd` file was edited, as shown previously.)

These files contain commands which are executed automatically whenever the user logs in. They are used to assign a custom prompt for the user, and to set “environment” variables to indicate, for example, the type of terminal the user is on or the locations of personal command directories.

An initialization file is a convenience, not an absolute necessity. The commands it contains could always be typed by hand each time the user logs in. But this would be rather tiresome; we suggest you take advantage of the initialization file capability, and let the computer do the work for you!

We have left sample `.login` and `.profile` initialization files in the guest home directory, `/usr/guest`. You may copy one of these into the new user’s home directory. We’ll continue our example for new user `holmes`, assuming that he is running the standard Bourne shell (`sh(1)`), and will therefore require a `.profile` file rather than a `.login`. Corresponding details for the C-shell are given afterwards.

```
cp /usr/guest/.profile /usr/holmes
cd /usr/holmes
ed .profile
```

We are now in the `ed(1)` editor, ready to modify the file. (You may wish to use the screen editor `vi` or `ice`). The first thing to do is change the user prompt from “GUEST:” to “holmes:”. The line in question is

```
PS1 = "GUEST: "
```

To move to that line, we type

```
/GUEST/
```

and to change, and verify, the new prompt:

```
s/GUEST/holmes/p
```

The new line should now be printed.

A very important shell variable is `TERM`, which is used to specify the type of terminal being used. The Professional 350 console is known as `pro`, so the line initializing `TERM` should read

```
TERM = pro
```

This variable is examined by full screen editors such as **vi**, and programs such as **more(1)**. If a VT100 terminal were hooked to the Professional (see the section “Connecting a Second Terminal”), we might use instead

```
TERM = vt100
```

The complete list of terminals known to **vi** is given in **termcap(5)**. The section “Setting Up Screen Editors” also shows a mechanism for automatically determining the terminal the user might be on.

Another shell variable we might wish to modify is **PATH**, which contains a list of directories, separated by colons, in which the shell looks to find commands. The standard locations to look in are the current directory (**.**) and the VENIX system “binary” directories **/bin** and **/usr/bin**. (If you list the contents of these last two, you may see some familiar names.) Users frequently have their own private command directory called **bin** under their home directory. To add this name to the standard **PATH** for **holmes**, insert a new line at the beginning of the file:

```
$a
PATH = $PATH:$HOME/bin
```

The **PATH** variable is set to its original value plus the new directory **\$HOME/bin**. **\$HOME** is a shell variable equivalent to the user’s login (home) directory, in this case **/usr/holmes**. The expression **\$HOME/bin** then corresponds to the directory name **/usr/holmes/bin**. (We haven’t actually made such a directory yet, but when we do, the shell will search it to find any command name typed by the user.)

The **EXINIT** variable is useful for users of the **vi** and **ex** editors. It contains initialization commands which are read by these editors whenever they are invoked, and can be used to customize function keys and preset any of the numerous editor parameters. See the section “Setting Up Screen Editors” for details.

If **holmes** were running the C-shell, the sample file **.login** should be copied over from **/usr/guest** to **/usr/holmes**. The following lines give the equivalent C-shell versions of the variables initialized above:

```
set prompt = "holmes: "
setenv TERM pro
set path = (. /bin /usr/bin ~/bin)
```

The syntax used is somewhat different than that of the **.profile** file. The **setenv** (“set environment”) is used for variables like **TERM** which are to be passed to programs the user runs, not just used by the C-shell itself.

SETTING UP VENIX

The sample **.login** file also contains the entry

```
set history = 20
```

This causes the C-shell to remember the last twenty commands typed by the user, so that it can recall and re-issue them if the user requests. This is a terrifically useful feature; see the C-shell documentation in the *User Guide* for more details.

Other commands can be inserted within **.profile** or **.login** files to print login messages, for example

```
echo "Glad to have you aboard, sir."
```

or perform routine login functions. A favorite is

```
/usr/games/fortune
```

which will print a few words of wisdom on each login.

1.15.4 New Passwords

If system security is desired, then you should give yourself a password. Type in:

```
passwd
```

On the screen it will say, "Changing password ..." and then VENIX will prompt:

```
New password:
```

Type in a password which has at least four characters. You will be prompted to type the password a second time. VENIX prompts:

```
Retype new password:
```

The password will not be visible on the screen (to keep the password confidential), but it has been entered. The next time you log in, when the password prompt appears, type in the password.

1.16 MULTI-USER SET UP

1.16.1 Connecting a Second Terminal

The Professional 350 communications port provides a standard RS232 serial interface, so that it may be hooked to a variety of different devices, such as modems (like the DF03), printers (like the LA50), or terminals (like the VT100). If you wish to have a second user log in to PRO/VENIX via the comm port (either by terminal or modem), you will have to edit the **/etc/ttys** file so that PRO/VENIX will send a "login:" message to that port. It is *not* necessary to modify this file if you have a modem which will be exclusively used for out-going calls.

Login as super user on your console. Type:

SUPER> ed /etc/ttys

a number will appear on the screen, giving the number of characters in the file.

At this point you must decide whether you wish to hook your Professional to a modem or a dumb terminal, and select the appropriate comm port option: either "com1.modem" or "com1." The first option, "com1.modem," is used for modems, and causes VENIX to obey "modem control" conventions; that is, it will sense when the modem has answered an incoming call (carrier detect), and when it has hung up. Plain "com1" is used with terminals, and causes VENIX to ignore all modem conventions.

The file /etc/ttys contains listings, one per line, for all possible active terminals. The first digit of each line is '0' if the line is unused, and '1' if a "login:" should appear on it. The second digit indicates the baud rate.

If you wish to use VENIX with a dumb terminal on the comm line, ("com1" option), type:

```
/com1/
```

On the screen you will see:

```
0lcom1
```

To make the terminal active, the 0 will be reset to 1. Type the substitute command:

```
s/01/11/
```

The second digit on the line indicates the baud rate: a '1' as shown means 9600 baud. See ttys(5) in the *Programmer Reference Manual* for details on other choices. If your terminal is at a different baud rate, you can change the second digit at the same time.

If, on the other hand, you wish to use VENIX with a modem on the comm port ("com1.modem"), type:

```
/com1.modem/
```

to select the right line, and then

```
s/03/13/
```

to enable it. The '3' is used to specify that either a 300 or 1200 baud modem is used (when a user tries to log in, VENIX monitors the line and automatically switches from one speed to another if "break" characters are found, indicating a mismatched rate.) Other speed options are available; see ttys(4).

After editing the "com1" or "com1.modem" line in the file, type

```
w
```

```
q
```

SETTING UP VENIX

to write out and finish editing.

To check that the file is correct, type:

```
cat /etc/ttys
```

If everything is O.K., the file should look like either

```
12console
11com1
03com1.modem
```

or

```
12console
01com1
13com1.modem
```

depending on whether you chose the modem or dumb terminal input.

VENIX will not be aware of your changes in `/etc/ttys` until you reboot the computer, or type the command

```
kill -2 1
```

This forces the `init` process to examine the newly edited `/etc/rc` file, and issue new logins appropriately. After the reboot or the `kill` command, VENIX should issue a “login:” to the comm line.

At this point, if a terminal is hooked to the comm port, a “login” should appear on the screen. If nothing happens, make sure that the terminal is turned on, and that the cable is securely connected between the two machines. Check the terminal baud rate; if it does not match the rate specified when you edited `/etc/ttys` (by default, 9600), then you must either change the terminal baud rate, or re-edit `/etc/ttys` and go through the above procedure again. Another possible cause for problems is if you accidentally selected “com1.modem” instead of “com1” in `/etc/ttys`.

If a modem is hooked to the comm port, it should find a “Data Terminal Ready” signal (DTR) from the Professional; this will be indicated on the appropriate modem lamp, if any. When called, the modem should now answer the phone with a tone. If the DTR lamp is not on, or the modem does not answer the phone, check your cable. (Note that for modem control to work, connector pins 2,3,4,5,6,8 and 20 must all be wired through.)

After setting the terminal options, reboot and you will see the login.

1.16.2 Shutting Down the System

If you have a multi-user system, you can power off or reboot as long as the system is quiet and there are no other users on the system. (Use the **who** command to ascertain who is logged in.) If there are other users, type CTRL-D on their terminals to log them off. Alternatively, the super user can run the **ps** command to find out the process identities of users still on the system, and then use the **kill** command to kill the processes.

1.17 BACKING UP YOUR PRO/VENIX DISTRIBUTION

We strongly recommend that you duplicate the floppy diskettes on which your PRO/VENIX system came. (A back-up copy can be a life-saver in the event that your system fails and you need to reload it from scratch.) A useful command for this purpose is **dd(1)**, which stands for “device-to-device” copy. For each of the distributed diskettes, follow this procedure: place the original diskette in drive 0 (top or left-hand drive); place a blank diskette in drive 1 (bottom or right-hand drive); and type

```
dd < /dev/f0 > /dev/f1 bs = 10b count = 79
```

The “**bs=10b**” tells **dd** to copy 10 512-byte blocks at a time; the “**count=79**” causes 79 transfers to be made. As a result, a full 790 blocks are copied—the length of a floppy diskette.

This command can be used to back-up *any* of your diskettes, not just those that VENIX came on. **dd** simply makes an image copy; it does not depend on any particular disk format. If you find that you use this command frequently, you might wish to place the command in a shell script.

1.18 BACKING UP FILES AND FILE SYSTEMS

The importance of “backing up” files cannot be over-emphasized! “Backing up” is the process of making copies of files on a medium separate from that which holds your primary version of the files. Any computer system will crash from time to time, due to power failures, hardware problems, software bugs, or maybe even the phase of the moon. Most often this will happen without the loss of any significant data. However, there will be occurrences of loss with varying degrees of significance. Backing up is also good protection for those instances when users get sloppy and accidentally remove files, especially with wild card filename specifiers.

SETTING UP VENIX

1.18.1 Backing Up System Files

Once your system is fully customized for your hardware, you should back it up once so that you can easily restore it should disaster strike. You can use the **tar** command to backup the system. See “Backing Up the System” in the **VENIX MAINTENANCE** chapter.

1.18.2 Backing Up User Files

One way to backup user files is to create a file system on a floppy diskette, **mount** it under an empty directory (**/f0**) and then copy files over to it. (See “Formatting Diskettes” in the **VENIX MAINTENANCE** chapter.) However, you may find it time consuming to backup many files and directories this way, so this technique is not recommended for backing up large quantities. We recommend using the **tar** command for user (as well as system) backup.

1.19 ABOUT DISK PARTITIONS

1.19.1 Disk Partitions

Most hard disk units are divided into two or more partitions. This is done both for convenience and protection. Should one partition be damaged, the other area is still left intact.

There are three basic uses for the disk partitions:

1. The *system* partition (known as **/dev/w0.sys**) holds all the **VENIX** commands, libraries, devices, and so on, as well as space for swapping processes (the swap area).
2. The *temporary* partition (known as **/dev/w0.tmp**) is used by compilers, editors, and other programs for temporary data storage. The file system on this partition is cleared and then **mounted** under directory **/tmp** every time the system is booted.
3. The *user* partition (known as **/dev/w0 usr**) covers all the disk area unused by the system and temporary partitions. The user partition holds the remaining system commands and libraries, and is available for user programs and files.

1.19.2 List of File Systems in **/etc/checklist**

/etc/checklist contains a list of file systems which are checked when **fsck** is run automatically. If you list the contents of **/etc/checklist**, you see two entries:

`/dev/rw0.sys`:System Area

`/dev/rw0.usr`:User Area

`/dev/rw0.sys` (or `/dev/rw0.usr`) specifies the partition, followed by a colon and the labels “System Area” or “User Area.” The device names given here have an extra “r” in front of them to designate the “raw” version, which can be **fucked** slightly faster than the ordinary “block” version.

1.20 LIST OF PRE-CONFIGURED DEVICES

VENIX has been pre-configured for a number of devices which include:

Description	Device Names
one console, “console”	<code>/dev/console</code>
one winchester, “w0”	<code>/dev/w0.sys</code> , <code>/dev/w0.usr</code> , <code>/dev/w0.tmp</code>
one communications port, “com1”	<code>/dev/com1</code> , <code>/dev/com1.modem</code> , etc.
one printer port, “lp”	<code>/dev/lp</code>
two floppy drives, “f0 and f1”	<code>/dev/f0</code> , <code>/dev/f1</code>

The extended bitmap option board is used automatically if installed.

You can list the device directory with the command:

```
l /dev
```

Notice that most of the disk partitions have two entries, the “block” versions called “w0.sys”, “w0.usr”, ... and the “raw” versions called “rw0.sys”, “rw0.usr.” These raw versions refer to the identical disk areas as their block counterparts; however, raw versions provide unbuffered and more direct access to the disk, which in some cases is more efficient to use.

1.21 MISCELLANEA

1.21.1 Setting Up Screen Editors

The simple screen editor **ice** will work only on the Pro console or a VT52 or VT100 compatible terminal. No special setup work is required to use it; see **ice(1)** for usage details.

The screen editor **vi**, on the other hand can work on a variety of different terminals. For **vi**, look in the file `/etc/termcap` for a listing of the terminals which are supported. If your video terminal is not listed, substantial modification may be necessary. Consult your VENIX distributor. The file `/etc/termcap` should be edited to remove entries for dumb terminals not used with your Professional. See **termcap(5)** for details on the file format.

SETTING UP VENIX

The section “Setting Up .profile and .login Files” discussed the setting of the **TERM** variable in the user’s shell, used to tell **vi** and some other screen editors what type of terminal the user is on. The example in that section showed the **TERM** variable set to a single particular name. However, if users are logging in on both the Pro console and a dumb terminal on the comm port, it is convenient to have the shell set **TERM** to one of two names, depending on which terminal the user logged in on. This can be done with some simple shell programming in the **.profile** or **.login** files. The trick is to have the shell first run the **tty(1)** command to determine which line you are on, and then set **TERM** with the appropriate name. To accomplish this, the following statements may be placed in the **.profile** for users of the Bourne shell:

```
case `tty` in
    /dev/console)  TERM = pro ;;
    /dev/com1)    TERM = vt100 ;;
esac
export TERM
```

The corresponding entry in **.login** for users of the C-shell is:

```
switch ( `tty` )
    case /dev/console:
        setenv TERM pro
        breaksw
    case /dev/com1:
        setenv TERM vt100
        breaksw
endsw
```

Note the grave accents around the **tty** command. See the Bourne and C-shell documentation for details on these constructions.

The **EXINIT** variable is used to preset editing parameters for **vi** and **ex**. The sample copies of **.profile** and **.login** provided in directory **/usr/guest** set this variable to map a number of the Pro function keys to commonly used editor functions. We suggest you copy the setting provided there, in which case the following function key assignment will work when running **vi**.

The **Find** key does a forward search; the **Prev Screen** key scrolls the screen up; the **Next Screen** key scrolls the screen down. The **Select** key sets marker **z** at the current cursor position, and the **Remove** key deletes from the cursor position to the point where that marker was last set. The **Insert Here** key restores the text deleted by the **Remove** key. (Thus the **Select**, **Remove**, and **Insert Here** keys can be used to mark and move a section of text.) Finally, the **Exit** key causes the edit buffer to be written out if necessary and the editor exited (this is the “ZZ” command). See the **vi** documentation for details on how the **EXINIT** variable may be set.

The function keys on the Professional keyboard send a sequence consisting of an ASCII “escape” followed by a single letter. The user should NOT press function keys other than the seven listed above and the four “arrow” keys, as the single letters sent by other function keys will be interpreted as a **vi** command, often causing undesirable results.

1.21.2 Setting Up **nroff**

nroff(1) formats text for a number of different printers. For a list of printers supported by **nroff**, look in the *User Reference Manual* under **nroff(1)**, the **-T** option. The flag **-Tpro** will produce output formatted for the Professional 350 console screen; the flag **-Tla100** will produce output for a DEC LA-100 printer. (These special options are used in order to generate output that contains the appropriate codes to produce bold print and perform other functions which vary from device to device.)

If your printer is not among the list of already supported devices, you will have to write a new terminal descriptor table. See the **NROFF TERMINAL DESCRIPTOR TABLE FORMAT** chapter in the *Document Processing Guide* for instructions.

1.21.3 Operating Without Floating Point Hardware

If your processor does not have the floating point option, VENIX includes a software emulator, which runs slower than the hardware. If you want faster floating point calculations, install a floating point chip in your Professional 350.

Contents

VENIX MAINTENANCE.....	2-1
2.1 INTRODUCTION	2-1
2.2 MAINTAINING FILE SYSTEMS.....	2-2
2.3 BACKING UP FILES	2-7
2.4 USEFUL COMMANDS AND FILES	2-10
2.5 SYSTEM ACCOUNTING	2-13
2.6 LOGIN TIME.....	2-14
2.7 SYSTEM FILES AND DIRECTORIES.....	2-15
2.8 SYSTEM ERRORS AND CRASHES.....	2-18

Chapter 2

VENIX MAINTENANCE

2.1 INTRODUCTION

This document is a guide to administering a VENIX system. Topics covered include: maintaining file systems and managing disk space; backing up user files; and recovering and diagnosing system errors. This guide should be read in conjunction with **SETTING UP VENIX** which contains useful information concerning the installation of a VENIX system.

Throughout this document, references are made to VENIX commands which can be useful for system maintenance. The descriptions of these commands are intended to suggest their most common uses, but no attempt is made to explain all command features and options. The *User Reference Manual* contains a complete write-up of all the VENIX commands. Sections in the *Programmer Reference Manual* describe system calls, library functions, and file formats. Certain system programs which are never run directly by users (such as the clock daemon **cron**) are documented in **SYSTEM MAINTENANCE PROCEDURES**, section 8 in the *Installation and System Managers's Guide*.

2.1.1 The Super User and Super Group

The system administrator should use the **root** login so that he has super user capabilities. The super user can access any file without regard to permission settings, and can signal, terminate or suspend/resume any process, no matter who owns it. The super user can also execute certain privileged commands, such as **chown** (change the ownership of a file). Since the super user is free from most restrictions, he must take great care not to accidentally erase files or damage the system.

Users may obtain limited super capabilities by being in the "super" group. Members of the super group have the ability to manipulate system files and directories (for example, to add or remove commands from the **/bin** directory). However, they have none of the special abilities the **root** has to access any user file regardless of permission setting, or to kill any process. (Super group members

VENIX MAINTENANCE

are also given permission to do certain things useful for real-time programming, such as run at very high priority.)

The user and group ID's are assigned for each user in the `/etc/passwd` file. **SETTING UP VENIX** describes how to edit this file to add new users. The super user has user ID of zero; the super group has group ID of zero. Normal users are given non-zero user ID's, and assigned to the default group called "other" or placed in different groups as desired. Group names and group ID's are found in the file `/etc/group`.

2.2 MAINTAINING FILE SYSTEMS

2.2.1 Creating New File Systems

A "file system" is a collection of files on a disk partition, internally structured so that VENIX can locate and manipulate individual files and directories. An empty file system must first be created on a disk partition before files can be used there. On a floppy diskette, a disk partition covers the entire physical unit. The hard disk, on the other hand, is divided into three partitions: the system, user and temporary areas.

On the hard disk, when VENIX is initially transferred onto the winchester, file systems are automatically made for the system, user and temporary partitions. When you wish to use a floppy diskette as an extension of your hard disk file system, that is, to **mount** it on your directory tree, you must first make a file system on the diskette. You do not need to make a file system on diskettes which are to be used for back-up or transfer purposes with the **tar** or **dump** commands.

2.2.2 Making a File System on Floppies

To make a file system on a diskette, place a diskette in the upper disk drive (f0). Be sure that the diskette is seated properly, and that the arrow on the diskette is aligned with that on the drive. Close the door and type:

```
/etc/mkfs /dev/f0 790
```

mkfs is the "make a file system" command. The disk drive is specified as `/dev/f0`. The length of the file system in blocks on the floppy diskette is 790.

mkfs erases all files (if any) on the diskette, and makes a file system. For this reason, you must be careful not to accidentally create a file system on a diskette containing valuable files.

If you wish to check for bad blocks on the diskette, use the **-b** flag, e.g.

```
/etc/mkfs -b /dev/f0 790
```

This will take a few minutes to run.

To make a file system on more diskettes, remove the diskette in drive 0, insert a new diskette, and re-type the **mkfs** command.

For convenience and compatibility with other versions of UNIX, a command called **format** is provided, which runs **/etc/mkfs** for you. Note that **format** on the Dec Professional only makes file systems; it can *not* be used to do track and sector formatting. The Professional 350 does not have the capability of reformatting other types of diskettes, so you must always use pre-formatted RX50 diskettes.

The command

```
format /dev/f0
```

makes a file system on the diskette and performs bad-block checking, and thus is exactly equivalent to

```
/etc/mkfs -b /dev/f0 790
```

To avoid the bad-block check, use the **-i** flag with **format**, as in:

```
format -i /dev/f0
```

This is equivalent to

```
/etc/mkfs /dev/f0 790
```

A few common problems using the **mkfs** command can be easily corrected:

1. If you type **mkfs** but omit the device name (**/dev/f0**) after it, on the screen it will say:

```
Usage: mkfs device length
```

Retype the command, with **/dev/f0** included.

2. If you forget to put in a diskette, and type the **mkfs** command, the following error message will appear on the screen:

```
Error on the floppy, unit 0.  
Diskette improperly inserted ...
```

When the error message stops, insert a diskette and try again.

VENIX MAINTENANCE

2.2.3 Mounting and Unmounting

The **mount** command attaches a file system to a directory on the hard disk. When VENIX is booted on the winchester, file systems on the system and user partitions are **mounted** and **umounted** (unmounted) automatically. Once users are logged onto the system, they can access any part of the system directly, without any special mounting procedures.

If file systems are located on a floppy diskette, they must be “manually” **mounted** and **umounted**. Files systems on floppy diskettes are logically attached to the directory hierarchy by mounting them on an empty directory on the hard disk. Once this is done, the previously empty directory now accesses the root of the new file system; any files on that system now appear in that directory.

A file system must be created on the diskette before it is **mounted**. If a diskette already contains a file system, then go ahead with the **mount** command. If you are using a blank diskette, make a file system first, using the **mkfs** command. Before **mounting**, run an **fsck** to ensure that the file system is O.K. The use of **fsck** is described later. For example, the commands:

```
fsck /dev/f0  
mount /dev/f0 /f0
```

will first check the file system on the floppy diskette (known as **/dev/f0**), and then **mount** that file system in a directory (**/f0**) in the system area of the hard disk.

Once the file system is mounted on the hard disk, you can use any normal VENIX commands. Any of the file systems on the diskette are now a part of the regular system and user areas on the hard disk. You will find that the copy commands will enable you to copy files from the winchester to the floppy diskette.

The command

```
umount /dev/f0
```

will unmount the system. It is crucial that the file system(s) on a disk are **umounted** before the disk is physically unloaded! If this is not done, when you attempt to mount the next file system, that file system will be corrupted.

If you remove a diskette without **umounting** it, you may have to reboot the system, which will run an **fsck** automatically to clean up the file system. When you attempt to re-mount the floppy diskette, first run an **fsck** before **mounting**.

Type:

fsck /dev/f0

then the **mount** command.

It is impossible to **mount** a file system if the given directory is in use (or is someone's current directory). It is also impossible to **umount** a file system if that system is in use. Type **cd /** to return to the root directory and then **umount**.

Mounting file systems is not considered a "privileged" activity; any user may mount or unmount file systems so long as the given systems or directory are not in use.

Typing:

mount

alone will give you a list of currently mounted file systems. The **mount** listing is helpful if you are not sure if you've successfully **umounted** the diskette.

If a diskette is physically write-protected, or should not be written on, then it can be mounted "read-only" by giving the **-r** flag after the directory name, as in:

mount /dev/f0 /f0 -r

VENIX will prevent writing on any file or directory in file systems mounted this way. If a diskette is physically write-protected and mounted without the **-r** flag, errors messages will result even if you do not explicitly write on the diskette. This is because VENIX always tries to update the "last-examined" date on files when they are read. The solution in this case is to simply **umount** the diskette and re-**mount** it "read-only."

2.2.4 Using fsck

fsck checks the file system on each partition of the hard disk, or a floppy diskette. An **fsck** of the system and users areas on the hard disks is done automatically. The disk partitions are listed in the file **/etc/checklist**.

To check the file system on the floppy diskette, run the **fsck** command, specifying the device name for the floppy diskette (**/dev/f0**):

fsck /dev/f0

VENIX MAINTENANCE

fsck checks all the blocks on the partitions, and lets the user know if any blocks are bad. **fsck** is talkative; it displays what is happening in some detail on the screen. After the **fsck** is run, you will see the output:

```
/dev/rw0.sys (system area)
```

```
Phase 1 - Check Blocks  
Phase 2 - Not needed  
Phase 3 - Check Pathnames  
Phase 4 - Check Reference Counts  
Phase 5 - Check Free List
```

```
/dev/rw0.usr (user area)
```

```
Phase 1 - Check Blocks  
Phase 2 - Not needed  
Phase 3 - Check Pathnames  
Phase 4 - Check Reference Counts  
Phase 5 - Check Free List
```

If **fsck** does find inconsistencies in a file system, it will usually ask you if you wish to fix things up. In general you can just type “y” (“yes”); however, you should be aware that **fsck** may clean up the file system by removing all inconsistent files.

The “adjusted link counts” and “unreferenced files” found by **fsck** can be fixed with little danger of file loss, unless the directories leading to the file are damaged. The most frequent error is a “bad free list” which can always safely be fixed. If any fixing is done, it is a good idea to run **fsck** a second time since in some cases not all errors will be fixed in just one run.

Sometimes damaged file systems will result in error messages calling some files “BAD” (i.e., files with impossible block locations) and others “DUP”s (files containing blocks also found in other files). In Phase 3, always answer “yes” if asked to remove a file previously marked BAD. If a file has just been reported to have DUP’s previously, you might wish to avoid removing it immediately, since some of these duplicates may be cleared up when the BAD block files are deleted. A second **fsck** can then be run to see which files still contain duplicated blocks.

If you are running **fsck** on a floppy diskette, you have the option to copy important files before **fsck** cleans up the file system. When **fsck** asks for permission to clean up a specific file, you may type “n” (“no”). Temporarily **mount** the file system “read only” on the hard disk:

```
mount /dev/f0 /f0 -r
```

and copy the file to another file system. Then **umount** the floppy diskette, and run **fsck** again on the diskette in the floppy drive.

If **fsck** reports errors and you give permission for fixing, you should immediately reboot the computer after **fsck** finishes.

Inconsistencies in file systems are rare, and are usually due to power failures or other sudden crashes which result in a halt of the computer without giving VENIX time to flush its internal I/O buffers. If you find that **fsck** frequently turns up problems with your hard disk, or floppy diskette, then something may be seriously wrong with your disk hardware or device driver.

2.3 BACKING UP FILES

Files on the winchester can be backed up by three methods:

- **mounting** a diskette and copying the files
- using the **tar** (tape archive) command
- using the **dump** command

Both **tar** and **dump** write out information in their own particular formats, which are totally separate from the VENIX file system format. Diskettes written out by either program are not initialized with **/etc/mkfs** (make a file system). **mounted** diskettes must have a file system.

If you **mount** and copy files to the floppy diskette, any previous iterations of a file are overwritten, and only your most recent version is saved. This saves space on the diskette. One disadvantage is that **mounting** and copying is a lengthy process if you need to a number of files.

For most purposes, **tar** is the best method for backing up files. **dump** is provided only for compatibility with older versions of VENIX; its use is not recommended. **tar** provides all the functionality of **dump**, and considerably more flexibility, so we will not explain use of **dump** here. Interested users can examine the command write-up in the *User Reference Manual*.

2.3.1 Backing Up User Files

The most common way to lose user files is through accidental deletion by mistaken use of wildcard file names. Hardware failures can also wreak havoc with a file system. Since VENIX uses a tree-structured file system, it can be very sensitive to bad blocks in key areas, so frequent backup is recommended.

VENIX MAINTENANCE

The program **tar** can be used to backup user files. **tar** backs up directories, subdirectories, and files. Saving a directory causes all files and sub-directories beneath it to be saved. Incidentally, **tar** is the usual format for transferring files between UNIX, VENIX, and other UNIX-derived systems.

To begin the backup of your files on the winchester, first place a diskette in floppy drive 0 (**/dev/f0**). It is not necessary to calculate the number of files which will fit on the diskette since **tar** will prompt when the diskette is full. When the diskette has no more space, a message will appear on the screen

```
XX blocks used on medium
Change medium and press 'return'.
```

at which point you should take out the full diskette, and insert a new one in the floppy drive.

To clear the diskette and store files on it, use the **c** flag. ("**c**" is a mnemonic for "create.")

If you already have files on the diskette, use the "**r**" ("restor") key to add new files to the diskette. "**r**" appends files to the end of the diskette.

For example, to store the directories **cairo** and **alexandria** onto a floppy disk in drive zero (that is **/dev/f0**), type:

```
tar c cairo alexandria
```

To add the file **sphinx** in the directory **cairo** to the floppy diskette type:

```
tar r cairo/sphinx
```

If you use the "**r**" key, the diskette must already have been initialized with **tar c**.

to add another file **pyramids** to the end of the last file on the floppy on **/dev/f1** (lower drive)

```
tar rvf1 pyramids
```

tar also allows backup of only those files modified after a certain date. To create a new diskette in drive 1 holding only those files in **cairo** modified in the last seven days:

```
tar cdf1 7 cairo
```

To view the files on a **tar** floppy, simply type:

```
tar tv
```

To extract all files from the floppy:

```
tar xv
```

To extract a specific file in a directory, for example, file **pyramids** in the directory **cairo**:

```
tar xv cairo/pyramids
```

tar always restores files using the same pathnames given when the files were saved. This means that full pathnames (that is, names beginning with a “/”) should be avoided. For example, suppose that user Dennis has a home directory

```
/usr/dennis
```

and a directory beneath that:

```
/usr/dennis/letters
```

If directory **letters** is saved using the command

```
tar c /usr/dennis/letters
```

a subsequent extraction, with

```
tar x
```

will always bring back the old files to **/usr/dennis/letters**, overwriting any files of the same name in that directory. For this reason, it is desirable to save files according to a “relative” name, for example from Dennis’ home directory (**/usr/dennis**):

```
cd (change directory to /usr/dennis)
```

```
tar c letters
```

Now the command

```
tar x
```

will extract the saved files into whatever directory Dennis is in when he issues the command. So to restore the saved files to a temporary directory, the following commands could be used:

```
cd
```

```
mkdir temp
```

```
cd temp
```

```
tar x
```

Now the extracted files are placed in

```
/usr/dennis/temp/letters
```

while the directory

```
/usr/dennis/letters
```

VENIX MAINTENANCE

is untouched. Of course, if Dennis really wishes to extract his files into `/usr/dennis/letters`, and overwrite any copies of the same files on disk, he can always use the commands

```
cd
tar x
```

and this will happen.

The following commands can be used to backup the entire system, including system directories:

```
cd /
tar c0 .
```

This will require at least ten diskettes, since you are copying out all the files that were originally supplied you with VENIX as well as any new files you may have created. If you wish to copy only those files you have modified, you can take advantage of the `d` option in `tar` to select those files modified since VENIX was installed. For example, if VENIX was installed two weeks ago, the command:

```
cd /
tar cd0 14 .
```

could be used. The modification dates of files supplied with VENIX will reflect the time the master VENIX system was created, and will likely be considerably earlier than the date of installation.

2.3.2 Backing Up the System

SETTING UP VENIX describes the mechanism for duplicating the VENIX diskettes provided for distribution. If you haven't yet done this, do it now! The command to copy a floppy in drive zero to a floppy in drive one is:

```
dd if = /dev/f0 of = /dev/f1 bs = 10b count = 79
```

This command can be repeated to duplicate each of the distributed diskettes.

2.4 USEFUL COMMANDS AND FILES

2.4.1 Message of the Day

The message (if any) in the file `/etc/motd` (**message of the day**) is given whenever a user logs in. This is a convenient mechanism for sending users general notices (e.g. "disk space low — clean up your files!").

2.4.2 The `/etc/wall` Command

The system administrator can communicate to all users through `/etc/wall` (**w**rite to **a**ll) which acts like `write(1)`, but sends the message to every user logged in.

2.4.3 The `ps` Command

The `ps(1)` command will give a list of all processes running on the system, as well as their process ID's, current state (running, waiting, swapped, etc.) and CPU consumption.

2.4.4 The `kill` Command

The `kill` command can be used to stop a process. Each process has an ID number (pid), which you will see listed when you run the `ps` command. For example, the command:

```
kill 1197
```

will send a "terminate" signal to the process with ID number "1197". This is enough to finish most processes. To be sure, the command:

```
kill -9 process ID
```

will always terminate the process.

If you don't remember the process ID number, or don't want to take time to get a display status (perhaps your printer is going haywire), you can type:

```
kill 0
```

This will terminate all processes, including those which are running in the background.

2.4.5 The `suspend` and `resume` Commands

The `suspend(1)` command will suspend a process with given ID; the process will remain frozen until it is **resumed**. Any user can **suspend** or **resume** his or her own processes; the super-user can do it to any process.

2.4.6 The `chmod` Command

`chmod` can be used to change the access permissions on files. The command:

```
chmod 660 file1 file2
```

changes the permission setting on *file1* and *file2* to the octal value "660". The three digits "660" represent the values for the user, group, and other permission, with each digit encoding the read/write/execute permission.

VENIX MAINTENANCE

4	2	1	4	2	1	4	2	1
<i>read</i>	<i>write</i>	<i>execute</i>	<i>read</i>	<i>write</i>	<i>execute</i>	<i>read</i>	<i>write</i>	<i>execute</i>
USER			GROUP			OTHER		

The value “660”, then, corresponds to read/write permission for user and group, and no permission for others. When a long list of the file is done (**l** command), the permission setting will be seen as:

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

It is also possible to indicate the modes symbolically. For example, the command **chmod o+r file1** adds to category other (**o**) the permission to read (**r**).

2.4.7 The **chown** and **chgrp** Commands

The super user can change user and group file ownerships. **chown** is used to change the ownership of a file. For example, the command:

```
chown max file.a
```

sets the ownership of **file.a** to user **max**.

The **chgrp** command changes the group ownership of a file. The command:

```
chgrp biolab file.a
```

changes the group ownership of **file.a** to group **biolab**.

The **l** command will list information on any file, including user and group ownership. For example, if you type:

```
l file.a
```

VENIX will print the access permission settings, and then the user owner (plus other information about the file):

```
-rw-rw-r-- 1 max 4639 Aug 1 13:24 file.a
```

l -g file.a will do the same, but the group ownership of the file will be listed:

```
-rw-rw-r-- 1 biolab 4639 Aug 1 13:24 file.a
```

User and group ownerships are specified in the **/etc/passwd** file along with the user name and directory. Group ID's and group names are listed in the **/etc/group** file.

2.4.8 The **find** Command

find is a versatile command that can be used to locate files, beneath a directory, that match any of a variety of conditions. For example:

```
find /usr -size +100 -atime +30 -print
```

will list all files under directory **/usr** which are greater than 100 blocks in size and have not been accessed in thirty days.

2.4.9 Clock Daemon

When VENIX comes up, the clock daemon **cron** may be started up by a command in `/etc/rc`. As distributed, however, the **cron** command is commented out. The comment colon needs to be removed to enable **cron** to work.

cron sleeps in the background, waking up periodically to examine the file `/usr/lib/crontab` which contains a list of commands and when to execute them. (The frequency at which **cron** checks the file is given by the first command argument, and is usually once every 10 minutes)

crontab normally contains a reference to run the command `/usr/lib/atrun` every half hour, which in turn executes commands that users have scheduled through **at(1)**. The frequency of **atrun**'s execution can be adjusted here. Also, additional lines can be added to **crontab** to regularly run more commands at certain hours, days of the week or month. (Commands which need only be run once can be scheduled with **at**; commands which must be run regularly should be run directly from **crontab**.) See **cron** in **SYSTEM MAINTENANCE PROCEDURES**, section 8 in this manual.

cron is really very helpful. For an example of how **cron** functions "behind the scenes," the following line could be placed in `/usr/lib/crontab` to invoke **find** at 4am every morning to remove all files whose names begin with "junk" that have not been accessed in thirty days:

```
0 4 * * * find /usr -name "junk*" -atime +30 -exec rm {} \;
```

2.5 SYSTEM ACCOUNTING

2.5.1 Disk Space

Within each file system, VENIX itself cannot restrict the number of disk blocks available to any particular user: any user may extend his or her directories and create new and larger files until the total amount of space available in that file system is reached. When there is no room left on a file system, programs will complain of I/O errors and an "Out of space" message will appear on the console terminal. This will continue until someone removes some files, or until the system administrator removes some.

Listed below are several utilities which are useful for managing disk space. Some of these commands, such as **df**, are applied to entire disk partitions by their `"/dev"` name. Others, such as **du**, are applied to named directories. All of these are described in section one of the *User Reference Manual* along with additional options.

VENIX MAINTENANCE

2.5.2 The df Command

df(1) (“disk free”) determines the amount of free blocks on the system or user partition. For example:

```
df /dev/w0 usr
```

will give a free block count on the user area.

2.5.3 The quot Command

The **quot(1)** command lists the number of blocks per user on the system or user area.

```
quot /dev/w0 usr
```

will give such a list of blocks for each user.

2.5.4 The du Command

du(1) (“disk usage”) determines the number of blocks used under a particular directory, and can give totals for each sub-directory.

```
du /usr/letters/auntie
```

lists the number of blocks used by all files and totals for all subdirectories under **/usr/letters/auntie**.

2.5.5 The ncheck Command

The **ncheck(1)** lists all files and directories on the hard disk or a floppy diskette. This is helpful when you wish to see all the directories listed. (The **l** command lists only the contents of a directory.) For example:

```
ncheck /dev/w0.sys
```

checks the system area.

To check a floppy diskette:

```
ncheck /dev/f0
```

2.6 LOGIN TIME

2.6.1 The ac Command

The **ac(1)** utility can be used to list the amount of terminal time taken by each user, as collected in the **/usr/adm/wtmp** file. The command

```
ac -p
```

will print totals for each user; see **ac(1)** for other options.

Accounting will only be done if the **/usr/adm/wtmp** file exists, so the system administrator should create an empty **wtmp** if accounting of this type is desired.

This can be done with the command

```
cp /dev/null /usr/adm/wtmp
```

Once this file is created, it will grow *ad infinitum*, so relevant information should periodically be collected with **ac** and the file cleared.

2.6.2 The who Command

The **who(1)** command with no arguments will give a list of all users currently logged in, and the time they logged in. The command

```
who /usr/adm/wtmp
```

will give the precise times each user logged in and out over the history of the **wtmp** file.

2.7 SYSTEM FILES AND DIRECTORIES

The following is a quick overview of the system directories and files:

```

/          root
/dev/      device nodes
           console      main console
           com?         terminals
           w0.*         disks
           rw0*         raw disks
           ...
           aw0*         asynchronous raw disks
           ...
/bin/      utility programs, cf /usr/bin (1)
           as           assembler
           cc           C compiler executive, cf /usr/lib/c0
           ...
/lib/      object libraries and other stuff, cf /usr/lib
           libc.a       system calls, standard I/O, etc. (2,3,3S)
           libm.a       math routines (3M)
           libplot.a    plotting routines, plot(3)
           c0           pass of cc(1)
           ...
/etc/      essential data and dangerous maintenance utilities
           passwd       password file (4)
           group        group file (4)
           motd         message of the day, login(1)
           mtab         mounted file table, mtab(4)
           dtab         dump history, dump(1)

```

VENIX MAINTENANCE

ttys	properties of terminals, ttys(4)
getty	part of login, getty(8)
init	the father of all processes, init(8)
rc	shell program to bring system up
cron	the clock daemon, cron(8)
...	
/tmp/	temporary files, cf /usr/tmp
e*	used by ed(1)
pc*	used by cc(1)
...	
/usr/	mounted user file system, general purpose directory
demo/	demonstration
guest/	guest login directory
adm/	administrative information
bin/	utility programs
dict/	hash tables, etc. for spell
games/	games
...	
include/	standard #include files
a.out.h	object file layout, a.out(4)
stdio.h	standard I/O, stdio(3)
math.h	(3M)
...	
sys/	system-defined layouts
...	
lib/	object libraries to keep /lib/ small
atrun	scheduler for at(1)
struct/	passes of struct(1)
tmac/	macros for nroff(1)
suftab	table of suffixes for nroff(1)
...	
/u?/	other mounted user file systems
...	
/venix	bootable image of the kernel
/etc/.profile	login profile for root
/.profile	boot profile

The **dtree(1)** command can be used to generate a complete list of system names.

2.7.1 File Modes

Most system commands and other files are given read/execute permission to everyone, and write permission to the root only. Certain commands which need to be accessible by users, but also do privileged things, are given the “set UID” mode. When these commands are executed by users, the effective ID is made “root” so that privileged things can be done. At the time of this writing the following files were set-UID:

/bin/ps	/bin/mkdir	/bin/su
/bin/mount	/bin/umount	/bin/rmdir
/bin/login	/bin/mv	/bin/at
/bin/rmail	/bin/newgrp	/bin/lpstop
/bin/mail	/bin/passwd	/usr/lib/atrun
/usr/lib/uucp/uucico	/usr/lib/uucp/uuxqt	

These programs will not run correctly without set-UID mode. If these files’ permissions are listed (with `ls` or `l`), an `s` will be shown in place of the “user” `x` bit. The command

chmod u + s file

will make `file` a set-UID program;

chmod u - s file

will turn off set-UID action.

Commonly used programs have another special mode bit, called the “sticky” bit. “Sticky” programs are saved permanently (“stick around”) in the swap area upon their first use (after the system is booted). They can subsequently be run slightly faster than normal, since they can be quickly found in the swap area. At time of writing, the following commands were among those made “sticky”:

/bin/sh	/bin/l	/bin/vi	/lib/c0
/bin/ps	/bin/ls	/bin/ice	/lib/c1

If these files’ permissions are listed, a `t` will be shown in place of the “others” `x` bit.

The sticky bit is only in effect if the programs are pure (type 410), so that the code segment can be shared by multiple users. This corresponds to the `-n` flag of `cc(1)` or `ld(1)`. (Even if the sticky bit is not set, programs of this type will have their code segment shared automatically if they are executed simultaneously by two or more users.)

One special rule about “sticky” commands: after they are run once, they should never be removed or replaced, or the file system will become slightly corrupted. If you need to remove/replace such a file, you should rename the file to something else (e.g. “file.old”) and remove it after the system is next rebooted.

VENIX MAINTENANCE

2.7.2 File Links

Certain system files are linked more than once; that is, there is more than one name corresponding to those files. For example, the `l` and `ls` files are linked together; the actual program checks the second letter of its name (present in `argv[0]`) to determine whether it was called “`ls`” or “`l`”, and acts appropriately. The user can create his or her own links with the `ln(1)` command. The link must be within a mounted file system.

The following files were linked together at the time of this writing:

<code>/bin/l</code>	linked with	<code>/bin/ls</code>
<code>/bin/mount</code>	"	<code>/etc/mount</code>
<code>/bin/umount</code>	"	<code>/etc/umount</code>
<code>/bin/mail</code>	"	<code>/bin/rmail</code>

2.8 SYSTEM ERRORS AND CRASHES

There are two types of system error messages. Recoverable messages indicate some difficulty which the administrator should be aware of, but do not result in a system-wide stoppage. Messages beginning with “`PANIC: ...`” indicate unrecoverable system errors, and result in the system coming to an abrupt halt. (Time is taken to flush the system buffers, however, so the integrity of the file system is usually maintained.)

2.8.1 Reloading VENIX

Should your system crash in such a way that the system area becomes unbootable, you can restore VENIX without losing the user files. This is accomplished with the bootable XFER diskette used for system installation. See **SETTING UP VENIX** for a description of the installation procedures. In Part II of the installation procedure, installing the user area, you will be asked, “Do you wish to restore the user area (y or n)?” Type `n` (“no”). XFER will skip directly to Part III, restoring the system area, leaving the user area untouched. Follow the same loading instructions for the system area that were used when VENIX was originally installed. When asked, “Do you wish to restore the system area (y or n)?”, type `y` (“yes”). Then proceed with the loading process. The system area will be restored, but the user files should be intact. Then, boot the system.

When running the XFER diskette, you also have the capability of running a selected number of VENIX maintenance and backup programs. This is useful if you wish to reinstall the original VENIX user area, but have files on that area you first wish to preserve. The first question that the XFER diskette asks is “Do you wish to install PRO/VENIX (y or n)?” If you answer `n`, a shell will be executed for you. At this point, you may mount the winchester file system, and

extract any files you need. For example, the following commands would be issued to back up the user areas `/usr/George` and `/usr/fred`. First, mount the user area:

```
mount /dev/w0.usr /usr
```

Then insert a blank diskette in drive one and run **tar**:

```
tar cvf /dev/f1 /usr/George /usr/fred
```

Note that **tar** is told to use floppy drive 1 for backup. Be careful not to specify drive 0 (the default device) or you will overwrite the XFER diskette!

After you are finished with the shell, typing CTRL-D will bring you back into the XFER program. You may then continue reinstalling VENIX.

2.8.2 Rebooting After a Crash

When you reboot after a crash, an **fsck** should run automatically on the hard disk partitions. This will check all file systems which could have been in use at the time of the crash (see the section on **fsck** earlier in this chapter). If there are any problems, they should be repaired, (keep running **fsck** until the problems have been fixed).

To boot VENIX at all, three files (and the directories leading to them) must be intact. First, the initialization program `/etc/init` (see **init** in **SYSTEM MAINTENANCE PROCEDURES**, section 8) must be present and executable. If it is not, the CPU will loop in user mode. For **init** to work correctly, `/dev/console` and `/bin/sh` must be present. If either does not exist, the system begins thrashing: **init** will go into a fork/exec loop trying to create a shell with proper standard input and output.

If you cannot get the system to boot, you will have to run on a backup copy. The damaged system may then be repaired with **fsck** as described earlier. If serious surgery is required on the root file system, it is prudent to simply discard it and return to the most recent backup system available, since files may have been insidiously corrupted. If necessary, you can try to salvage any important files modified since the last backup was made.

2.8.3 Recoverable Errors

2.8.3.1 Device Errors

There are a number of situations which will cause messages such as, "Error on floppy, unit 0" or "No inodes on winchester unit 2" to appear on the console terminal. The error is specified in conjunction with a named device — the floppy drive, or winchester. These system errors, coming from the kernel, are always highlighted.

VENIX MAINTENANCE

Floppy disk errors are tagged with labels of units 0 and 1, which refer to drives 0 and 1.† Winchester disk errors specify errors on units 0, 1, 2, and 3, which refer to the temporary, system, swap, and user areas, respectively.

The error message may also indicate “buffered” and “unbuffered”. For example, “Error on floppy (unbuffered), unit 0.” See the driver descriptions in **DEVICES, section 7** of this manual for a complete description of each unit number for the winchester and floppy, buffered and unbuffered.

A long listing of the `/dev` directory, as in

l /dev

will also provide the names for each device entry. A pair of numbers, separated by commas, will be given with each entry, the second of which numbers corresponds with the unit number given in system error messages.

The following is a list of recoverable system error messages.

The device errors are:

Error on . . . (device name)

This is the most common kind of error message and is produced by a device driver. Many of these messages will refer to the floppy disk drive, and will occur for example when an attempt is made to access an empty drive.

These error messages have the general form:

Error on (device name and unit number)

[. . .] while [reading or writing] block number X. Status XXX

Block number refers to the block that was being read from or written to when the error occurred. The status number reflects the contents of the driver error register.

For example, if the left hand disk drive was empty and you tried to read a diskette, the message would appear:

Error on the floppy, unit 0

Diskette improperly inserted while reading block number 0.

Status 0120

Error messages will appear other than “Diskette improperly inserted . . .” “Write protected” indicates that an attempt was made to write on a diskette which had a write protect sticker. Other error

† If you are using non-standard versions of floppy units (e.g. `/dev/f0.m11` for Micro-11 compatibility) then the unit numbers will have 8 or 16 added to them.

messages, such as “CRC error” or “Bad format”, may be due to improper seating of the diskette; however, if they occur repeatedly, after the diskette has been checked and reinserted, then the diskette is bad and will need to be replaced.

It is possible, but unlikely, that you encounter a message: “Error on the winchester, unit ...” “Data mark not found” or “ID not found.” These errors are frequently corrected by reformatting the disk, which can be done only by the bootable XFER diskette. Check the unit number given in the error message. XFER reformats the user area before reinstalling user area files; it reformats the system, temporary, and swap areas before reinstalling system area files. This reinstallation will erase all existing files on the given areas. Depending on the location of the block in error, you may be able to extract needed files from the disk before reformatting.

No space on . . . (*device name*)

All blocks are used up on the device specified (“floppy, unit 0” or “winchester, unit 1”, etc.) See the section on disk space.

Bad block on . . . (*device name*)

A block was found with a number placing it outside the bounds of the file system. Run **fsck** to delete the file with the bad block and fix up the file system.

No inodes on . . . (*device name*)

All the inode pointers for the file system on the device specified have been used up. This should only occur if you have many small files on a file system. When a file system is made, the **/etc/mkfs** command automatically allocates enough inodes to handle most disk requirements. If this is insufficient, you can clear away all files and make a new file system. **/etc/mkfs** allows the user to specify a “prototype” file system giving the number of inodes to be allocated.

Bad superblock count on . . . (*device name*)

An invalid freelist count or invalid inode count was found on the super block of the device. At this point, all the counts are zeroed which will almost certainly lead to a “No space” diagnostic. Reboot, and let **fsck(1)** repair the bad file system.

2.8.3.2 Non-device (Internal) Errors

The error messages described in this section occur because certain system resources are totally consumed and the system will have to stop a program. The

VENIX MAINTENANCE

system will continue running, but the cause of the error (e.g. invalid interrupt) should be checked out.

A number of these error messages refer to the **param.h**, **config**, and **l.s** files which control various system tables. These references are provided only for the benefit of users who have PRO/VENIX system configuration kits, and thus can modify the files and create a new kernel.

Out of swap space

A program or shared segment needs to be swapped out, and there is no swap space left on disk.

Segment table full

A shared text (pure procedure) program is being executed or a shared data segment is being created, and the table for such things is full. The maximum number of shared segments is set by the NSEG parameter in the **param.h** file.

In-core inode table full

The in-core inode table is full, causing an `open()` call to fail. The maximum number of in-core inodes is set by the NINODE parameter in the **param.h** file.

Fast alarm overflow

The in-core timeout table (used by fast alarms, etc.) is full and a timeout call has been lost. The maximum number of timeouts is set by the NCALL parameter in the **param.h** file.

In-core file table full

The in-core file table (one for each open file on the system) is full, causing an `open()` call to fail. The maximum number of files is set by the NFILE parameter in the **param.h** file.

Unknown interrupt at X

An unexpected interrupt has occurred through interrupt vector X (octal) modulo 0100. Only the low 6 address bits are significant. Either the **l.s** table is set up incorrectly or the hardware has generated a bogus interrupt.

2.8.4 Unrecoverable Errors (Panics)

The following messages will appear after the word "PANIC", and are followed by a total system hang-up. These messages will not be useful to most users since these kinds of problems can only be fixed by modifying the system libraries or tables, which requires a PRO/VENIX system configuration kit.

Invalid major device number

The **getblk** routine was called with a nonexistent major device as an argument. Check the “rootdev,” “swapdev,” and “pipedev” assignments in the **config** file

Missing d-tab entry

Null device table entry for the major device used as an argument to **getblk**.

Initialization I/O error

An I/O error reading the super-block for the root file system during initialization. This is probably a hardware error.

Bad system table (getfs)

An internal VENIX table has been corrupted, discovered by routine **getfs**.

Bad system table (iget)

Same as above, except the routine which discovered the error is **iget**.

Bad system table (newproc)

Same as above, except routine is **newproc**.

Swap error

An unrecoverable I/O error during a swap. It really shouldn't be a PANIC, but it's hard to fix.

Missing entry (unlink)

The directory containing a file being deleted can't be found.

Trap

An unexpected trap has occurred within the system. This is accompanied by two numbers: a *pc* which was the value (in octal) of the program counter when the trap took place; and a *traptype* which encodes which trap occurred.

The trap types are:

- 0 bus error
- 1 illegal instruction
- 2 BPT/trace
- 3 IOT
- 4 power fail
- 5 EMT
- 6 recursive system call (TRAP instruction)

VENIX MAINTENANCE

- 7 programmed interrupt
- 8 floating point trap
- 9 segmentation violation
- 10 cache parity error

In some of these cases it is possible for 16 to be added into the trap type; this indicates that the processor was in user mode when the trap occurred.

A symbol table listing of the kernel can be obtained with the command

nm -ng /venix

Users with PRO/VENIX configuration kits can use this table to locate the routine in which the trap occurred.

Contents

UUCP IMPLEMENTATION DESCRIPTION	3-1
3.1 INTRODUCTION	3-1
3.2 UUCP — UNIX TO UNIX FILE COPY.....	3-2
3.3 UUX — UNIX TO UNIX EXECUTION	3-4
3.4 UUCICO — COPY IN, COPY OUT	3-7
3.5 UUXQT — UUCP COMMAND EXECUTION.....	3-11
3.6 UULOG — UUCP LOG INQUIRY	3-11
3.7 UUCLEAN — UUCP SPOOL DIRECTORY CLEANUP.....	3-12
3.8 SECURITY	3-12
3.9 UUCP INSTALLATION	3-13
3.10 LOGIN/SYSTEM NAMES.....	3-15
3.11 ADMINISTRATION.....	3-19
3.12 SHELL FILES	3-21
3.13 LOGIN ENTRY	3-21

Chapter 3

UUCP IMPLEMENTATION DESCRIPTION

3.1 INTRODUCTION

Uucp is a series of programs designed to permit communication among UNIX and VENIX systems using either dial-up or hardwired communication lines. It is used for file transfers and remote command execution. This paper describes the second implementation of the system.

Uucp is a batch type operation. Files are created in a spool directory for processing by uucp daemons. There are three types of files used for the execution of work. Data files contain data for transfer to remote systems. Work files contain directions for file transfers among systems. Execution files are directions for UNIX command executions which involve the resources of one or more systems.

The uucp system consists of four primary and two secondary programs. The primary programs are:

- uucp This program creates work and gathers data files in the spool directory for the transmission of files.
- uux This program creates work files, execute files and gathers data files for the remote execution of UNIX commands.
- uucico This program executes the work files for data transmission.
- uuxqt This program executes the execution files for UNIX command execution.

UUCP

The secondary programs are:

- `uulog` This program updates the log file with new entries and reports on the status of uucp requests.
- `uuclean` This program removes old files from the spool directory.

The remainder of this chapter describes the operation of each program, the installation of the system, the security aspects of the system, the files required for execution, and the administration of the system.

3.2 UUCP — UNIX TO UNIX FILE COPY

The `uucp` command is the user's primary interface with the system. The `uucp` command was designed to look like `cp` to the user. The syntax is:

```
uucp [option] ... source ... destination
```

where the *source* and *destination* may contain the prefix *system-name!* which indicates the system on which the file or files reside or where they will be copied.

The options interpreted by `uucp` are:

- `-d` Make directories when necessary for copying the file.
- `-c` Don't copy source files to the spool directory, but use the specified source when the actual transfer takes place.
- `-gletter` Put *letter* in as the grade in the name of the work file. (This can be used to change the order of work for a particular machine.)
- `-m` Send mail on completion of the work.

The following options are used primarily for debugging:

- `-r` Queue the job but do not start the uucico program.
- `-sdir` Use directory *dir* for the spool directory.
- `-xnum` *Num* is the level of debugging output desired.

The destination may be a directory name, in which case the file name is taken from the last part of the source's name. The source's name may contain special shell characters such as "`?*[]`". If a source argument has a *system-name!* prefix for a remote system, the file name expansion will be done on the remote system.

The command:

```
uucp *.c usg!/usr/dan
```

will set up the transfer of all files whose names end with “.c” to the “/usr/dan” directory on the “usg” machine.

The source and/or destination names may also contain a *~user* prefix. This translates to the login directory on the specified system. For names with partial path-names, the current directory is prepended to the file name. File names with *../* are not permitted.

The command:

```
uucp usg!~dan/*.h ~dan
```

will set up the transfer of files whose names end with “.h” in dan’s login directory on system “usg” to dan’s local login directory.

For each source file, the program will check the source and destination file-names and the system-part of each to classify the work into one of five types:

- [1] Copy source to destination on local system.
- [2] Receive files from other systems.
- [3] Send files to remote systems.
- [4] Send files from remote systems to other remote systems.
- [5] Receive files from remote systems when the source contains special shell characters as mentioned above.

After the work has been set up in the spool directory, the uucico program is started to try to contact the other machine to execute the work (unless the *-r* option was specified).

3.2.1 Type 1

A *cp* command is used to do the work. The *-d* and the *-m* options are not honored in this case.

3.2.2 Type 2

A one line work file is created for each file requested and put in the spool directory with the following fields, each separated by a blank. (All work files and execute files use a blank as the field separator.)

UUCP

- [1] R
- [2] The full path-name of the source or a *~user/path-name*. The *~user* part will be expanded on the remote system.
- [3] The full path-name of the destination file. If the *~user* notation is used, it will be immediately expanded to be the login directory for the user.
- [4] The user's login name.
- [5] A “-” followed by an option list. (Only the *-m* and *-d* options will appear in this list.)

3.2.3 Type 3

For each source file, a work file is created and the source file is copied into a data file in the spool directory. (A *-c* option on the **uucp** command will prevent the data file from being made. In this case, the file will be transmitted from the indicated source.) The fields of each entry are given below:

- [1] S
- [2] The full-path name of the source file.
- [3] The full-path name of the destination or *~user/file-name*.
- [4] The user's login name.
- [5] A “-” followed by an option list.
- [6] The name of the data file in the spool directory.
- [7] The file mode bits of the source file in octal print format (e.g. 0666).

3.2.4 Type 4 and Type 5

Uucp generates a **uucp** command and sends it to the remote machine; the remote uucico executes the **uucp** command.

3.3 UUX — UNIX TO UNIX EXECUTION

The **uux** command is used to set up the execution of a UNIX command where the execution machine and/or some of the files are remote. The syntax of the **uux** command is:

```
uux [- ] [option] ... command-string
```

where the command-string is made up of one or more arguments. All special shell characters such as “<>|^” must be quoted either by quoting the entire command-string or quoting the character as a separate argument. Within the command-string, the command and file names may contain a *system-name!* prefix. All arguments which do not contain an “!” will not be treated as files. (They will not be copied to the execution machine.) The “-” is used to indicate that the standard input for *command-string* should be inherited from the standard input of the **uux** command. The options, essentially for debugging, are:

- r Don’t start uucico or uuxqt after queuing the job;
- xnum *Num* is the level of debugging output desired.

The command:

```
pr abc | uux - usg!lpr
```

will set up the output of “pr abc” as standard input to an “lpr” command to be executed on system “usg”.

uux generates an execute file which contains the names of the files required for execution (including standard input), the user’s login name, the destination of the standard output, and the command to be executed. This file is either put in the spool directory for local execution or sent to the remote system using a generated send command (type 3 above).

For required files which are not on the execution machine, **uux** will generate receive command files (type 2 above). These command-files will be put on the execution machine and executed by the uucico program. (This will work only if the local system has permission to put files in the remote spool directory as controlled by the remote *USERFILE*.)

The execute file will be processed by the uuxqt program on the execution machine. It is made up of several lines, each of which contains an identification character and one or more arguments. The order of the lines in the file is not relevant and some of the lines may not be present. Each line is described below:

3.3.1 User Line

```
U user system
```

where the *user* and *system* are the requester’s login name and system.

UUCP

3.3.2 Required File Line

F *file-name real-name*

where the *file-name* is the generated name of a file for the execute machine and *real-name* is the last part of the actual file name (contains no path information). Zero or more of these lines may be present in the execute file. The uuxqt program will check for the existence of all required files before the command is executed.

3.3.3 Standard Input Line

I *file-name*

The standard input is either specified by a “<” in the command-string or inherited from the standard input of the **uux** command if the “-” option is used. If a standard input is not specified, **/dev/null** is used.

3.3.4 Standard Output Line

O *file-name system-name*

The standard output is specified by a “>” within the command-string. If a standard output is not specified, **/dev/null** is used. (Note: the use of “>>” is not implemented.)

3.3.5 Command Line

C *command [arguments] ...*

The *arguments* are those specified in the command-string. The standard input and standard output will not appear on this line. All required files will be moved to the execution directory (a subdirectory of the spool directory) and the UNIX command is executed using the Shell. In addition, a shell “PATH” statement is prepended to the command line as specified in the uuxqt program.

After execution, the standard output is copied or set up to be sent to the proper place.

3.4 UUCICO — COPY IN, COPY OUT

The uucico program will perform the following major functions:

- Scan the spool directory for work.
- Place a call to a remote system.
- Negotiate a line protocol to be used.
- Execute all requests from both systems.
- Log work requests and work completions.

Uucico may be started in several ways:

- a) by a system daemon,
- b) by one of the uucp, uux, uuxqt or uucico programs,
- c) directly by the user (this is usually for testing),
- d) by a remote system. (The uucico program should be specified as the “shell” field in the `/etc/passwd` file for the uucp logins.)

When started by method a, b or c, the program is considered to be in MASTER mode. In this mode, a connection will be made to a remote system. If started by a remote system (method d), the program is considered to be in SLAVE mode.

The MASTER mode will operate in one of two ways. If no system name is specified (`-s` option not specified) the program will scan the spool directory for systems to call. If a system name is specified, that system will be called, and work will only be done for that system.

The uucico program is generally started by another program. There are several options used for execution:

- `-r1` Start the program in MASTER mode. This is used when uucico is started by a program or “cron” shell.
- `-ssys` Do work only for system `sys`. If `-s` is specified, a call to the specified system will be made even if there is no work for system `sys` in the spool directory. This is useful for polling systems which do not have the hardware to initiate a connection.

The following options are used primarily for debugging:

UUCP

- *ddir* Use directory *dir* for the spool directory.
- *xnum* *Num* is the level of debugging output desired.

The next part of this section will describe the major steps within the `uucico` program.

3.4.1 Scan For Work

The names of the work related files in the spool directory have format:

type . system-name grade number

where:

Type is an upper case letter, (C= copy command file, D= data file, X= execute file);

System-name is the remote system;

Grade is a character;

Number is a four digit, padded sequence number.

The file

`C.res45n0031`

would be a work file for a file transfer between the local machine and the "res45" machine.

The scan for work is done by looking through the spool directory for work files (files with prefix "C."). A list is made of all systems to be called. `Uucico` will then call each system and process all work files.

3.4.2 Call Remote System

The call is made using information from several files which reside in the `uucp` program directory. At the start of the call process, a lock is set to forbid multiple conversations between the same two systems.

The system name is found in the `L.sys` file. The information contained for each system is:

[1] system name,

- [2] times to call the system (days-of-week and times-of-day),
- [3] device or device type to be used for call,
- [4] line speed,
- [5] phone number if field [3] is ACU or the device name (same as field [3]) if not ACU,
- [6] login information (multiple fields),

The time field is checked against the present time to see if the call should be made.

The phone number may contain abbreviations (e.g. mh, py, boston) which get translated into dial sequences using the **L-dialcodes** file.

The **L-devices** file is scanned using fields [3] and [4] from the **L.sys** file to find an available device for the call. The program will try all devices which satisfy [3] and [4] until the call is made, or no more devices can be tried. If a device is successfully opened, a lock file is created so that another copy of uucico will not try to use it. If the call is complete, the login information (field [6] of **L.sys**) is used to login.

The conversation between the two uucico programs begins with a handshake started by the called, SLAVE system. The SLAVE sends a message to let the MASTER know it is ready to receive the system identification and conversation sequence number. The response from the MASTER is verified by the SLAVE and if acceptable, protocol selection begins. The SLAVE can also reply with a "call-back required" message in which case, the current conversation is terminated.

3.4.3 Line Protocol Selection

The remote system sends a message

Pproto-list

representing a line protocol.

The calling program checks the proto-list for a letter corresponding to an available line protocol and returns a use-protocol message. The use-protocol message is

Ucode

UUCP

where *code* is either a one character protocol letter or N which means there is no common protocol.

3.4.4 Work Processing

The initial roles (MASTER or SLAVE) for the work processing are the mode in which each program starts. (The MASTER has been specified by the `-r1 uucico` option.) The MASTER program does a work search similar to the one used in the "Scan For Work" section.

There are five messages used during the work processing, each specified by the first character of the message. They are:

- S send a file,
- R receive a file,
- C copy complete,
- X execute a uucp command,
- H hangup.

The MASTER will send **R**, **S** or **X** messages until all work from the spool directory is complete, at which point an **H** message will be sent. The SLAVE will reply with **SY**, **SN**, **RY**, **RN**, **HY**, **HN**, **XY**, **XN**, corresponding to yes or no for each request.

The send and receive replies are based on permission to access the requested file/directory using the *USERFILE* and read/write permissions of the file/directory. After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message **CY** will be sent if the file has successfully been moved from the temporary spool file to the actual destination. Otherwise, a **CN** message is sent. (In the case of **CN**, the transferred file will be in the spool directory with a name beginning with "TM".) The requests and results are logged on both systems.

The hangup response is determined by the SLAVE program by a work scan of the spool directory. If work for the remote system exists in the SLAVE's spool directory, an **HN** message is sent and the programs switch roles. If no work exists, an **HY** response is sent.

3.4.5 Conversation Termination

When a **HY** message is received by the MASTER it is echoed back to the SLAVE and the protocols are turned off. Each program sends a final “OO” message to the other. The original SLAVE program will clean up and terminate. The MASTER will proceed to call other systems and process work as long as possible or terminate if a `-s` option was specified.

3.5 UUXQT — UUCP COMMAND EXECUTION

The `uuxqt` program is used to execute files generated by `uux`. The `uuxqt` program may be started by either the `uucico` or `uux` programs. The program scans the spool directory for execute files (prefix “X.”). Each one is checked to see if all the required files are available and if so, the command line or send line is executed.

The execute file is described in the “Uux” section above.

3.5.1 Command Execution

The execution is accomplished by executing a `sh -c` of the command line after appropriate standard input and standard output have been opened. If a standard output is specified, the program will create a send command or copy the output file as appropriate.

3.6 UULOG — UUCP LOG INQUIRY

The `uucp` programs create individual log files for each program invocation. Periodically, `uulog` may be executed to prepend these files to the system logfile. This method of logging was chosen to minimize file locking of the logfile during program execution.

The `uulog` program merges the individual log files and outputs specified log entries. The output request is specified by the use of the following options:

- `-sys` Print entries where `sys` is the remote system name;
- `-user` Print entries for user `user`.

The intersection of lines satisfying the two options is output. A null `sys` or `user` means all system names or users respectively.

3.7 UUCLEAN — UUCP SPOOL DIRECTORY CLEANUP

This program is typically started by the daemon, once a day. Its function is to remove files from the spool directory which are more than 3 days old. These are usually files for work which can not be completed.

The options available are:

- *-ddir* The directory to be scanned is *dir*.
- *-m* Send mail to the owner of each file being removed. (Note that most files put into the spool directory will be owned by the owner of the uucp programs since the setuid bit will be set on these programs. The mail will therefore most often go to the owner of the uucp programs.)
- *-nhours* Change the aging time from 72 hours to *hours* hours.
- *-ppre* Examine files with prefix *pre* for deletion. (Up to 10 file prefixes may be specified.)
- *-xnum* This is the level of debugging output desired.

3.8 SECURITY

The uucp system, left unrestricted, will let any outside user execute any commands and copy in/out any file which is readable/writable by the uucp login user. It is up to the individual sites to be aware of this and apply the protections that they feel are necessary.

There are several security features available aside from the normal file mode protections. These must be set up by the installer of the uucp system.

- The login for uucp does not get a standard shell. Instead, the uucico program is started. Therefore, the only work that can be done is through uucico.
- A path check is done on file names that are to be sent or received. The *USERFILE* supplies the information for these checks. The *USERFILE* can also be setup to require call-back for certain login-ids. (See the “Files required for execution” section for the file description.)
- A conversation sequence count can be set up so that the called system can be more confident that the caller is who he or she says he/she is.
- The uuxqt program comes with a list of commands that it will execute. A “PATH” shell statement is prepended to the command line as specified in the

uuxqt program. The installer may modify the list or remove the restrictions as desired.

- The **L.sys** file should be owned by uucp and have mode 0400 to protect the phone numbers and login information for remote sites. (Programs uucp, uucico, uux, uuxqt should be also owned by uucp and have the setuid bit set.)

3.9 UUCP INSTALLATION

There are several source modifications that may be required before the system programs are compiled. These relate to the directories used during compilation, the directories used during execution, and the local uucp *system-name*. Changes to these sources, however, can be done only by installations with UNIX source licenses. Except where noted, installations with binary-only licenses must use the predefined default names.

The four directories are:

- lib** (**/usr/src/cmd/uucp**) This directory contains the source files for generating the uucp system.
- program** (**/usr/lib/uucp**) This is the directory used for the executable system programs and the system files.
- spool** (**/usr/spool/uucp**) This is the spool directory used used during uucp execution.
- xqtdir** (**/usr/spool/uucp/.XQTDIR**) This directory is used during execution of execute files.

The names given in parentheses above are the default values for the directories. The italicized names *lib*, *program*, *xqtdir* and *spool* will be used in the following text to represent the appropriate directory names.

The file **/usr/include/ident.h** may be modified to change the installation name uucp uses. This file is read by uucp at run-time (even though it appears to be a C source file); therefore, even those installations possessing a binary-only UNIX license may change their uucp name.

There are two files which may require modification if the installation holds a source license: the **makefile** file and the **uucp.h** file. The following paragraphs describe the modifications. The modes of *spool* and *xqtdir* should be made "0777".

UUCP

3.9.1 Uucp.h Modification

Change the *program* and the *spool* names from the default values to the directory names to be used on the local system using global edit commands.

3.9.2 Makefile Modification

There are several **make** variable definitions which may need modification.

- | | |
|--------|--|
| INSDIR | This is the <i>program</i> directory (e.g. INSDIR = /usr/lib/uucp). This parameter is used if “make cp” is used after the programs are compiled. |
| IOCTL | This is required to be set if an appropriate ioctl interface subroutine does not exist in the standard “C” library; the statement “IOCTL=ioctl.o” is required in this case. |
| PKON | The statement “PKON=pkcon.o” is required if the packet driver is not in the kernel. |

3.9.3 Compile the System

The command:

make

will compile the entire system. The command

makecp

will copy the commands to the appropriate directories.

The programs uucp, uux, and uulog should be put in “/usr/bin”. The programs uuxqt, uucico, and uuclean should be put in the *program* directory.

3.9.4 Files Required for Execution

There are four files which are required for execution, all of which should reside in the *program* directory. The field separator for all files is a space unless otherwise specified.

3.9.5 L-devices

This file contains entries for the call-unit devices and hardwired connections which are to be used by uucp. The special device files are assumed to be in the /dev directory. The format for each entry is:

line call-unit speed

where:

line is the device for the line (e.g. cul0),
call-unit is the automatic call unit associated with *line* (e.g. cua0),
 (Hardwired lines have a number “0” in this field.),
speed is the line speed.

The line:

```
cul0 cua0 300
```

would be set up for a system which had device “cul0” wired to a call-unit “cua0” for use at “300” baud.

3.9.6 L-dialcodes

This file contains entries with location abbreviations used in the **L.sys** file (e.g. py, mh, boston). The entry format is:

```
abb dial-seq
```

where;

abb is the abbreviation,
dial-seq is the dial sequence to call that location.

The line:

```
py 165 -
```

would be set up so that entry “py7777” would send “165-7777” to the dial-unit.

3.10 LOGIN/SYSTEM NAMES

It is assumed that the login name used by a remote computer to call into a local computer is not the same as the login name of a normal user of that local machine. However, several remote computers may employ the same login name.

Each computer is given a unique system name which is transmitted at the start of each call. This name identifies the calling machine to the called machine.

3.10.1 USERFILE

This file contains user accessibility information. It specifies four types of constraint:

- [1] which files can be accessed by a normal user of the local machine,
- [2] which files can be accessed from a remote computer,
- [3] which login name is used by a particular remote computer,
- [4] whether a remote computer should be called back in order to confirm its identity.

Each line in the file has the following format:

login,sys [*c*] *path-name* [*path-name*] ...

where:

- login* is the login name for a user or the remote computer,
- sys* is the system name for a remote computer,
- c* is the optional call-back required flag,
- path-name* is a path-name prefix that is acceptable for user.

The constraints are implemented as follows:

- [1] When the program is obeying a command stored on the local machine, MASTER mode, the path-names allowed are those given for the first line in the *USERFILE* that has a login name that matches the login name of the user who entered the command. If no such line is found, the first line with a null login name is used.
- [2] When the program is responding to a command from a remote machine, SLAVE mode, the path-names allowed are those given for the first line in the file that has the system name that matches the system name of the remote machine. If no such line is found, the first one with a null system name is used.
- [3] When a remote computer logs in, the login name that it uses must appear in the *USERFILE*. There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a null system name.
- [4] If the line matched in [3] contains a "c", the remote machine is called back before any transactions take place.

The line:

```
u,m /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with “/usr/xyz”.

The line

```
dan, /usr/dan
```

allows the ordinary user *dan* to issue commands for files whose name starts with “/usr/dan”.

The lines:

```
u,m /usr/xyz /usr/spool  
u, /usr/spool
```

allows any remote machine to login with name *u*, but if its system name is not *m*, it can only ask to transfer files whose names start with “/usr/spool”.

The lines:

```
root, /  
, /usr
```

allows any user to transfer files beginning with “/usr” but the user with login root can transfer any file.

3.10.2 L.sys

Each entry in this file represents one system which can be called by the local uucp programs. The fields are described below.

3.10.2.1 system name

The name of the remote system.

3.10.2.2 time

This is a string which indicates the days-of-week and times-of-day when the system should be called (e.g. MoTuTh0800 – 1730).

UUCP

The day portion may be a list containing some of

Su Mo Tu We Th Fr Sa

or it may be **Wk** for any week-day or **Any** for any day.

The time should be a range of times (e.g. 0800–1230). If no time portion is specified, any time of day is assumed to be ok for the call.

3.10.2.3 device

This is either ACU or the hardwired device to be used for the call. For the hardwired case, the last part of the special file name is used (e.g. tty0).

3.10.2.4 speed

This is the line speed for the call (e.g. 300).

3.10.2.5 phone

The phone number is made up of an optional alphabetic abbreviation and a numeric part. The abbreviation is one which appears in the **L-dialcodes** file (e.g. mh5900, boston995–9980).

For the hardwired devices, this field contains the same string as used for the device field.

3.10.2.6 login

The login information is given as a series of fields and subfields in the format

expect send [expect send] ...

where; *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The *expect* field may be made up of subfields of the form:

expect[–send–expect]...

where the *send* is sent if the prior *expect* is not successfully read and the *expect* following the *send* is the next expected string.

There are two special names available to be sent during the login sequence. The string “EOT” will send an EOT character and the string “BREAK” will try to send a BREAK character. (The BREAK character is simulated using line speed changes and null characters and may not work on all devices and/or systems.)

A typical entry in the `L.sys` file would be:

```
sys Any ACU 300 mh7654 login uucp ssword: word
```

The expect algorithm looks at the last part of the string as illustrated in the password field.

3.11 ADMINISTRATION

This section indicates some events and files which must be administered for the uucp system. Some administration can be accomplished by shell files which can be initiated by crontab entries. Others will require manual intervention. Some sample shell files are given toward the end of this section.

3.11.1 SQFILE — Sequence Check File

This file is set up in the program directory and contains an entry for each remote system with which you agree to perform conversation sequence checks. The initial entry is just the system name of the remote system. The first conversation will add two items to the line, the conversation count, and the date/time of the most recent conversation. These items will be updated with each conversation. If a sequence check fails, the entry will have to be adjusted.

3.11.2 TM — Temporary Data Files

These files are created in the spool directory while files are being copied from a remote machine. Their names have the form

TM.pid.ddd

where *pid* is a process-id and *ddd* is a sequential three digit number starting at zero for each invocation of uucico and incremented for each file received.

After the entire remote file is received, the *TM* file is moved/copied to the requested destination. If processing is abnormally terminated or the move/copy fails, the file will remain in the spool directory.

The leftover files should be periodically removed; the uuclean program is useful in this regard. The command:

```
uuclean -pTM
```

will remove all *TM* files older than three days.

UUCP

3.11.3 LOG — Log Entry Files

During execution of programs, individual *LOG* files are created in the spool directory with information about queued requests, calls to remote systems, execution of **uux** commands and file copy results. These files should be combined into the *LOGFILE* by using the *uulog* program. This program will put the new *LOG* files at the beginning of the existing *LOGFILE*. The command:

uulog

will accomplish the merge. Options are available to print some or all the log entries after the files are merged. The *LOGFILE* should be removed periodically since it is copied each time new *LOG* entries are put into the file.

The *LOG* files are created initially with mode 0222. If the program which creates the file terminates normally, it changes the mode to 0666. Aborted runs may leave the files with mode 0222 and the *uulog* program will not read or remove them. To remove them, either use **rm**, **uuclean**, or change the mode to 0666 and let **uulog** merge them with the *LOGFILE*.

3.11.4 STST — System Status Files

These files are created in the spool directory by the *uucico* program. They contain information of failures such as login, dialup or sequence check and will contain a TALKING status when two machines are conversing. The form of the file name is

STST.sys

where *sys* is the remote system name.

For ordinary failures (dialup, login), the file will prevent repeated tries for about one hour. For sequence check failures, the file must be removed before any future attempts to converse with that remote system.

If the file is left due to an aborted run, it may contain a TALKING status. In this case, the file must be removed before a conversation is attempted.

3.11.5 LCK — Lock Files

Lock files are created for each device in use (e.g. automatic calling unit) and each system conversing. This prevents duplicate conversations and multiple attempts to use the same devices. The form of the lock file name is:

LCK..str

where *str* is either a device or system name. The files may be left in the spool directory if runs abort. They will be ignored (reused) after a time of about 24

hours. When runs abort and calls are desired before the time limit, the lock files should be removed.

3.12 SHELL FILES

The uucp program will spool work and attempt to start the uucico program, but the starting of uucico will sometimes fail. (No devices available, login failures etc.). Therefore, the **uucico** program should be periodically started. The command to start uucico can be put in a "shell" file with a command to merge *LOG* files and started by a crontab entry on an hourly basis. The file could contain the commands:

```
program /uulog
program /uucico -r1
```

Note that the **-r1** option is required to start the uucico program in MASTER mode.

Another shell file may be set up on a daily basis to remove *TM*, *ST* and *LCK* files and *C.* or *D.* files for work which can not be accomplished for reasons like bad phone number, login changes etc. A shell file containing commands like

```
program /uuclean -pTM -pC. -pD.
program /uuclean -pST -pLCK -n12
```

can be used. Note the **-n12** option causes the *ST* and *LCK* files older than 12 hours to be deleted. The absence of the **-n** option will use a three day time limit.

A daily or weekly shell should also be created to remove or save old *LOGFILES*. A shell like:

```
cp spool /LOGFILE spool /o.LOGFILE
rm spool /LOGFILE
```

can be used.

3.13 LOGIN ENTRY

One or more logins should be set up for uucp. Each of the */etc/passwd* entries should have the *program/uucico* as the shell to be executed. The login directory is not used, but if the system has a special directory for use by the users for sending or receiving files, it should be given as the login entry. The various logins are used in conjunction with the *USERFILE* to restrict file access. Specifying the shell argument limits the login to the use of uucp (uucico) only.

UUCP

3.13.1 File Modes

It is suggested that the owner and file modes of various programs and files be set as follows.

The programs `uucp`, `uux`, `uucico` and `uuxqt` should be owned by the `uucp` login with the “setuid” bit set and only execute permissions (e.g. mode 04111). This will prevent outsiders from modifying the programs to get at a standard shell for the `uucp` logins.

The `L.sys`, `SQFILE` and the `USERFILE` which are put in the program directory should be owned by the `uucp` login and set with mode 0400.

Contents

A DIAL-UP NETWORK OF UNIX SYSTEMS.....	4-1
4.1 SYSTEM OPERATION AND DESIGN	4-1
4.2 PROCESSING	4-2
4.3 PRESENT USES	4-5
4.4 PERFORMANCE.....	4-6
4.5 SYSTEM GOALS	4-6

Chapter 4

A DIAL-UP NETWORK OF UNIX SYSTEMS

4.1 SYSTEM OPERATION AND DESIGN

The basic operation of the network is very simple. Each participating system has a spool directory, in which work to be done (files to be moved, or commands to be executed remotely) is stored. A standard program, **uucico**, performs all transfers. This program starts by identifying a particular communication channel to a remote system with which it will hold a conversation. **uucico** then selects a device and establishes the connection, logs onto the remote machine and starts the **uucico** program on the remote machine. Once two of these programs are connected, they first agree on a line protocol, and then start exchanging work. Each program in turn, beginning with the calling (active system) program, transmits everything it needs, and then asks the other what it wants done. Eventually neither has any more work, and both exit.

In this way, all services are available from all sites; passive sites (systems that do not have the hardware to initiate a connection), however, must wait until called. A variety of protocols may be used; this conforms to the real, non-standard world. As long as the caller and called programs have a protocol in common, they can communicate. Furthermore, each caller knows the hours when each destination system should be called. If a destination is unavailable, the data intended for it remain in the spool directory until the destination machine can be reached.

The implementation of this network among independent sites, all of which store proprietary programs and data, illustrates the pervasive need for security and administrative controls over file access. Each site, in configuring its programs and system files, limits and monitors transmission. In order to access a file a user needs access permission for the machine that contains the file and access permission for the file itself. This is achieved by first requiring the user to use a password to log into the local machine and then the local machine logs into the remote machine whose files are to be accessed. In addition, records are kept identifying all files that are moved into and out of the local system, and how the requestor of such accesses is identified. Some sites may arrange to permit users only to call up and request work to be done; the calling users are then called back

NETWORK

before the work is actually done. It is then possible to verify that the request is legitimate from the standpoint of the target system, as well as the originating system. Furthermore, because of the call-back, no site can masquerade as another even if it knows all the necessary passwords.

Each machine can optionally maintain a sequence count for conversations with other machines and can require a verification of the count at the start of each conversation. Thus, even if call back is not in use, a successful masquerade requires the calling party to present the correct sequence number. A would-be impersonator must not just steal the correct phone number, user name, and password, but also the sequence count, and must call in sufficiently promptly to precede the next legitimate request from either side. Even a successful masquerade will be detected on the next correct conversation.

4.2 PROCESSING

The user has two commands which set up communications, **uucp** to set up file copying, and **uux** to set up command execution where some of the required resources (system and/or files) are not on the local machine. Each of these commands will put work and data files into the spool directory for execution by **uucp** daemons.

4.2.1 File Copy

The **uucico** program is used to perform all communications between the two systems. It performs the following functions:

- [1] Scan the spool directory for work.
- [2] Place a call to a remote system.
- [3] Negotiate a line protocol to be used.
- [4] Start program **uucico** on the remote system.
- [5] Execute all requests from both systems.
- [6] Log work requests and work completions.

uucico may be started in several ways;

- a) by a system daemon,
- b) by one of the **uucp** or **uux** programs,
- c) by a remote system.

4.2.2 Scan For Work

The file names in the spool directory are constructed to allow the daemon programs (**uucico**, **uuxqt**) to determine the files they should look at, the remote machines they should call and the order in which the files for a particular remote machine should be processed.

4.2.3 Call Remote System

The call is made using information from several files which reside in the uucp program directory. At the start of the call process, a lock is set on the system being called so that another call will not be attempted at the same time.

The system name is found in a “systems” file. The information contained for each system is:

- [1] system name,
- [2] times to call the system (days-of-week and times-of-day),
- [3] device or device type to be used for call,
- [4] line speed,
- [5] phone number,
- [6] login information (multiple fields).

The time field is checked against the present time to see if the call should be made. The *phone number* may contain abbreviations (e.g. “nyc”, “boston”) which get translated into dial sequences using a “dial-codes” file. This permits the same *phonenumber* to be stored at every site, despite local variations in telephone services and dialing conventions.

A “devices” file is scanned using fields [3] and [4] from the “systems” file to find an available device for the connection. The program will try all devices which satisfy [3] and [4] until a connection is made, or no more devices can be tried. If a non-multiplexable device is successfully opened, a lock file is created so that another copy of **uucico** will not try to use it. If the connection is complete, the *login information* is used to log into the remote system. Then a command is sent to the remote system to start the **uucico** program. The conversation between the two **uucico** programs begins with a handshake started by the called, *SLAVE*, system. The *SLAVE* sends a message to let the *MASTER* know it is ready to receive the system identification and conversation sequence number. The response from the *MASTER* is verified by the *SLAVE* and if acceptable, protocol selection begins.

NETWORK

4.2.4 Line Protocol Selection

The remote system sends a message

Pproto-list

where *proto-list* is a string of characters, each representing a line protocol. The calling program checks the *proto-list* for a letter corresponding to an available line protocol and returns a *use-protocol* message. The *use-protocol* message is

Ucode

where *code* is either a one character protocol letter or an *N* which means there is no common protocol.

4.2.5 Work Processing

During processing, one program is the *MASTER* and the other is *SLAVE*. Initially, the calling program is the *MASTER*. These roles may switch one or more times during the conversation.

There are four messages used during the work processing, each specified by the first character of the message. They are

S	send a file,
R	receive a file,
C	copy complete,
H	hangup.

The *MASTER* will send **R** or **S** messages until all work from the spool directory is complete, at which point an **H** message will be sent. The *SLAVE* will reply with **SY**, **SN**, **RY**, **RN**, **HY**, **HN**, corresponding to **yes** or **no** for each request.

The send and receive replies are based on permission to access the requested file/directory. After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message **CY** will be sent if the VENIX **cp** command, used to copy from the spool directory, is successful. Otherwise, a **CN** message is sent. The requests and results are logged on both systems, and, if requested, mail is sent to the user reporting completion (or the user can request status information from the log program at any time).

The hangup response is determined by the *SLAVE* program by a work scan of the spool directory. If work for the remote system exists in the *SLAVE*'s spool

directory, a **HN** message is sent and the programs switch roles. If no work exists, an **HY** response is sent.

4.2.6 Conversation Termination

When an **HY** message is received by the *MASTER* it is echoed back to the *SLAVE* and the protocols are turned off. Each program sends a final “OO” message to the other.

4.3 PRESENT USES

One application of this software is remote mail. Normally, a VENIX system user writes “mail dan” to send mail to user “dan”. By writing “mail usg!dan” the mail is sent to user “dan” on system “usg”.

A primary use of a network is software maintenance. Relatively few of the bytes passed between systems are intended for people to read. Instead, new programs (or new versions of programs) are sent to users, and potential bugs are returned to authors. It's a good idea to implement a “stockroom” which allows remote users to call in and request software. Also a “stock list” of available programs, and new bug fixes and utilities should be added regularly. In this way, users can always obtain the latest version of any program. Although the stock list is maintained on a particular system, the items in the stockroom may be warehoused in many places; typically each program is distributed from the home site of its author. Where necessary, **uucp** does remote-to-remote copies.

Another application of the network for software maintenance is to compare files on two different machines. A very useful utility has been the **diff** program which compares two text files and indicates the differences, line by line, between them. Only lines which are not identical are printed. Similarly, the program **uudiff** compares files (or directories) on two machines. One of these directories may be on a passive system.

To avoid moving large numbers of usually identical files, **uudiff** computes file checksums on each side, and only moves files that are different for detailed comparison. For large files, this process can be iterated; checksums can be computed for each line, and only those lines that are different actually moved.

The **uux** command has been useful for providing remote output. There are some machines which do not have hard-copy devices, but which are connected over 9600 baud communication lines to machines with printers. The **uux** command

NETWORK

allows the formatting of the printout on the local machine and printing on the remote machine using standard VENIX command programs.

4.4 PERFORMANCE

Throughput, of course, is primarily dependent on transmission speed. The table below shows the real throughput of characters on communication links of different speeds. These numbers represent actual data transferred; they do not include bytes used by the line protocol for data validation such as checksums and messages. At the higher speeds, contention for the processors on both ends prevents the network from driving the line full speed. The range of speeds represents the difference between light and heavy loads on the two systems. If desired, operating system modifications can be installed that permit full use of even very fast links.

Nominal speed	Characters/sec.
300 baud	27
1200 baud	100-110
9600 baud	200-850

In addition to the transfer time, there is some overhead for making the connection and logging in ranging from 15 seconds to 1 minute. Even at 300 baud, however, a typical 5,000 byte source program can be transferred in four minutes instead of the 2 days that might be required to mail a tape.

4.5 SYSTEM GOALS

A full system of remote software maintenance is a network goal. The availability of file transfer on a network of compatible operating systems makes it possible just to send programs directly to the end user who wants them. This avoids the bottleneck of negotiation and packaging in the central support group. The "stockroom" serves this function for new utilities and fixes to old utilities. However, it is still likely that distributions will not be sent and installed as often as needed. Users are justifiably suspicious of the "latest version" that has just arrived; all too often it features the "latest bug." What is needed is to address both problems simultaneously:

1. Send distributions whenever programs change.
2. Have sufficient quality control so that users will install them.

To do this, systematic regression testing should be done both on the distributing and receiving systems. Acceptance testing on the receiving systems can be

automated and permits the local system to ensure that its essential work can continue despite the constant installation of changes sent from elsewhere. The work of writing the test sequences should be recovered in lower counseling and distribution costs.

Some slow-speed network services should be implemented, like inter-system **mail** and **diff**, plus the many implied commands represented by **uux**. Also there's a need for inter-system **write** (real-time inter-user communication) and **who** (list of people logged in on different systems). A slow-speed network of this sort is very useful for speeding up counseling and education, even if not fast enough for the distributed data base applications that attract many users to networks. Effective use of remote execution over slow-speed lines, however, is dependent on the general installation of multiplexable channels so that long file transfers do not lock out short inquiries.

Table of Contents

7. DEVICES

intro	introduction to device drivers
async	asynchronous raw I/O
com	communications port
console	Professional 350 console terminal
floppy	floppy disk
lpr	line printer
mem	primary memory
null	data sink
tty	general terminal interface
win	winchester disk

8. SYSTEM MAINTENANCE PROCEDURES

intro	introduction to system maintenance
boot	startup procedures
cron	clock daemon
getty	set typewriter mode
init	process control initialization
makekey	generate encryption key

NAME

intro — introduction to device drivers

DESCRIPTION

VENIX uses two types of device interfaces: block and character. The block interface is buffered and handled in chunks of 512 bytes at a time; this includes standard disk I/O and magtape. The character interface is handled on an unbuffered, byte-by-byte basis; all terminals and other serial I/O lines are handled in such a manner. However, disks and magtape can also be treated in a character (read “unbuffered”) fashion. This is called raw I/O, and is generally faster than regular block I/O.

The most common character device is the terminal, which is explained in detail in **tty(7)**. The rest of this introduction is concerned with the ways in which the system and user deal with disks.

In order to successfully utilize the disk resources of a VENIX system it is important to understand the difference between physical and logical representation of disk regions. A single physical disk may be logically divided by the device driver into a number of partitions, each one of which is treated by the software as a separate pseudo-disk. These pseudo-disks may be situated on any part of any physical drive, and some may overlap others. Each pseudo-disk has a unique pseudo-disk (also called minor device) number, generally between 0 and 7. There is typically one pseudo-disk which represents the entire physical disk volume, while others correspond to portions of a physical volume. The pseudo-disk or minor device number assignments for each disk are found in the individual writeups in this section.

Users of VENIX do not customarily deal with these pseudo-disks by their number. Instead, they refer to file names which appear in the root file system in directory **/dev**. The utility **mknod(1)** is used to create entries in **/dev**: it in effect maps a pseudo-disk to a particular **/dev** file entry. This correspondence can be seen in a **/dev** directory listing, where for each file name, the actual pseudo-disk number appears as the second of two numbers in the field where the file sizes are usually placed. (The first number is the major device number, indicating which disk controller the pseudo-disk actually resides on.)

For purposes of clarity, the standard names for disks in **/dev** have two parts separated by a dot ‘.’, for example **/dev/r10.sys**. The **r10** indicates that the device is part of the RL disk unit number 0. Customarily, the first two letters refer to the type of physical drive (in this case, an RL or RL-emulating drive), and the number to its physical drive number (unit

0). The `sys` indicates that it refers to the pseudo-disk partition used for system programs. Other partitions on the same disk unit would be called `/dev/r10.usr0`, `/dev/r10.usr1`, ... referring to various partitions assigned to individual users, `/dev/r10.tmp` for the temporary areas (used for intermediate files by compilers, editors, and other programs), and `/dev/r10.all` for the complete physical drive. The latter of course overlaps with all the other partitions. Frequently the `.all` is simply dropped from the latter and only the preceding part of the name is used: `/dev/r10`.

Traditionally, file names of the above form are used when doing regular block buffer I/O. If raw, unbuffered I/O is wanted, the file name is given an additional `r` before the name; for example, `/dev/rr10` refers to all of physical drive RL unit 0, to be accessed in raw mode. If asynchronous I/O is used, an `a` instead of an `r` precedes the name: `/dev/ar12` refers to all of physical drive RL unit 2, to be accessed asynchronously. (Raw and asynchronous I/O is always done to complete disk units; see `async(7)`.)

For the purpose of moving around in the VENIX file system, the general user does not even need to know about the device name associated with his or her individual files, since disks devices are attached or mounted (see `mount(1)`) to directories within the root file system. The VENIX distribution comes with several directories in the root file system — `/u0`, `/u2`, and `/u3` — which will be used as the roots of the three user file systems when the system is put into multi-user mode. To keep things simple, the main disk device unit 0 (for example, `/dev/r10`) is usually mounted on `/u0`, disk device unit 1 on `/u1`, and so on, so it is easy to tell what logical disk all files with a given pathname reside on.

In summary, there are three basic levels on which disks are used. At the highest and most general level (on which general users operate), a file's pathname specifies the location of a file on some file system; the actual storage medium involved is of no importance (although the first component of the name may be some clue to the logical device involved). At the next level (of which the system administrator must have some knowledge), we have device file names in the `/dev` directory, which refer to logical disk partitions. Finally, at the bottom level (known only to the device driver), the mapping between logical partitions and their physical location on disk is made.

BUGS

VENIX should have flag-free disk packs.

NAME

async — asynchronous raw I/O

DESCRIPTION

Asynchronous I/O (a process continues running while the I/O transfer is taking place) may be used on any Direct Memory Access (DMA) device. This increases a process's I/O throughput and flexibility, at the expense of more programming and buffer overhead. Typically, asynchronous I/O is useful when multiple queuing of buffers is required (the time between transfers is short) or data comes in sporadic bursts and the process must continue processing.

Asynchronous I/O is restricted to raw mode (i.e. character type special files: the VENIX buffering scheme is by-passed) on DMA devices such as disks, many A/D's and D/A's, etc. The minor device number (see **mknod(1)**) of the special file has the following "magic" values added to get different flavors of asynchronous I/O.

0200 (128)	enable Asynchronous I/O
0100 (64)	send SIGAIO (16) upon I/O completion

For example, suppose the special file `/dev/rrk2` exists for raw I/O on drive 2 of a RK05 disk system and it was created by:

```
/etc/mknod /dev/rrk2 c 11 2
```

The asynchronous version would be created by:

```
/etc/mknod /dev/ark2 c 11 130
```

The **aiowait** and the standard **open**, **seek**, **read**, **write**, etc. system calls are used with asynchronous I/O, with the difference that **read** and **write** system calls immediately return to the caller. The returned transfer count is set as if the transfer actually took place.

It is possible to have the same raw device open for both synchronous (normal) and asynchronous I/O. Keep in mind that raw I/O to disks defeats the VENIX file structure, thus a disk (or disk partition) is generally exclusively used for raw mode I/O.

SEE ALSO

aiowait(2), **signal(2)**, **mknod(1)**
 "VENIX Programming" in the *Programming Guide* for a discussion of asynchronous I/O programming

DIAGNOSTICS

Signal SIGIOT (signal number 6) is sent if there is a device I/O error during asynchronous I/O. It is currently impossible to know anything more about the nature of the error.

BUGS

Since signals are not queued, it is possible to lose a SIGAIO if multiple queuing is used and the queuing rate is faster than the scheduling interval. Thus **aiowait(2)** should be used with fast throughput and multiple queuing.

There is a system dependent maximum number (currently 5) of simultaneously queued asynchronous I/O requests, which if exceeded, causes the caller on the next request to go to sleep until one of the queued asynchronous I/O requests finishes.

NAME

com1, com1.modem, com1.dout — communications port

DESCRIPTION

com1 refers to the asynchronous communications port which behaves as described in **tty(7)**.

com1.modem refers to the identical port, but includes modem control support: a process opening **com1.modem** is suspended until the carrier is detected; the last process closing **com1.modem** causes an automatic hang-up on the modem by clearing data terminal ready.

com1.dout refers also to the same line, but is intended for dial-out purposes. After **com1.dout** is opened, an **ioctl** call to set exclusive-use (TIOCXCL) will suspend any other process reading or writing to the port, and prevent further opening of the line, thereby guaranteeing exclusive use for the process dialing out. This is designed to prevent a login process on **/dev/com1** or **/dev/com1.modem** from interfering with a program (for example **cu(1)**) which is reading and writing to **/dev/com1.dout**. (TIOCXCL calls done on **com1** or **com1.modem** prevent further opening of the line, but do not suspend the activity of other processes which have already opened the line).

Input and output for each line may be set to run at the speeds as indicated in **tty(7)**, with the following exceptions: 200 baud (B200) is not possible, and “External A” (EXTA) maps to 19200 baud.

FILES

/dev/com1	minor device #0
/dev/com1.modem	minor device #64
/dev/com1.dout	minor device #128
/dev/tty	

SEE ALSO

tty(7), **ioctl(2)**

NAME

console — Professional 350 console terminal

DESCRIPTION

The discussion of typewriter I/O given in **tty(7)** applies to this device. However, since the console is not an asynchronous port, attempts to change the speed via **ioctl(2)** are ignored.

The console (known as *pro* in **/etc/termcap**; see **termcap(5)**) is programmable through command sequences sent to it. The console emulates a DEC VT52-type terminal, and provides a number of extended features as well. All commands are preceded by an ESCAPE character and followed by one or more characters of information.

In the following description, ESC stands for the ESCAPE code (ASCII value 033 octal). The character(s) following the ESCAPE are given first alphabetically, and then by their octal ASCII values.

Basic VT52 emulation command set:

ESC H (110)	home cursor
ESC A (101)	cursor up
ESC B (102)	cursor down
ESC C (103)	cursor right
ESC D (104)	cursor left
ESC I (011)	reverse scroll
ESC = (075)	enable alternate keypad mode; the keypad will transmit escape sequences described below rather than 0, 1, etc.
ESC > (076)	reset alternate keypad
ESC F (106)	enable graphics character set
ESC G (107)	reset to normal character set
ESC Y (131) row column	direct cursor addressing add octal 040 to row and column values; the origin (0,0) is the upper left
ESC K (113)	clear to end of line
ESC J (112)	clear to end of page

Extended command set:

In the following description, “X” refers to the single ASCII control value of X.

ESC ^F (006)	write characters in highlight mode
ESC ^E (005)	turn off highlight mode
ESC ^H (006)	write characters in reverse video mode
ESC ^G (005) ⁷	turn off reverse video mode
ESC ^D (006)	write characters in underline mode
ESC ^C (005)	turn off underline mode
ESC ^P (020)	turn on “plotting” mode; cursor disappears, and all characters are written without first clearing cell
ESC ^Q (021)	turn off plotting mode
ESC ^N (016)	“normal” mode: turns off highlight, reverse video, underline, and plotting modes
ESC ^R (022)	set high screen intensity; appropriate only when using color monitor
ESC ^S (023)	set low screen intensity; appropriate only when using monochrome monitor
ESC ^T (024)	draw user-specified character. The following 6 pairs of bytes (12 bytes total) are used to draw the character. Each pair of bytes corresponds to a row of 12 pixels on the screen which are displayed on the screen with the bits ordered in the following way:

```

|0|1|2|3|4|5||0|1|2|3|4|5|
  (byte0)   (byte1)

```

That is, bit 0 (the LSB) of the first byte appears as the left-most pixel of the row; bit 5 of the second byte appears as the right-most pixel of the row. Bit 0200 (octal) must be set in each byte, even though that bit is not displayed. The rows are written from top to bottom. This sequence should not be used in narrow or double-sized characters.

- ESC ^V (026) Enable narrow-character mode. Subsequent characters will be half normal width. This allows 137 characters to appear on a single line.
- ESC ^W (027) Enable double-size/double-height mode. Subsequent characters will be twice normal width and height.
- ESC ^X (028) Enable normal character size mode. Subsequent character written will be normal size. NOTE: all cursor positioning commands position the cursor in units of the current character size.

- ESC c (0143) type Set cursor to mode according to *type*. Types are: 0 (ASCII 060), block cursor; 1 (ASCII 061), blinking block; 2 (ASCII 062), underscore; 3 (ASCII 063), underscore blinking.
- ESC r (0162) Report cursor position in units corresponding to current character size. The row and column are returned, in units identical to that specified by the cursor positioning command.
- ESC s (0163) Show subsequent control characters in the ASCII range 000–040 with special fonts, instead of performing action (e.g. newline (ASCII 012) is displayed as a tiny “nl” and does not perform a newline).
- ESC t (0164) Turn off above mode.

Function key assignment:

<i>Key</i>	<i>Escape Sequence</i>
F1 (Hold Screen)	(Xon/Xoff)
F2 (Print Screen)	ESC b
F3 (Break)	(intrpt sig)
F4 (Set-Up)	ESC d
F5	ESC e
F6 (Interrupt)	^C
F7 (Resume)	ESC g
F8 (Cancel)	^C
F9 (Main Screen)	ESC i

F10 (Exit)	ESC j
F11 (ESC)	ESC
F12 (BS)	backspace
F13 (LF)	linefeed
F14 (Addtnl Options)	ESC n
F15 (Help)	ESC o
F16 (Do)	newline
F17	ESC q
F18	ESC r
F19	ESC s
F20	ESC t
Up Arrow	\$A
Down Arrow	\$B
Right Arrow	\$C
Left Arrow	\$D
Find	\$E
Insert Here	\$F
Remove	\$G
Select	\$H
Prev Screen	\$I
Next Screen	\$J

Pressing the “Compose Character” key alternately enables and disables the graphics character set. The “Compose” lamp is lit when this mode is “on”. When “on”, bit 8 (octal 0200) is set in all codes sent by pressing non-function keys, and in the last of the two or three codes.

Numeric Keypad:

<i>Key</i>	<i>Normal mode</i>	<i>Alternate Mode</i>
PF1	ESC P	ESC P
PF2	ESC Q	ESC Q
PF3	ESC R	ESC R
PF4	ESC S	ESC S
0	0	ESC ?p
1	1	ESC ?q
2	2	ESC ?r
3	3	ESC ?s
4	4	ESC ?t
5	5	ESC ?u
6	6	ESC ?v
7	7	ESC ?w
8	8	ESC ?x
9	9	ESC ?y
,	,	ESC ?l
-	-	ESC ?m
Enter	\n	ESC ?M

Special Control Characters:

<i>Key</i>	<i>Value</i>
CTRL-@	000
CTRL-<SPACE>	000
CTRL-[033
CTRL-\	034
CTRL-]	035
CTRL-^	036
CTRL-_ _	037

The console driver sets the display memory base to physical address 017600000.

FILES

/dev/console
/dev/tty

CONSOLE (7)

CONSOLE (7)

SEE ALSO

tty(7), init(8)

“VENIX Maintenance” in this manual, and *Professional 300 Series Technical Manual*

NAME

floppy — floppy disk

DESCRIPTION

f? refers to an entire diskette as a single sequentially-addressed file. Its logical 512-byte blocks are numbered 0 to 789.

These files access the disk via the system's normal buffering mechanism and may be read without regard to physical disk records. There is also a raw interface which provides for direct transmission between the disk and the user's read or write buffer. A single **read** or **write** call results in exactly one I/O operation, therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw floppy files begin with **rf** and end with a number which selects the same disk as the corresponding **f** file.

In raw I/O the count should be a multiple of 512-bytes: a disk block. Likewise **seek** calls should specify a multiple of 512 bytes.

FILES

/dev/f?, /dev/rf?

BUGS

In raw I/O **read** and **write(2)** truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks.

NAME

lpr — line printer

DESCRIPTION

lpr provides the interface to a printer via the serial printer port. The line behaves as described in **tty(4)**, so it may also be used as an interactive terminal line. Input and output for each line may be set to run at the speeds as indicated in **tty(4)**, with the following exceptions: 200 baud (B200) is not possible, and “External A” (EXTA) maps to 19200 baud.

FILES

/dev/lpr

SEE ALSO

lpr(1)

NAME

mem, kmem — primary memory

DESCRIPTION

mem is a special file that is an image of the primary memory of the computer. For example, it may be used to examine and even patch the system. **kmem** is the same as **mem** except that kernel virtual memory rather than physical memory is accessed.

Byte addresses are interpreted as memory address. Reading non-existent locations finds -1 at those places.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

On PDP-11's, the I/O page begins at location 0160000 of **kmem** and per-process data for the current process begins at 0140000.

FILES

/dev/mem, /dev/kmem

BUGS

On PDP-11's, memory files are accessed one byte at a time, an inappropriate method for some device registers.

NULL(7)

NULL(7)

NAME

null — data sink

DESCRIPTION

Data written to a null special file is callously discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

tty — general terminal interface

DESCRIPTION

This section describes both a particular special file, and the general nature of the terminal interface.

The file `/dev/tty` is, in each process, a synonym for the control terminal associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

As for terminals in general: all of the low-speed asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of the interface.

When a terminal file is opened, the system causes the process to wait until a connection is established. In practice users' programs seldom open these files; they are opened by **init** (see **init(8)**) and become a user's input and output file. The very first terminal file open in a process becomes the control terminal for that process. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a **fork**, even if the control terminal is closed. The set of processes that thus share a terminal is called a process group; all members of a process group receive certain signals together (see `^C` below and **kill(2)**).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even when output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. When the input limit is reached all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not however necessary

to read a whole line at once; any number of characters may be requested in a read, even one, without losing information. (Once the user has typed the newline character, the line is buffered internally and can be read in any size chunks). There are special modes, discussed below, that permit the program to read each character as typed without waiting for a full line.

During input, erase and kill processing is normally done. By default, the 'delete' character erases the last character typed, except it will not erase beyond the beginning of a line or EOT. If the terminal is a CRT, then an attempt is made to remove the preceding character (by a backspace, space, and backspace sequence); otherwise a '<' is echoed. By default, the ^U character (CONTROL U; typed by depressing the 'control' key while typing 'u') is set as the kill character: it kills the entire line up to the point where it was typed, but not beyond the beginning of the line or EOT. Both the delete and kill characters may be changed by the user.

When desired, all upper-case letters may be mapped into the corresponding lower-case letter on input, and everything echoed to the terminal in upper-case only. Real upper-case letters may be generated by preceding it by an 'escape', which echos as '\$'. In addition, the following escape sequences can be generated on output and accepted on input:

<i>for</i>	<i>use</i>
`	\$'
	\$!
~	\$^
{	\$(
}	\$)

See the **LCASE** flag below.

Certain ASCII control characters have special meaning. These characters are not passed to a reading program except in raw mode when they lose their special meaning.

^D (EOT) may be used to generate an end-of-file from a terminal. When an EOT is received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOT is discarded. Thus if there are no characters waiting, which is to say the EOT occurred at the beginning of a line, zero characters will be passed back, and this is the standard end-of-file indication.

- ^C** is not passed to a program but generates an interrupt signal which is sent to all processes with the associated control terminal. Normally each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. See **signal(2)**. All typed characters not yet read are killed.
- ^Z** generates the **quit** signal. Its treatment is identical to the interrupt signal except that unless a receiving process has made other arrangements it will not only be terminated but a core image file will be generated.
- ^S** delays all printing on the terminal until anything is typed in; it is not echoed. Usually a **^Q** is used to start output again.
- ^Q** restarts printing after a **^S**, or, in **SCROLL** mode, after screen output stops; it also is not echoed.
- ^R** reviews (re-echos) all typed characters that have not yet been read by any program.
- ^E** kills all typed in characters that have not yet been read by any program.

BREAK

Identical to '**^C**', but works even in **RAW** mode and does not kill all unread characters.

When the carrier signal from a dataset drops (usually because the user has hung up his remote terminal) a **hangup** signal is sent to all processes with the terminal as their control terminal. Unless other arrangements have been made, this signal causes the process to terminate. If the hangup signal is ignored, any read returns with an end-of-file indication. Thus programs that read a terminal and test for end-of-file on their input can terminate appropriately when hung up on.

Terminal I/O is buffered: when characters are written to the terminal, they are actually sent as soon as previously-written characters have finished being typed; input characters are echoed by putting them in the output queue as they arrive. When a process produces characters more rapidly than they can be sent out, it will be suspended when its output queue exceeds some limit. When the queue has drained to some threshold the program is resumed.

Several **ioctl(2)** calls apply to terminals. Most of them use the following structure, defined in `<sgtty.h>`.

```

struct sgttyb {
    char    sg_ispeed;
    char    sg_ospeed;
    char    sg_erase;
    char    sg_kill;
    int     sg_flags;
};

```

The `sg_ispeed` and `sg_ospeed` fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface (and closely to the DZ-11). If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are defined in `<sgtty.h>`.

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

Currently, only 300 and 1200 baud are really supported on dial-up lines. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supported; full duplex 212 datasets work fine. Speed cannot be changed on terminals attached to a DL-11 or KL-11; the VENIX console, whose special use is described in **boot(8)**, is one such terminal.

The `sg_erase` and `sg_kill` fields of the argument structure specify the erase and kill characters respectively. (Defaults are 'delete' and ^U.)

The `sg_flags` field in the argument structure contains several bits that determine the system's treatment of the terminal:

CRT	0100000	Terminal is a CRT
SCROLL	0040000	Output stops automatically every 20 lines
XTABS	0006000	Expand tabs to spaces on output
RAW	0000040	Raw mode: 8 bit interface (turns off CRT, XTABS, CRMOD, LCASE and CBREAK)
EVENP	0000200	Enables even parity
ODDP	0000100	Enables odd parity
CRMOD	0000020	Map CR into LF; echo LF as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper to lower case (Escapes work)
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Automatic flow control

The undefined bits are used on some UNIX systems to specify delays on certain characters and parity; these delays and parity are currently ignored under VENIX.

Several **ioctl** calls have the form:

```
#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;
```

The applicable codes are:

TIOCGETP Fetch the parameters associated with the terminal, and store in the pointed-to structure.

TIOCSETP Set the parameters according to the pointed-to structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN Set the parameters but do not delay or flush input.

TIOCEXCL Set "exclusive-use" mode: no further opens are permitted until the file has been closed.

TIOCNCXCL Turn off "exclusive-use" mode.

TIOCHPCL When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.

TIOCFLUSH

All characters waiting on input or output are flushed.

TIOCQCNT Returns the count of characters currently typed in but not yet read in `sg_ispeed` (0 to 255) and the count of characters on the output queue, which is probably rapidly changing, in `sg_ospeed` (0 to about 100).

FILES

<code>/dev/tty</code>	the user terminal
<code>/dev/tty*</code>	all terminals on the system
<code>/dev/console</code>	the console terminal

SEE ALSO

`stty(1)`, `signal(2)`, `ioctl(2)`, `getty(8)`
 “VENIX Programming” in the *Programming Guide* for an example of using `ioctl`.

NOTES

The following differences are noted between the VENIX tty handler and that in some other versions of UNIX:

1. Reading in RAW mode returns the number of characters asked for, not just one as in other versions.
2. The SCROLL mode and TIOCQCNT command (return character count in buffer) are not supported by some other versions.
3. Delay mode for output of certain characters is not supported by VENIX.
4. The interrupt character (^C), delete character (DEL) and kill character (^U) are different than in some other versions. (The delete and kill characters, however, can be changed by `ioctl` calls or the `stty` command.)

BUGS

Half-duplex terminals are not supported.

The deleting of characters on a CRT cannot be exact, due to the interspersing of input and output. Furthermore, ‘delete’ assumes that one character’s position is to be removed, inappropriate for tabs and

TTY(7)

TTY(7)

control characters. If there is confusion as to exactly what has been deleted, the re-echo (^R) feature should clear it up.

NAME

w0.* — winchester disk

DESCRIPTION

w0.sys, **w0.tmp**, and **w0.usr** refer to sections of the 5 or 10mb winchester disk drive.

Minor no.	Starting block	Length in blocks	Usage	Suggested name
0	64	768	tmp & pipes	/dev/w0.tmp
1	832	2944	system area	/dev/w0.sys
2	3776	832	swap area	
3	4608	14976	user (5184 w/5mb)	/dev/w0.usr

The partitions are the same size on either the 5 or 10mb disks, except for the user area.

The **w0** files discussed above access the disk via the system's normal buffering mechanism and may be read without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation, therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw winchester files begin with **rw0** and end with a name which selects the same disk section as the corresponding **w0** file. Additionally, raw winchester files can be accessed asynchronously (**async(7)**) by using the name **aw** followed by a name which selects the same disk section as the corresponding **rl** file.

In raw I/O the buffer must begin on a word boundary.

FILES

/dev/w0*, /dev/rw0*, /dev/aw0*

BUGS

In raw I/O **read** and **write(2)** truncate file offsets to 512-byte block boundaries, and **write** scribbles on the tail of incomplete blocks.

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section contains descriptions of programs used by VENIX itself for initialization and background maintenance. These programs are not intended to be invoked by users, but to be run by the kernel, by other commands, or by the `/etc/rc` command file.

NAME

boot — startup procedures

DESCRIPTION

The bootstrap must reside in the otherwise unused block zero of the boot device. It can be read in and started by the standard ROM program. This bootstrap is capable of loading type 407 executable files (not separate I&D). The kernel `‘/venix’` is loaded and executed. If `‘/venix’` is not found, the four lamps on the Pro keyboard are lit. If a disk error is detected, the two left-hand lamps are lit.

After VENIX has started running and sized memory, it prints a copyright message and the amount of memory available for users in kilobyte units. The root and pipe file systems are then mounted. `/etc/init` starts running, which in turn opens `/dev/console` and starts `/bin/sh`. An `fsck` will then run automatically to check the partitions on the hard disk.

FILES

`/venix`

SEE ALSO

`init(8)`, `a.out(4)`

NAME

cron — clock daemon

SYNOPSIS

/etc/cron [*min*]

DESCRIPTION

cron executes commands at specified dates and times according to the instructions in the file **/usr/lib/crontab**. Since **cron** never exits, it should only be executed once. This is best done by running **cron** from the initialization process through the file **/etc/rc**.

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0–59), hour (0–23), day of the month (1–31), month of the year (1–12), and day of the week (1–7 with 1=monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. (It is not permissible to combine a list and a range in one field.) The sixth field is a string that is executed by the shell at the specified times. A percent character in this field is translated to a new-line character.

Crontab is examined by **cron** every *min* minutes, or 1 if no number is specified on the command line. It sets its effective ID to 1 (“sys” user) while running.

FILES

/usr/lib/crontab

NAME

getty — set typewriter mode

SYNOPSIS

/etc/getty [*char*]

DESCRIPTION

getty is invoked by **init(8)** immediately after a typewriter is opened following a dial-up. It reads the user's login name and calls **login(1)** with the name as an argument. While reading the name **getty** attempts to adapt the system to the speed and type of terminal being used.

init calls **getty** with a single character argument taken from the */etc/ttys* file entry for the terminal line. This argument determines a sequence of line speeds through which **getty** cycles, and also the 'login:' greeting message, which can contain character sequences to put various kinds of terminals in useful states.

The user's name is terminated by a newline or carriage-return character. In the second case CRMOD mode is set (see **ioctl(2)**), so that future carriage-returns will be treated as newlines.

The name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

If the terminal's 'break' key is depressed, **getty** cycles to the next speed appropriate to the type of line and prints the greeting message again.

Finally, **login** is called with the user's name as an argument. See **ttys(4)** for a list of terminal options in the */etc/ttys* file.

SEE ALSO

init(8), **login(1)**, **ioctl(2)**, **ttys(4)**

NAME

init, **rc** — process control initialization

SYNOPSIS

/etc/init

/etc/rc

DESCRIPTION

init is invoked as the last step of the boot procedure. Generally its role is to create a process for each typewriter on which a user may log in.

When **init** first is executed the console typewriter **/dev/console**. is opened for reading and writing and the shell is invoked immediately. This feature is used to execute **fsck**. When the shell terminates, **init** comes up multi-user and the process described below is started.

When **init** comes up multiuser, it invokes a shell, with input taken from the file **/etc/rc**. This command file performs housekeeping like removing temporary files, mounting file systems, and starting daemons.

Then **init** reads the file **/etc/ttys** and forks several times to create a process for each typewriter specified in the file. Each of these processes opens the appropriate typewriter for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input, output and error files. Opening the typewriter will usually involve a delay, since the open is not completed until someone is dialed up and carrier established on the channel. Then **/etc/getty** is called with argument as specified by the last character of the **ttys** file line. **getty** reads the user's name and invokes **login(1)** to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of **init**, which has been waiting for such an event, wakes up and removes the appropriate entry from the file **utmp**, which records current users, and makes an entry in **/usr/adm/wtmp**, which maintains a history of logins and logouts. Then the appropriate typewriter is reopened and **getty** is reinvoked.

FILES

/dev/tty?, **/etc/utmp**, **/usr/adm/wtmp**, **/etc/ttys**, **/etc/rc**

SEE ALSO

login(1), **kill(1)**, **sh(1)**, **ttys(4)**, **getty(8)**

NAME

makekey — generate encryption key

SYNOPSIS

`/usr/lib/makekey`

DESCRIPTION

makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e. to require a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two (the salt) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

makekey is intended for programs that perform encryption (e.g. **ed(1)** and **crypt(1)**). Usually its input and output will be pipes.

SEE ALSO

crypt(1), **ed(1)**

Printed in U.S.A.

AA-BM31A-TH